

Analyse syntaxique et intelligence artificielle

ou

Les aventures de Syntaxikos.

pièce en quatre actes et en prose

programmes en basic, en pascal et en prolog.

Avis au lecteur

La simulation du comportement humain sous entend d'une part un mode de représentation des connaissances construit autour de quelque chose qui ressemble à un dictionnaire encyclopédique, et d'autre part un mécanisme permettant de déduire de nouveaux faits à partir des informations ou règles contenues dans ce dictionnaire et à partir des faits contenus dans une base de données décrivant l'état du monde à étudier. Cet outil de base, le moteur d'inférence, peut alors être employé à de multiples usages, chacun constituant un système expert. Un tel système est toujours jumelé à un interface de dialogue homme-machine qui permet de rajouter des règles dans le dictionnaire et des faits dans la base de donnée. La simplicité de cet interface semble maximum si la procédure utilisée est la langue naturelle elle même, ce qui pose évidemment le problème de l'analyse des langues naturelles mais aussi celui de la génération des réponses et ce dernier problème est moins trivial qu'il n'y paraît.

Ce livre présente quelques exemples basés sur des jeux : un petit système capable de répondre à des questions en tenant des raisonnements simples, un analyseur de thesaurus, un analyseur de grammaire sous forme de bakus appliqué au pascal puis au français, un jeu d'aventure acceptant des phrases construites autour d'un dictionnaire contenant les descriptions syntaxique et sémantique d'environ 650 mots, un programme qui invente des contes et des poèmes, un autre qui joue aux devinettes et devine ce à quoi l'on pense, un autre qui dessine des moutons, des courbes en coordonnées polaires ou cartésiennes, des flocons de neige et toutes sortes de fractales, un autre qui parle javanais, invente des mots et des jeux de mots, un autre qui permet de s'y retrouver dans une classification, un autre encore qui, par analyse des états initiaux et finaux d'une part et des actions possibles d'autre part, résout plusieurs problèmes classiques comme les tours de Hanoï, le problème du singe et de la banane, et le problème de la chèvre du choux et du loup, le problème des 8 dames, le problème du coloriage d'une carte. Un autre qui recherche dans un réseau un chemin optimum selon plusieurs critères. Un autre qui parcourt des files, des tables de hsh-code et des arbres binaires. Un système capable de dériver ou intégrer des expressions mathématiques et de résoudre des équations, Une grammaire réduite du français sous forme de règles Prolog, et un programme qui l'utilise pour traduire les ordres et les questions adressées à un robot manipulant des objets. Un programme gérant un arbre généalogique. Un programme de gestion d'une base de données contenant un compilateur pour le langage SQL d'interrogation de la base, un système expert basé sur des règles de production du type si .. alors ..

Les programmes exposés sont écrits en basic ("basica" d'Ibm), en pascal ("turbo pascal" delphi), ou en Prolog (une version personnelle). Ils ont été mis au point sur un Ibm Pc mais ils n'utilisent aucune fonction spéciale qui poserait problème pour le passage sur une autre machine : pas de musique, pas de graphique.

Ecrits dans les années 1970, il n'y a aucune utilisation des « modèles de langages » bien plus récents.

Extrait du catalogue de nos personnages

le Grand Syntaxiqueurune pièce unique, en argent massif.
sur commande numérotée exclusivement.
référence : AK-47

le 1er courtisan modèle standard, dit vulgum pécus.
matricule : 837-1515-1610-1715

le 2ème courtisan vil, népotique et choc.
Sonorité à la tête du client.
référence :
de DO-D-K-FO-NIK
à SI-6-6-6-6-gare-606-6-6-606-6-gare

des chambellans très bel ensemble E à dessiner en
rouge comme étant sous ensemble de
l'ensemble de référence, à dessiner en
bleu, et dont on demande le cardinal.
durée : 3 heures - coefficient : 4

Syntaxikos le héro au sourire si doux.
Existe en deux tailles: debout, couché
N°. de sécurité sociale: 1481054395105

Sémantika l'héroïne au man.
en polyorildur de metynol hydrafisé.
mensurations : 94-62-96. macarel !

Syllogismus hypotheticus . co-drinking friend.
mettra de la gaieté dans votre vie.
Bonne année : BROUILLY-1948

Basic ami d'enfance de Syntaxikos.
soldé pour cause de changement de
propriétaire. exemple: 130 GOTO 1020

Pascallina borlandini di turbo. bonne à tout faire, polonaise.
elle fera l'admiration de vos amis.
adresse : PC-\$09DE

Prologistos valet pensant de Syntaxikos.
En options : les bretelles et l'étui.
référence : SYMBOLIQUE

un robot valet actant de Syntaxikos.
inusable, en promotion pendant 3 mois à
dater de six pour les maris des veuves
réduction : 23% ht

Dictionnaria précédente de Sémantika.
En échangerait un petit illustré contre
Deux encyclopédiques en trois volumes.
téléphoner : PIN-PON-18-22

Sphinx à utiliser à jeun, au saut du lit.
heure : 8-h-30

Base de Donnée une jeune et jolie aristocrate.
livrée aussi sans la particule et
en kit, à remonter dans nos locaux
au temple. référence : BD-1789

Bakus le seul bon Dieu de la bande
<attribut> := <vin> / <grappe>

Acte I :

RIGOLO, RIGOLAM, RIGOLAMUS

if you = pas rigolo goto 2ème partie

Quelqu'un, le grand syntaxiqueur.
le syntaxiquatoir est grand ouvert, quelqu'un passe.

Le grand syntaxiqueur :

- Eh ! vous là-bas! oui, vous, eh! attention !!!
ah ca y est, il est tombé dedans - c'est bien
fait.

Il se penche, on entend le ronflement du syntaxiquatoir
syntaxiquant.

Scène 1

un psychologue

Décidément très atteints, nos héros se rendent chez un psychanalyste.

1.1 Principe de l'analyse

C'est un classique de l'intelligence artificielle. (dites merci à Joseph Weizenbaum. Allez, mouchez vot'nez et dites merci au monsieur) Le programme joue le rôle d'un psychiatre et soutient un dialogue avec un patient. Très spectaculaire, quoique ce soit un petit programme et qu'en fait il ne comprenne rien du tout. Il utilise un fichier de données contenant des mots clés et des moules de réponse à choisir aléatoirement selon le mot clé. Les moules de réponse contiennent des variables, en petit nombre, dont l'une sert à glisser le nom du "malade" ou un "bibiche" moqueur dans la réponse, une autre sert à rajouter au moule la fin de la phrase du "malade" en lui faisant éventuellement quelques modifications ("tu" devient "je" ,"mon" devient "ton" ... et réciproquement). Les mots clés peuvent avoir un attribut précisant qu'ils doivent être supprimés de la table après leur première, et donc unique, utilisation; Ils peuvent aussi avoir l'attribut de mémorisation c'est à dire que la phrase du "malade" est mémorisée, on la ressortira quand on ne saura pas quoi dire. Une autre solution quand on ne sait pas quoi dire étant de poser une question directive, pouvant amener un des mots clé prévus.

1.2 Exemple de dialogue

Les phrases marquées ? sont celles frappées par Syntaxikos. ou alors c'est l'inverse, je ne sais plus ! et puis de toute façon, il faut respecter le secret médical.

vous habitez chez vos parents ?
 ? *vous perdez votre temps*
on parle de vous, pas de moi
 ? *j 'ai envie de vous gifler*
depuis quand avez-vous envie de me gifler ?
 ? *depuis cinq minutes*
et dans votre enfance
 ? *ca aurait été pareil*
est ce que cela a un rapport avec le fait que je perds mon temps ?
 ? *certainement*
je n'en suis pas si sûr que vous

1.4 Données de Psy - psy.txt

le fichier est constitué d'un ensemble de lignes, chaque ligne contient un descriptif de mot clé ou un descriptif de réponse. Le premier caractère des lignes spécifiant le type de mot clé ou le type de réponse.

Le fichier est organisé en séquences : il y a plusieurs réponses possibles à un mot clé, le mot clé, et ses synonymes, sont en fin de séquence.

Le premier caractère des lignes mot clé a les codes suivants :

- 0,1 ce mot clé est effacé après emploi de toutes les réponses
 0 pour le premier mot clé d'un groupe, 1 pour les autres
- 6,7 ce mot clé peut resservir
 6 pour le premier mot clé d'un groupe, 7 pour les autres
- m placé en tête de séquence, indique que la phrase doit être mémorisée, on

la ressortira pour relancer le dialogue.

l'ordre dans lequel on place les mots clé n'est pas indifférent, car si une phrase en contient plusieurs, c'est celui le plus proche du début de fichier qui sera utilisé pour générer une réponse. on place donc en tête du fichier les mots clé spécifiques (vacances, voiture,...), ayant souvent l'attribut d'utilisation unique, et en fin du fichier les mots clé généraux (je pense, je veux, ...).

Lors des réponses, les caractères & ou \$ sont remplacés par le nom du joueur, ou « bibiche »

Le premier caractère des lignes réponse a les valeurs suivantes ::

- 2 réponse simple
- 3 réponse suivie de la fin de la phrase retournée mot à mot
 je dis que j'ai faim
 devient depuis quand tu as faim

la première séquence de messages est utilisée pour répondre en utilisant l'une des phrases du malade que l'on a mémorisé (lorsque le mot clé contenu dans cette phrase a l'attribut « m » en début de séquence), les deux dernière séquences sont utilisés lorsque aucun mot clé n'a été reconnu.

*

* moules pour ressortir les phrases mémorisées

*

3,est-ce que cela a un rapport avec le fait que

3,pourquoi disais-tu que

3,\$, il y a une chose qui m'a frappé :

3,qu'est ce qui t'a fait dire que

3, \$, est-ce vrai que

0,\$\$\$

*

* réponses correspondant à un mot clé unique : argent

*

2,et qu'est ce que tu ferais si tu gagnais au loto

2,tu en as bien assez comme ça

6,argent

*

* réponses correspondant à un mot clé ayant des synonymes : vacances, congé

*

2,ah les baléares ... les cocotiers ...

2,oui, allez fous moi le camp et que je ne te revois plus

0,vacances

1,congé

*

2,parles moi de ta famille

2,es tu heureux \$

6,famille

7,père

7,mère

7,fil

7,fil

7,frère

7,soeur

7,filles

2,tu préfères les femmes ou les hommes

2,les femmes, c'est un problème pour toi

2,et le sexe

6,femme

7,femmes

*

2,et les vacances, comment les prends tu

2,comment passes tu tes week end \$

0,loisir

1,loisirs

1,passé temps

*

2,le sexe, c'est un problème qui te tourmente

2,tu y vas tous les combien &

2,tu es en manque &,je préfère passer derrière toi

6,sexe

*

2,chacun ses goûts \$

6,hommes

2,c'est un vrai veau

2,c'est une voiture potable

6,renault

7,r4

7,r5

7,peugeot

7,citroen

7,simca

7,fiat
2,de quelle marque
2,ca ne consomme pas trop
0,voiture
1,auto
2,moi aussi
0,train
2,je t'en prie, reste poli
2,si tu continues sur ce ton, le ton va monter entre nous
2,espece de \$
2,attention, je ne suis pas du genre à me laisser insulter
2,sale \$
2,espece d'andouille
2,pauvre petit con
2,non mais sans blague
2,c'est pas possible, il se croit tout permis
0,merde
1,con
1,idiot
1,zut
1,gueule
1,enmerde
1,foutre
1,fouttre
1,abruti
1,taré
1,débil
1,débile
1,patate
2,pourquoi amène tu la discussion sur l'amitié
2,\$ est-ce que tes amis t'ennuient
2,est ce que tes amis profitent de toi
2,es tu sur d'avoir des amis
2,quel est le métier de tes amis
6,amis
7,ami
7,amitié
7,amie
2,comment ça va, quel est ton problème \$
2,ca fait une heure qu'on discute et tu me dis bonjour
0,bonjour
2,rêves tu souvent
2,es tu perturbé par tes rêves
2,quelles personnes apparaissent dans tes rêves
2,raconte moi ce rêve
2,à quoi ces rêves te font-ils penser
2,il n'y a jamais de femmes dans tes rêves

6,rève
 7,rêves
 7,cauchemar
 7,cauchemars
 2,le match, faut sortir mon vieux
 2,y jouent comme des pieds
 2,et t'as vu ce but
 6,match
 2,et bien tu vois \$, ton cas n'est pas désespéré
 0,blanc
 2,effectivement \$,ton cas est grave
 0,noir 1,bleu 1,rouge 1,vert 1,marron 1,orange 1,rose
 1,violet
 1,jaune
 2,ah chapeau ce but
 6,but
 2,tu aimes les enfants
 0,enfants
 2,les excuses ne sont pas nécessaires
 2,s'il te plaît, ne t'excuse pas \$
 2,à quoi penses tu quand tu t'excuses
 2,ne sois pas si défensif
 0,désolé
 1,excuse
 1,pardon
 1,excuses
 2,essaye d'analyser ce sentiment
 6,sentiment
 2,on dit ça ...
 2,est ce bien raisonnable
 6,problème
 7,problèmes
 7,difficulté
 2,ou la la, ca va très très mal
 2,tu devrais essayer mennie grégoire
 0,suicide
 1,suicider
 1,mort
 2,comment ça a commencé
 2,et ça te prend comment
 2,mais ce n'est pas raisonnable
 2,ça a des répercussions sur ta santé
 6,angoisse
 7,angoisses
 7,honte
 7,ennerve
 7,excite

2,est-ce la vraie raison
 2,peut être y a t-il une autre raison
 2,est ce que cela n'explique pas d'autres troubles
 6,cause
 7,causes
 3,pourquoi veux tu savoir si j'ai
 3,ca ne te regarde pas de savoir si j'ai
 0,as tu
 3,qu'est-ce qui te fait dire
 0,je me dis
 3,pourquoi dis tu
 0,je dis
 1,je te dis
 3,ca te prend souvent,\$ d'avoir envie
 3,depuis quand as-tu envie
 m
 0,j'ai envie
 2,ah oui c'est vrai,\$ je ne m'y retrouve plus dans mes notes
 2,c'est pour vérifier si tu ne me racontes pas des coups
 0,déjà dit
 3,comment sais-tu que tu ne peux pas
 2,as-tu essayé
 2,peut être que maintenant tu le pourrai \$
 3,penses tu que tu devrais pouvoir
 3,mais \$ pourquoi ne peux-tu pas
 3,veux tu que je
 3,voudrais-tu être capable de
 3,est ce que cela te gêne \$ de ne pas pouvoir
 m
 0,je ne peux plus
 1,je ne peux pas
 3,mais bien sur & que je sais
 3,moi oui, mais toi, tu sais
 0,sais tu
 1,connais tu
 3,effectivement \$, c'est ennuyeux de se sentir
 3,depuis quand te sens tu
 m
 0,je me sens
 3,ah oui, c'est classique, moi aussi je sens
 m
 0,je sens
 3,tu crois vraiment que je suis incapable de
 3,pourquoi penses-tu que je suis infoutu de
 0,tu ne peux pas
 2,c'est une question délicate
 2,je me suis moi aussi posé cette question

0,suis je
 2,ce n'est pas moi qui as des problèmes \$, c'est toi
 3,pourquoi es-tu intéressé de savoir si je suis ou non
 3,prefererai-tu que je ne sois pas
 0,es tu
 1,êtes vous
 3,c'est à toi de décider si tu peux
 2,tu es assez grand pour le savoir
 2,ce n'est pas à moi de répondre \$
 3,aimerai-tu réellement
 0,puis je
 3,tu veux que je
 3,tu crois que je ne pourrai pas
 0,peux tu
 1,pourrai tu
 1,pouvez vous
 3,est-ce que parfois, tu ne voudrais pas être
 3,peut être aimerai-tu être
 3;est-ce que ca te soulage de penser que je suis
 3,qu'est-ce qui te fait dire que je suis
 0,tu es
 1,vous êtes
 3,es-tu venu me voir parceque tu es
 3,combien de temps as-tu été
 3,penses-tu qu'il est anormal d'être
 3,aises-tu être
 3,et parmi tes amis, il y en a un qui est
 m
 0,je suis
 3,depuis combien de temps n'es-tu plus
 3,ne l'est tu pas encore, parfois
 m
 0,j étais
 1,j ai été
 3,moi aussi, souvent je veux
 3,pourquoi veux-tu
 3,d'accord, supposons que tu arrives à
 m
 0,je veux
 1,je voudrai
 2,quelle est la différence
 3,pourquoi préfère-tu
 m
 0,je préfère
 1,je préférerai
 3,tu penses vraiment \$
 2,mais tu n'en es pas sur

0,je pense
 1,je suppose
 3,est-ce seulement pour toi que tu aimes
 2,tu dis ça sans conviction \$
 3,pourquoi aimes-tu
 m
 0,j aime
 3,c'est physique ou c'est psychique que tu n'aimes pas
 2,on va approfondir ce point,
 3,n'as tu jamais aimé
 3,depuis quand n'aimes-tu pas
 m
 0,je n aime pas
 1,je n aime plus
 1,je déteste
 3,depuis combien de temps as-tu peur
 2,est-ce vraiment justifié
 3,comment combats tu cette peur
 2,en ce moment, as-tu peur de quelquechose
 m
 0,j ai peur
 3,pourquoi t'intéresses-tu à ma
 2,et la tienne
 0,ta
 3,est-ce normal de s'intéresser à mon
 2,et le tien
 0,ton
 1,votre
 2,pourquoi pas
 2,dis-tu cela par simple esprit de contradiction
 2,en es-tu vraiment si sur que cela
 2,je note, je vérifierai plus tard
 0,non
 2,je commences à comprendre
 2,je comprends
 2,c'est peut être pas si évident
 2,tu dis ça pour me faire plaisir
 2,parfait
 2,je vois
 0,oui
 1,surement
 2,je n'en suis pas si sur que toi
 2,ne sois pas si agressif, ça te fait du mal
 2,disons plutôt peut être
 2,sans doute
 0,si
 2,tu ne sais pas

2,tu ne me sembles pas bien sur de toi
 3,il faudrait s'en assurer
 2,alors, cela mérite que l'on s'y attarde
 0,peut être
 2,sans exception
 2,tu exagères
 2,il y a quand même des exceptions
 2,cites-moi un exemple
 2,quand
 0,toujours
 2,et dans ton enfance
 2,pendant ton adolescence aussi
 0,depuis
 2,tu es sur
 2,quelle ressemblance vois-tu
 2,qu'est-ce que la similitude te suggère
 2,quelle autre connexion vois-tu
 2,de quelle manière
 0,semble
 1,ressemble
 1,comme
 2,veux-tu un autre rendez-vous pour continuer cette thérapie
 2,c'est peu
 2,ne sois pas si tranchant et agressif
 0,jamais
 2,au revoir
 2,salut
 0,au revoir
 2,ce genre de question de tourmente-t-elle souvent
 2,l'as-tu demandé à quelqu'un d'autre
 2,cette question t'intéresse t'elle ou dis-tu cela pour meubler
 2,quelle réponse te plairait le mieux
 2,le psychanalyste, c'est moi, c'est moi qui pose les questions
 4,cela ne t'apportera rien que je te dise
 4,et si je te posai cette question, que répondrai-tu
 2,si on te le demande, tu diras que tu ne sais pas
 4,tes amis te demandent-ils
 4,cela ne m'intéresse pas de savoir
 4,il n'est pas bon que tu ne découvre pas toi même
 0,pourquoi
 1,combien
 1,comment
 1,quand
 1,qu'est ce
 1,où est
 1,qui est
 1,quelle est

1,quel est
 1,quels sont
 1,quelles sont
 2,tu as vu le match hier
 6,foot
 7,basket
 7,rugby
 7,natation
 7,marche
 2,quel est ton braquet préféré, le 52 13
 6,vélo
 2,petit canaillou
 6,grimpeur
 2,quel est le sujet du livre que tu lis en ce moment
 6,lecture
 7,histoire
 3,depuis quand as-tu
 2,crois-tu être le seul dans ce cas
 m
 0,j ai
 3,mais ce n'est pas une tare de ne pas avoir
 3,et voudrais tu avoir
 m
 0,je n'ai pas
 1,je n ai plus
 2,course à pied
 0,cheval de course
 2,selle de cheval
 0,bout de ficelle
 2,marabout
 0,j en ai marre
 2,tu ne parles pas réellement de moi, n'est ce pas
 2,on parle de toi, pas de moi
 3,oh je
 0,tu
 1,vous
 *
 * quelques questions directives ne servant qu'une fois
 * quand on ne sais plus trop quoi dire
 *
 2,quelle est la couleur du cheval blanc d'henri quatre
 2,qu'est ce que tu as comme voiture
 2,comment vas-tu travailler
 2,tu habites chez tes parents
 2,au fond, je te trouve assez sympa
 2,tu ne veux pas diner avec moi ce soir
 2,t'as une bonne petite tête

2,écoute, il faut que je te dise, je t'aime
 2,quand pourrions-nous ...
 2,quel sport pratiques-tu
 2,tu ne m'as pas dis quel était ton métier
 2,et tes enfants, parles-moi de tes enfants
 2,quel est ton passe temps
 2,qu'est-ce que tu déteste le plus
 2,qu'est-ce que tu aime le plus
 0,xxx
 *
 * quelques phrases, quand on ne sait plus quoi dire
 *
 2,peux-tu t'étendre sur ce sujet
 2,allez, vas-y, dis-moi tout
 2,je ne suis pas sur d'avoir compris
 2,déballes ton sac, ça va te soulager
 2,c'est très intéressant
 2,détends toi
 2,qu'est-ce que cela te suggère
 2,tu m'intéresse énormément, causes toujours
 2,parles moi de toi, de tes passions
 0,xxxx
 *
 * liste des mots à échanger pour retourner les phrases
 *
 5,je,tu
 5,tu,je
 5,j,tu
 5,vous,je
 5,me,te
 5,te,me
 5,m,t
 5,t,m
 5,as,ai
 5,ai,as
 5,avez,ai
 5,suis,es
 5,es,suis
 5,êtes,suis
 5,mon,ton
 5,ton,mon
 5,ma,ta
 5,ta,ma
 5,moi,toi
 5,toi,moi
 5,dites,dis
 9,tttt

Voir les fichiers psy.pas, psymais.pas et psy.txt

un programme écrivain

A grands coups d'aléa leur vint la poésie
si bellement que de leur cérébrale industrie
le fruit nous est plus bas par le menu conté
en sorte que chacun pourra s'y essayer
et s'y désennuiera.

1.1 Les cadavres exquis

Le programme invente des poèmes, des proverbes, des contes, des pièces de théâtre, des titres pour les journaux à sensation, des archives familiales. Il est basé sur le principe des "cadavres exquis", est écrit en pascal et utilise un fichier de données contenant des moules pour les différents textes créés. Le programme choisit aléatoirement un moule, lequel contient des bribes de texte et des parties variables judicieusement disposées in the middle of it. Les parties variables sont remplacées en allant puiser aléatoirement dans un lot de bribes de texte correspondant à la catégorie de la partie variable, et icelle bribe de texte peut, elle-même, contenir des parties variables. Les parties variables sont définies par des règles qui sont de deux formes :

<nom de la partie variable>=	
ligne de texte	1ère possibilité de remplacement
...	
ligne de texte	Nème possibilité de remplacement
<nom de la partie variable>:=	
1ère ligne de texte	
...	
Nème ligne de texte	

Dans la première forme, une seule des lignes est utilisée, aléatoirement, lors du remplacement. Dans la seconde forme elles sont toutes utilisées. Autrement dit, le = est un "ou" et le := est un "et". Toute ligne de texte peut contenir des parties variables notées <xxx> ou [xxx]. Celles qui sont marquées [xxx] peuvent, aléatoirement, ne pas être remplacées, les autres sont obligatoires. les mots qui ne sont marqués ni <xxx> ni [xxx] sont copiés tels que. beaucoup de règles ne contiennent pas de partie variable, elles représentent : des hommes, des femmes, des verbes de crime, des mobiles de crime, des armes de crime, des lieux, des dates, ...

<lieu>=
rue du chat qui tousse
dans un wagon de première

Exemple d'un moule pour faire un titre de France Dimanche :

<journal>=
[introduction] [date] [lieu] <homme> <verbe crime> <femme> <arme> [but]

Il y a aussi un mécanisme permettant de noter certains des choix aléatoires pour pouvoir réutiliser ces choix en plusieurs endroits. cela permet de faire apparaître un même personnage ou objet à plusieurs endroits, dans le conte par exemple. Pour cela, on dispose d'une mémoire de N places qui ont chacune un nom symbolique et un contenu, et on note :

<groupe de libellés>=x= ou <groupe de libellés>=var nom=
pour initialiser la mémoire nommée « x » (1 à 9) ou « nom » avec une brique de texte du « groupe de libellés » et <x> ou <var nom> pour réutiliser le contenu de cette mémoire.

Mais un <malfaisant>=var malfaisant= tua la poule aux œufs d'or

lors de l'édition de cette ligne de texte, un malfaisant est choisi dans le lot des malfaisants possibles défini par :

<malfaisant>=
dragon à sept têtes
beaubourius ailé

et la valeur choisie est placée dans la variable « malfaisant »
si bien que l'on peut plus loin faire générer :

le héros retrouve le <var malfaisant> et lui fait la peau

il est possible de tester l'état d'une variable :

[si var nom_variable=valeur]	la variable a une certaine valeur
[si var nom_variable#valeur]	la variable n'a pas une certaine valeur
[si var nom_variable]	la variable existe
[si non var nom_variable]	la variable n'existe pas
[]	condition aléatoire

si la condition n'est pas réalisée, la fin de la ligne est ignorée

Le caractère & fait passer à la ligne dans le texte généré.

1.2 Exemples de textes générés.

Sensationnel il y a cinq minutes, dans les sous sol des galeries Lafayette, Cadet Roussel estourbit sa secrétaire avec un couteau de cuisine pour lui prendre son larfeuille. mais que fait la police, quand on pense que Cadet Roussel s'est évadé de prison l'an passé !!

ou encore :

De notre envoyé spécial à Knock le Zout : Dracula file le parfait amour avec madame pipi du musée Grévin. que va faire Cléopâtre, elle qui se croyait dans les petits papiers de Dracula? et comment vont réagir les investisseurs étrangers ? on s'interroge dans les milieux autorisés.

volets célestes ou effroyables oubliettes
c'est bien fait
le papillon du gaz les regarde voltiger
barbichets célestes et rudes éléphants
le clop pour familles nombreuses

C'est pas croyable, Agamemnon tente de suriner la mère Denis à l'aide d'un revolver acheté aux puces, et lui prend son dentier en or.

Il vaut mieux marcher à coté de ses pompes que perdre son temps - proverbe tibétain -

La brebis à cinq pattes

Cric crac, sabot, cuiller à pot, ouvrez bien vos oreilles et tendez bien vos yeux, plus j'vous en dirai, plus j'vous mentirai. Oyez braves gens ce qu'il advint au temps ou les poules avaient des dents et ou les chats pissaient par la patte, c'était dans un grand royaume du fin fond du limousin, une brebis à cinq pattes qui appartenait au fils du roi. tout baignait dans l'huile, les hommes chantaient en se rasant et les femmes en se coiffant, les poules faisaient des œufs gros comme des pommes. Mais voila t'y pas que Nabuchodonosor était en congés payés par là. or la brebis à cinq pattes, le fils du roi avait interdit qu'on la regarde, mais nabuchodonosor était trop curieux, et il alla la regarder, comme de bien entendu. Et bien braves gens, écoutez la vilénie : un palefrenier prit la brebis à cinq pattes et disparut avec un grand rire affreux. Suite à cela, le fils du roi se fâche tout rouge, chasse nabuchodonosor à grands coup de pompes et lui ordonne de retrouver la brebis à cinq pattes et de punir le palefrenier. Et bien je vous le donne Emile, voilà que nabuchodonosor soigne une petite souris verte malade. La petite souris verte se tortille le croupion de joie et lui apprend que la brebis à cinq pattes est une princesse métamorphosée, et lui donne un gourdin invisible pour l'aider à la retrouver. nabuchodonosor va faire un petit bisou à sa chère petite maman puis s'en va. Et chemine que chemineras tu, il finit, of course, par retrouver le palefrenier. grâce au gourdin invisible, il lui fait rendre la brebis à cinq pattes et pour finir il lui coupe les oreilles. Un foutu fieffé sale con de membre du parti socialiste témoin de tout cela coupe la tête du palefrenier pendant que Nabuckodonosor astique son gourdin invisible et emballe bien soigneusement les oreilles du palefrenier dans son mouchoir avant de repartir. Le membre du parti socialiste file à l'anglaise et s'empresse d'aller raconter des balivernes au roi. Oyez braves gens la félonie du membre du parti socialiste qui obtient ainsi la main de la princesse. Pendant ce temps, nabuchodonosor a finit par digérer la bagarre et clopin clopant, il prend la route du retour. Le palefrenier prends un hélicoptère pour le rattraper mais lui, pas con, se transforme en taupe et disparaît de là. Puis il rentre par la porte de derrière, sort les oreilles du palefrenier de sa poche revolver et les montre au peuple. Le roi, qui comprends tout, fait arrêter le membre du parti socialiste et ordonne de le hacher menu. Et ca rebaigne dans l'huile et dans la soie, c'est le happy end. Vite ils ont donc fait des riz de veau, des galettes et des clafoutis, et ils se sont mis à danser tant de danses que si vous y passez vous les y trouverez encore. Et même qu'y mont donné un grand coup de pied qui m'a ramené jusqu'ici pour vous raconter l'histoire, et si vous ne me croyez pas, vous n'avez qu'a aller leur demander. Et voilà, mon p'tit conte est fini.

1.3 le conte

Pour générer les contes, on part de la grammaire suivante, mettant en relief les constituants d'un conte (merci Vladimir Propp):

1.3.1 les fonctions des personnages

le héros, celui qui va au charbon
l'antagoniste et ses acolytes, y sont chiant ceux là, créateurs du manque, il faut les éliminer !
le donateur, qui donne un auxiliaire magique au héros
le faux héros qui cherche indûment à prendre la place du héros
la princesse, elle ne pense qu'à ça !!
le père de la princesse, qui donne à réaliser les tâches difficiles, qui punit le faux héros, et qui finit par payer la noce ; les baisés, comptez-vous !

1.3.2 les mouvements du conte

l'état de bonheur initial
la perte de cet état de bonheur par malfeasance, acceptation d'une proposition perfide ou transgression d'un interdit
la quête effectuée par le héros
la lutte avec l'antagoniste
le retour du héros
l'élimination du faux héros, usurpateur de la victoire
l'exécution des tâches difficiles
l'état de bonheur retrouvé

1.3.3 grammaire du conte

Le conte est le premier décrit dans le fichier de données.

1.5 Fichier de données de journal - journal.txt

```
<histoire>=
    <conte>
    <pièce>
    <journal>
    <proverbe>
    <poème>
    <archives>

*
**          *****
***          *      les contes      *
**          *****
*

<conte>:=
    <titre>
    <formulette d'entrée>
    <corps du conte>
    <formulette finale>

*

<titre>=
    la <objet du manque>=1=&&

*

<formulette d'entrée>:=
    <mise en bouche>
    <formulette 1>
    <formulette 2> &

*

<corps du conte>:=
    <début>
    <perte de l'état de bonheur>
    <départ du héros>
    <quête effectuée par le héros>
    <victoire du héros sur l'antagoniste>
    <retour du héros>
    <rétablissement de l'état de bonheur>

*

<début>:=
    <introduction>
    , <situation spacio temporelle>,
    <objet du manque futur>
    <état de bonheur> et <état de bonheur> .
```

*

<situation spacio temporelle>=

<situation> [époque]

<époque> [situation]

*

<objet du manque futur>=

une <1> qui appartenait à un <possesseur>=2=.

un <possesseur>=2= qui avait une <1>.

*

<perte de l'état de bonheur>=

<manque par malfaisance>

<manque par transgression>

*

<manque par malfaisance>:=

<liant> <liant simple> <création du manque>

<liant causal> <arrivée du héro>

<demande au héro pour partir en quête>

*

<manque par transgression>:=

<liant causal> <arrivée du héro>

<interdiction>

<transgression de l'interdit>

<liant> <création du manque>

<envoi du héro en quête>

*

<départ du héro>:=

<séquence du donateur>

<départ>

*

<séquence du donateur>:=

<liant causal> <bienfait> <remerciement>

<don d'un auxiliaire magique>

*

<victoire du héro sur l'antagoniste>:=

<lutte du héro contre l'antagoniste>

<séquence de la marque>

*

<séquence de la marque>=

<blessure du héro>=var blessure=

<séquence faux héro>

<séquence faux héro>

<remplissage>

*

<séquence faux héro>:=

<ruse du faux héro> <pendant ruse>=var f_héro=

<retour du faux héro>

<triomphe du faux héro>

*

<retour du héro>:=

[si non var f_héro]<coming back>

[si var f_héro]<coming back f_héro>

[poursuite du héro par l'antagoniste]

[si var f_héro]<le faux héro est démasqué> <punition du faux héro>

[si var blessure]<reconnaissance>

*

<poursuite du héro par l'antagoniste>:=

<mode de la poursuite>

<mode de l'échappatoire>

*

* les éléments de texte du conte *

*

* formulettes d'entrée

*

<mise en bouche>=

Peigne à dégrasser, peigne à démêler, quand on marche on fait du ch'min.

Jarnac, quiberon, cascarinette et griffon.

Cric crac, sabot, cuiller à pot.

Soulier d'guêtre, marche avec, marche ané.

*

<formulette 1>=

dominé, d'un chien, d'un chat, d'un ch'val, d'un pot percé par les deux bouts.

si on n'tombe pas dans la saleté, on n'a pas besoin d'se nettoyer.

qu'est ce qui m'donne tant d'émotion, c'est ma gueurnouille.

ca va bien, y a d'la musique dans ma boutique.

*

<formulette 2>=

j'suis pas payé pour dire la vérité.

plus j'vous en dirai, plus j'vous mentirai.

c'est un conte de ma grand mère, mon grand père n'en savait rien.

ouvrez bien vos oreilles et tendez bien les yeux.

*

<introduction>=

Oyez braves gens, ce qu'il advint

Il était une fois

Ca se passait

C'était

*

<époque>=

il y a bien longtemps

au temps ou les poules avaient des dents et ou les chats pissaient par <

la patte

dans une de mes vies antérieures

dans le bon vieux temps

*

<situation>=

dans un grand royaume

sur une montagne

dans un petit village

dans le fin fond du limousin

*

* le possesseur de l'objet du manque

<possesseur>=

jeune fils du roi

prince charmant

fils de paysan

pauvre bonhomme

fils d'archevêque

*

* l'état de bonheur initial

<état de bonheur>=

la récolte était bonne

les troupeaux étaient nombreux et les bêtes bien grasses

les gens étaient heureux

tout baignait dans l'huile

c'était le pied

les hommes chantaient en se rasant et les femmes en se pomponnant

les poules faisaient des oeufs gros comme des pommes

* celui qui cause le manque

<malfaisant>=

commis

palfrenier

voyageur

dragon à sept têtes

beaubourius

r.e.r. ium

*

<objet du manque>=

pomme magique

poule savante

brebis à cinq pattes

table qui ne désemplit pas

lampe merveilleuse

poule aux oeufs d'or

lime à épaissir

disquette carrée

vielle à roue carrée

*

<arrivée du héro>=

<homme>=7= était en congés payés par là.

<homme>=7= habitait par là bas et par hasard.

<homme>=7= était de passage en ces lieux.

<homme>=7= se promenait dans le coin.

*

<liant causal>=

mais voila t'y pas que

or voila que

et bien vous me croirez si vous voulez mais

et bien je vous le donne émile,

*

* liant

<liant>=

et bien braves gens, écoutez la vilenie :

or braves gens, il y a bien ici bas de vils sujets, et justement

et bien vous me croirez si vous voulez mais

c'est y possible, qui est ce qui peut imaginer des affaires de même, mais

ah il fallait s'y attendre,

et la, par grand malheur,

*

<liant simple>=

il advint que

il arriva que

un jour,

ça devait arriver,

et alors,

*

<remplissage>=

vous êtes anxieux de connaître la suite hein ? et bien voilà :

quelle histoire mes amis, quel suspense.

bon, euh, ah oui

là, je prendrai bien une petite bière.

*

* le manque

<création du manque>=

un <malfaisant>=5= enleva la <1> et disparut sans laisser de traces.

un <malfaisant>=5= jeta la <1> dans la mer et s'enfuit.

un <malfaisant>=5= prit la <1> et disparut avec un grand rire affreux.

un <malfaisant>=5= s'empara de la <1> et se tailla la route avec, <
comme de bien entendu.

*

<demande au héros pour partir en quête>=

le <2> bien enmerdé demande au <7> de partir à la recherche de la <1>.

et tout le bon peuple de supplier <7> de les aider à retrouver <
la <1> du <2>.

<7>, sympa, propose au <2> en larmes de lui retrouver sa <1>.

*

<interdiction>=

pas touche à ma <1> qu'il disait le <2>.

le <2> interdit à tous de regarder sa <1>, et part en voyage de noces.

or la <1>, il était placardé partout qu'il ne fallait pas la toucher.

*

<transgression de l'interdit>=

mais <7> était trop curieux, et il regarde la <1> comme de bien entendu.

cela surpassa la curiosité de <7> qui s'empressa d'aller la voir.

ah mais ça alors quelle hardiesse, quelle témérité, voilà <7> <
qui va tourner et retourner la <1>.

*

<envoi du héros en quête>=

de retour, le <2> se fâche tout rouge et envoie <7> à la recherche <
de la <1>.

suite à cela, le <2> met <7> dehors à grands coups de pompes <
et lui ordonne de retrouver la <1> et de punir le <5>.

vous vous doutez bien que le <2> quand il l'a su, il n'a pas été <
content et que <7> il n'a plus eu qu'à assumer sa connerie.

*

* bienfait effectué par le héros

<bienfait>=

<7> donne à manger à une <donateur>=3=,

<7> était un bon bougre et trouvant une <donateur>=3= malade, il la soigne.

<7> sauve une <donateur>=3= qui était tombé dans une fosse,

*

* donateur

<donateur>=

grenouille

petite vieille

petite souris verte

mouette égarée

*

* l'auxiliaire magique

<auxiliaire>=

cheval volant

sabre doré

gourdin invisible

fond de coiffe brodé

saucisson

gâteau creusois

cable v24 croisé

*

<remerciement>=

la <3> se tortille le croupion de joie,

la <3> se métamorphose en une très belle jeune fille qui

laquelle, pour le remercier,

*

* le don

<don d'un auxiliaire magique>=

lui fait don d'un <auxiliaire>=4= et le prévient des grands dangers <
qu'il aura en recherchant la <1>.

lui dit que la <1> est une princesse métamorphosée, et lui donne <
un <auxiliaire>=4= pour l'aider à la retrouver.

lui donne un <auxiliaire>=4= et lui montre le chemin qui doit le mener <
à la <1>.

*

<départ>=

<7> va faire un petit bisou à sa chère petite maman, puis s'en va,

<7> part avec le <4>,

<7> prends la route hardiment,

*

* la quête

<quête effectuée par le héro>=

et chemine, que chemineras-tu, il finit par apercevoir le <5>.

puis, après une longue poursuite, il rattrape le <5>.

of course, il retrouve bientôt le <5>.

& par routes et chemins notre héro pugnace, <

& de son ennemi partout cherchant l'infâme trace <

& sous un palétuvier il le découvre enfin <

& tout morveux tout miteux dans cet alexandrin. <

&

*

* la lutte

<lutte du héro contre l'antagoniste>=

le tue à l'aide du <4>, lui arrache les <marque>=9= et récupère la <1>.

grâce au <4>, il lui fait rendre la <1> et lui coupe les <marque>=9=.

et ca se passe comme de bien entendu, il arrache les <marque>=9= du <5>.

*

* l'objet de la marque

<marque>=

oreilles

langues

cheveux

yeux

*

* le faux héro

<faux héro>=

petit malin

rigolo

pêcheur à la ligne

membre du parti socialiste

*

* la ruse du faux héro

<ruse du faux héro>=

un <faux héro>=8= qui a vu tout ça de loin coupe la tête du <5> ,

un <faux héro>=8= témoin de tout cela réussit à s'emparer de la tête du <5> ,

un foutu fieffé sale con de <faux héro>=8= tranche la tête du <5> ,

*

<pendant ruse>=

après que <7> se soit éloigné.

pendant que <7> astique le <4> avant de repartir.

alors que <7> emballait bien soigneusement les <9> du <5> dans son mouchoir.

*

* retour du faux héro

<retour du faux héro>=

et le <8> revient avant lui auprès du roi.

et le <8> s'en vient montrer au roi la tête du <5>.

le foutu <8> file à l'anglaise et s'empresse d'aller raconter des <
balivernes au roi.

*

* le triomphe du faux héro

<triomphe du faux héro>=

le roi se met donc à préparer les noces de sa fille avec le <8>.

alors un grand banquet est organisé pour les fiançailles du <8> avec <
la princesse.

enfer et damnation, les fiançailles de la princesse avec le fieffé <8> <
sont annoncées.

oyez braves gens la félonie du <8>, qui obtient ainsi la main de la princesse.

*

* le faux héro est démasqué

<le faux héro est démasqué>=

<7> rentre par la porte de derrière et montre alors les <9> du <5> au peuple ,
c'est là que, <7> sort de sa poche revolver les <9> du <5> ,
mais c'est bien sur, <7> arrive en brandissant les <9> du <5> ,

*

<punition du faux héro>=

le sale foutu <8> est alors chassé à coups de pierre.
le <2> furieux coupe alors la tête du <8> et hache menu son corps.
le <8> passe alors un très sale quart d'heure.

*

<blessure du héro>=

mais dans la bagarre, <7> en prends plein la gueule.
le <5> a quand même le temps de bien amocher notre cher <7>.

*

<coming back>=

et, clopin, clopant, il se tire de là.
puis reprend la route du retour.
et, commençant à en avoir marre de tout ça, il rentre.

*

<coming back f_héro>=

et pendant ce temps, <7>, ayant digéré la bagarre, revient avec la <1>.
de son coté <7> finit par revenir.
bon, j'abrège, et le <7> se met sur la route du retour.

*

<mode de la poursuite>=

avec un lasso le <5> cherche à rattraper <7> ,
le <5> prends un hélicoptère pour rattraper <7> ,
le <5> envoie une escouade de ses acolytes pour rattraper <7> ,

*

<mode de l'échappatoire>=

qui s'en sort fort habilement grace au <4> .
lequel, pas con, se transforme en taupe et disparaît de là.
mais lui, ben, il euh il ... ,je sais plus quoi inventer, mais ... il se tire.

*

<reconnaissance>=

de retour, <7> montre sa blessure, et chacun de célébrer sa bravoure.
la blessure de <7> montra à tous sa victoire.

*

* triomphe du héro

<rétablissement de l'état de bonheur>=

et tout finit par s'arranger.

c'est le happy end.

et ca rebaigne dans l'huile et dans la soie.

vite, ils ont donc fait des riz de veau, des galettes et des clafoutis.

et ils se sont mis à danser tant de danses que si vous y passez, vous les <
y trouverez encore.

*

* formulette finale

<formulette finale>=

<formule finale 1> <formule finale 2>

*

<formule finale 1>=

et y m'ont donné un grand coup d'pied au cul qui m'a envoyé jusqu'ici

et j'ai trouvé une nichée d'hirondelles qui m'a ramené ici pour vous le <
raconter

et moi je suis revenu à la maison pour vous raconter tout cela

*

<formule finale 2>=

et voila, mon p'tit conte est fini

et si vous m'croiez pas, vous n'avez qu'à allez leur demander

```

*
*
**      *****
***      *      les pièces de théâtre      *
**      *****
*
<pièce>:=
    <personnages> æ
    <premier acte>
    <deuxième acte>
    <troisième acte> æ
    <quatrième acte>
    <cinquième acte>
*
<personnages>:=
    <titre pièce>&
    [] ou <sous titre>&
&
PERSONNAGES :&
    <homme>=1=, époux de <femme>=2=&
    <2>, épouse de <1>&
    <homme>=4=, amant de <2>&
    <homme>=3=, fils de <1> et <2>&
    [] <homme>=5=, confident de <3>&
    <femme>=6=, fille de <1> et <2>&
    <homme>=7=, le valet de chambre&
    [] <femme>=8=, la bonne&
    [] <homme>=9=, <emploi masculin>&
    [] <femme>=0=, <emploi féminin>&&
    la scène se passe <lieu>&
    -----&
<premier acte>:=&
    PREMIER ACTE&
    [] [ambiance 1] [ambiance 2]&
    <acte 1> <conj cause> <cause 1>&
    -----&
<deuxième acte>:=
    DEUXIEME ACTE&
    <4> <acte 2>&
    -----&
<troisième acte>:=

```

TROISIEME ACTE&

[] la scène se passe <lieu>&
[] [ambiance 1] [ambiance 2]&&
<3> <verbe acte 3> <4>&
<acte 3>&

-----&

<quatrième acte>:=

QUATRIEME ACTE&

<6> <verbe acte 4> <7>&
[si 9]<9> <acte 4>&
[si non 9]<4> <acte 4>&

-----&

<cinquième acte>:=

CINQUIEME ACTE&

[si 0]<0> <verbe acte 5> <7>&
[si non 0]<6> <verbe acte 5> <7>&
<acte 5>&

-----&

RIDEAU&

*

* le titre de la pièce

<titre pièce>=

<homme ou femme> s'en va t'en guerre
<homme ou femme> face à son destin
<homme ou femme> malgré lui
en attendant <homme ou femme>
le retour de <homme ou femme>
le fils de <homme ou femme>
as tu vu <homme ou femme>
les malheurs de <homme ou femme>
la chanson de <homme ou femme>
le fils de <homme ou femme>
la clémence de <homme ou femme>

*

* le sous titre

<sous titre>=

la vertu bafouée
le pitricol
le vice à l'état pur
le hasard retrouvé

deconnatos

les aventures de syntaxikos

*

* emplois masculins

<emploi masculin>=

flic

plombier

voisin

accordéoniste

pharmacien

notable

conseil en carambouille

général

précepteur escroc

écrivain

homme invisible

petit télégraphiste

palotin

mousquetaire du roy

pilote de ligne

bedeau

musicien à la veillée limousine

explorateur

apache

présentateur du journal télévisé

membre de plusieurs sociétés savantes

banquier juif

bourgeois

chapelier fou

*

* emplois féminins

<emploi féminin>=

concierge

filles de joie

petite main chez chanel

libraire

témoin de jehovah

jeune femme d'affaire arriviste

taxi girl

théâtreuse

travesti
rosière
femme de roulier
pythonisse
bergère
historienne de la fronde
délégue syndicale
emmerderesse
demoiselle du téléphone
dame pipi
échoitière
nourrice
personnage en quête d'auteur
meunière
*

* premier acte

<acte 1>=

au lever du rideau <1> se querelle avec <2>

<1>,seul, décide de supprimer <4>

<4> déclare sa flamme à <2>, ils sont inquiets

<1> annonce son intention d'acheter l'usine de bouteilles en plastique du père de <4>

<1> a des soupçons sur la fidélité de <2> mais décide de les taire

<2> annonce à <1> son intention de faire la grève des devoirs conjugaux

au lever du rideau, <6> est étendue inanimée. elle demande à <7> où elle <

se trouve, et s'inquiète

au cour d'un cocktail, <2> retrouve <1> dont elle s'était séparée depuis un an

<7> revient de mamouth avec <6> ils sont écoeurés

<2> regarde la télévision. Elle reconnaît avec stupeur <7>, son ancien mari, <

qui participe à un débat

*

* conjonction de cause

<conj cause>=

à cause

en raison

conséquence

*

* cause du premier acte

<cause 1>=

de la qualité de la nourriture

des dépenses domestiques jugées excessives

de l'affection envahissante de son caniche nain

de la prolifération étrange des mouches dans la cité

de la médiocrité des émissions télévisées

de la dévaluation galopante

de la vanité des choses de ce monde

des tigres de papier au service de l'impérialisme bourgeois

de la grève des transports urbains

*

* deuxième acte

<acte 2>=

affirme à <3> que <7> est une personne plus mystérieuse qu'il n'y paraît et <

lui explique pourquoi

dit à <3> de se méfier de <7> qui a de sombres visées sur le magot de la <

famille

cherche à avoir une conversation sérieuse avec <3> en l'absence de <7>.

fait part à <3> de sa passion coupable éprouvée envers <6> . réaction violente reproche à <6> son attitude distante vis à vis de <7>.

informe <2> que <6> n'est pas ce que l'on croit, mais son propre enfant évoque auprès de <6> sa jeunesse aux indes et révèle y avoir connu <7>, <

alors dans l'espionnage

et <6> cherchent comment échapper au chantage qu'exerce sur eux l'immonde <7>

cherche à convertir <6> à sa secte, après y avoir déjà entraîné <3>

*

* troisième acte

<verbe acte 3>=

rencontre par hasard

raconte ses chagrins à

refuse d'obéir aux volontés de

cherche vainement à calmer

fait une scène d'hystérie devant

va jusqu'à provoquer en duel

se fait habilement tirer les vers du nez par

assiste dans l'impuissance à la décadence de

*

* suite du troisième acte

<acte 3>=

les deux méditent une vengeance

les deux se séparent fort en colère

on se réconcilie

personne ne sait comment s'en sortir

c'est l'effet d'une malédiction jetée sur la famille

les deux s'engagent dans un passage secret

flash back

l'obstination d'un père fera-t-elle le malheur de ses enfants

la tromperie est démasquée

le crime restera t'il impuni

*

* quatrième acte

<verbe acte 4>=

met au monde un enfant de

finit par reconnaître son propre enfant en

est faussement accusée par

tombe malade. Elle reçoit la visite de

gagne au loto en jouant la date de naissance de
est compromise par
poursuivie par la malchance, fait des ménages chez le frère de
entre et dit : 'madame est servie' et lance des oeillades à
vampirise littéralement
tente de suborner le commissaire. Echech retentissant
part comme infirmière militaire. Adieux émouvants
*

* suite du quatrième acte

<acte 4>=

tente de s'interposer
que l'on avait cru mort reparait dans le monde. Grosse sensation
simule une maladie pour tromper
connaissait la vérité
joue un double jeu
fait une confession surprenante
refuse ce qu'on lui propose
fait rire tout le monde par ses spirituelles réparties
sort en claquant la porte
*

* cinquième acte

<verbe acte 5>=

conçoit une terrible jalousie envers
se fait surprendre avec
tourne autour du pot mais n'ose se déclarer à
embrasse
épouse
cherche maladroitement à séduire
déclare sa flamme à
tue
crible de sarcasmes
*

* fin du cinquième acte

<acte 5>=

et ils vivent heureux
mais le destin frappe à la porte
intervention d'un deux ex machina
et en avertit les journaux
et se livre à la police

et défaille de bonheur
et meurt
et s'en va
mais en réchappera miraculeusement

*

* ambiance

<ambiance 1>=

deux fauteuils louis XV
un vase, un téléphone
des fleurs
fenêtre donnant sur un jardin
jardin donnant sur une fenêtre
bibliothèque fournie
une table, des chaises, un repas inachevé
un feu dans une cheminée

*

* ambiance

<ambiance 2>=

musique douce
la radio à toute force
on entend des pas
bruits divers
la pendule sonne
rires étouffés
une porte claque
soleil éclatant
il pleut
odeur nauséabonde
panne d'électricité, une chandelle
sensation pénible

*

```

*
**          *****
***          *      les titres de journaux      *
**          *****
*
<journal>=
    <lieu et date> <fait divers> &&
*
<lieu et date>=
    [intro] [date] [lieu]
    [intro] [lieu] [date]
*
<fait divers>=
    <crime> &[analyse crime]
    <histoire rose> &[analyse rose]
*
<histoire rose>=
    <homme>=1= <verbe amour> <femme>=2= [verbe amitié]
    <femme>=1= <verbe amour> <homme>=2= <verbe amitié>
*
<crime>=
    <homme>=1= <verbe crime> <femme>=2= <conj crime> <
        <arme crime> <et verbe crime>
    <homme>=1= <verbe crime> <femme>=2= <conj crime> <
        <arme crime> [verbe 1 fin crime] [verbe 2 fin crime]
*
<analyse crime>=
    on s'interroge dans les milieux autorisés
    mais que fait la police, quand on sait que <1> s'est évadé <
        de prison l'an passé !
    comment va réagir <homme> ?, que vont faire les investisseurs étrangers?
    la police a mis son meilleur limier sur le coup : <homme> <
        et donc <1> n'en a plus pour longtemps à sévir
    <2> n'était pas très intéressante, mais quand même
    les amis de <2> comptent devancer la police et se venger eux mêmes
*
<analyse rose>=
    que va faire <femme> ? , elle qui se croyait dans les petits papiers <
        de <1>
    mais jusqu'ou ira <1> ?
    rien n'arrêtera donc <1>
*

```

*

* pour faire les introductions

<intro>=

c'est pas possible !!!

extraordinaire :

on aura tout vu :

mais que fait le gouvernement,

encore un méfait de l'alcool :

jusqu'ou iront ils,

écoutez moi ça:

c'est pas croyable,

de notre envoyé spécial :

ah ma bonne dame, c'est plus ce que c'était

si je l'avais pas vu, je l'croirais pas

et allez donc,

*

* pour faire des lieux

*

<lieu>=

à la courbe

à Saint Lumine

dans un immeuble cossu de la rue de rennes

en pleine rue

dans les Vosges

dans un coffre de R14

dans les sous-sol des galeries Lafayette

au métro Raspail

dans un train

au palace

à l'éllysée

au parc des princes

à l'intérieur d'une noix

à la conciergerie de l'académie française

sur un porte avion

sur un banc public

square des Batignolles

à la gare Montparnasse

sur une plage mazoutée

dans une H L M de la Motte Beuvron

à la rédaction du petit écho de la mode

*

* pour faire des dates

<date>=

hier matin

il y a deux minutes

avant hier

à l'instant

mardi dernier

à 3 heure ce matin,

peu après le coucher du soleil

à la tombée de la nuit hier soir,

*

** des hommes

*

<homme>=

son coiffeur

l'avocat marron

son chauffeur

le notaire véreux

le pdg en faillite

un jeune voyou

Marchais

Johnny Halliday

Alain Delon

Eric la terreur

l'infâme Olivier

le gentil Pierre

Guillaume

Philippe

le gros Philippe

un jeune militaire en permission

Mourouzi

Bernard Pivot

Spaggiari

Toutankhamon

Rouletabille

Superdupont

King Kong

père Dupanloup

Frankenstein

Agamemnon

Nabuchodonosor

*

** des femmes

*

<femme>=

Bénédicte

Domi

Marie Antoinette

sa concubine

sa secrétaire

Mireille Mathieu

Simone Weil

Alice Apritch

une danseuse des folies bergères

madame pipi du musée Grévin

Sheila

Cléopâtre

Maia l'abeille

la mère Denis

Berthe aux grands pieds

Barbarella

*

** des hommes et des femmes

*

<homme ou femme>=

<homme>

<femme>

*

* verbes d'amour

*

<verbe amour>=

fait tout ce que veux

ne jure plus que de

est mené par le bout du nez par

se fait mener à la baguette par

ne sent plus le temps passer avec

file le parfait amour avec

ne peut plus se passer de

roucoule avec

joue à la bête à deux dos avec

*

* armes de crimes

*

<arme crime>=

une hache

ses mains nues

un couteau de cuisine

un poison fulgurant

un revolver acheté aux puces

un fusil de chasse

une planche à clous

un piège à loup

un gourdin

*

* verbes d'amitié

*

<verbe amitié>=

et va lui faire tous ces caprices

et lui offre des chocolats

et compte partir avec en vacances aux baléares

et s'est fait couper les cheveux pour lui plaire

et ne sait plus quoi inventer pour lui plaire

et va changer d'employeur pour se rapprocher

*

* verbes de crime

*

<verbe crime>=

assassine

torture

tente de suriner

bute

tue

exécute

estourbit

*

* conjonction

*

<conj crime>=

avec

à l'aide d'

au moyen d'

*

* verbes de fin de crime

*

<verbe 1 fin crime>=

et découpe son corps en rondelles

et brule son corps dans sa chaudière

et l'arrose d'essence

et hache menu son corps

et lacère son corps

*

* autres verbes de fin de crime

*

<verbe 2 fin crime>=

et se livre à la police

et tente de maquiller ça en suicide

et met le feu pour dissimuler son forfait

et met ce qu'il en résulte dans une consigne de la gare Saint Lazare

et balance le tout dans son vide ordure et le bouche

et en fait un paquet et le jette dans le canal

et le cache dans le congélateur

et le coule dans le béton sous sa terrasse

*

* autres verbes de crime

<et verbe crime>=

et lui prends son porte monnaie

et lui vole tout son pèze

et lui coupe le doigt pour lui arracher son alliance

et lui prend son dentier en or

et lui vole la carte bleue

*

```

**                *****
***              *      les poèmes      *
**                *****

<poème>:=
                <nom 1> <adj 1> <conj> <adj 2> <nom 2> &
                <verbe 2> &
                <groupe nom 1> <verbe 1> &
                <nom 1> <adj 1> <conj> <adj 2> <nom 2> &
                <groupe nom 2> <relative> &&&

*

<nom 1>=
volets
refrains
réverbères
aigles
barbichets
reflets
riz de veaux
*

<adj 1>=
azurés
célestes
divins
aphrodisiaques
légendaires
*

<adj 2>=
sombres
rudes
tristes
effroyables
ténébreux
funestes
*

<nom 2>=
oubliettes
montagnes
éléphants
rosiers
regrets
trompettes
clafoutis

```

*

<groupe nom 1>=

le papillon du gaz

la voix de cristal

le feux follet

*

<verbe 1>=

les regarde voltiger

s'est brisé

recommence à siffler

s'installe pour souper

*

<conj>=

et

ou

comme

*

<verbe 2>=

ils erraient

qui est là

c'est bien fait

personne

nul autre

tous

*

<groupe nom 2>=

l'oeuf d'autruche de paques

la boussole

le rond de serviette

le clop

la crêpe

le saucisson

*

<relative>=

pour familles nombreuses

des filles perdues

du bon serviteur

au fibro chlorure de zitracil bi-anhydre

du livre des pharaons

*

```

**                *****
***              *      les proverbes      *
**                *****
*
<proverbe>=
    il vaut mieux <action 1> que <action 2> [signature]
    pourquoi <action 2> quand on peut <action 1> [signature]
    si on veut te faire <action 2> alors fait les <action 1>
*
<action 1>=
prendre son pied
aller se promener
boire un coup
dormir
faire faire aujourd'hui par un autre ce qu'on peut faire soi même demain
être riche et bien portant
*
<action 2>=
perdre son temps
être pauvre et malade
marcher à coté de ses pompes
se défoncer
apprendre le thibétain
descendre d'un ancêtre roturier
*
* signature
*****
<signature>=
&   - proverbe afouicain -
&   - proverbe chinois -
&   - la Rochefoucauld (1616-1736) -
&   - proverbe arabe -
&   - Napoléon -
&   - Charles de Gaulle -
&   - Pierre de Boishéraud (1948- ) -
*

```

*

*

Les archives de la Courbejollière

*

<archives>=

le <jour> <mois> <année> <naissance>

<mariage>

<décès>

en <mois> <année> <homme archive> <fait militaire>

<homme archive> <fait militaire> en <mois> <année>

<homme archive> <action> en <mois> <année>

en <mois> <année> <homme archive> <action>

<femme archive> <action> en <mois> <année>

en <mois> <année> <femme archive> <action de femme>

*

<jour>=

1

2

3

4

5

6

13

14

24

25

*

<mois>=

janvier

février

mars

avril

mai

juin

juillet

*

<année>=

1611

1716

1725

1514

1810

1793

1803

1875

1842

*

<naissance>=

est né <homme archive> <lieu de naissance> <

les témoins déclarant sont <fermier> et <artisan>

<témoin> est venu nous déclarer la naissance <lieu de naissance> de <femme archive>

*

<mariage>=

ce <jour> <mois> <année> <homme archive> a épousé demoiselle <femme archive> <

les témoins, qui ont déclaré ne savoir signer, sont <fermier> et <artisan>

*

<décès>=

<fermier> nous a déclaré ce jour <jour> <mois> <année> la mort de <homme archive> <
<lieu de décès>

*

<fait militaire>=

<fait1>

<fait1> et <fait2>

*

<fait1>=

a été tué au siège de la flocellière

a repoussé, devant henri IV, une troupe de ligueurs à monnières

a été blessé par un hussard à la bataille du mans

a eu un cheval tué sous lui à la bataille d'angers

a suivi la rochejaquelin à nantes

a forcé l'ennemi à abandonner ses canons

a suivi l'armée catholique dans toute la virée de galerne

a émigré en Allemagne puis en Russie et fait toutes les campagnes du corps

a été noyé dans la loire par l'envoyé de la convention, carrier

a acheté des biens nationaux d'église

*

<fait2>=

a été nommé lieutenant d'une compagnie de chevaux legers

a reçu son brevet de capitaine

a reçu la croix de saint louis

a été décoré de la légion d'honneur

a été reçu par le prince de condé

a été cité à l'ordre de son régiment

*

<action>=

a fait curer les douves
a loué la ferme des bois à <fermier>
a loué la ferme de la Courbejollière à <fermier>
a fait faire la table et les chaises de la salle à manger par <artisan>
a acheté un baril de chlorate
a refait les toitures de la Courbejollière
a refait le pont-levis en bois
a donné un champ pour construire l'école
s'est fait exproprier le potager par la mairie
a tiré les rats musqués avec un fusil allemand
a installé les colonnes de la salle à manger
a acheté la guérvivière
a hérité la Courbejollière de son oncle Louis
a hérité la Courbejollière de son oncle Sébastien
a brisé une roue de son carosse sur la route de Nantes
a installé le chauffage central
a aménagé sa chambre dans la tour

*

<action de femme>=

a brodé la tapisserie des fauteuils du salon
a planté les orangers
a fait la plate bande des rosiers du petit jardin
protégée par le docteur Darbefeuille, s'est échappée de la prison<
de l'entrepôt de Nantes, où elle aurait été noyée comme sa mère
a brodé les chaussons pour le passage de la duchesse de Berry
a brodé la nappe et les serviettes pour la grande table

*

<fermier>=

Jean Marie Vilaine
Michel Blanloeil
Jean Marc Leclerc
Marc Cormerai
Hervouet

*

<artisan>=

Pierre Berthou
Jean Leclerc
Michel Vilaine
Jean Paul Brosseau

*

<homme archive>=

le sieur Pierre Perrin de la Courbejollière

Alexandre Perrin de la Courbejollière
le général Bernard de Boishéraud
Jean de Boishéraud
l'oncle Sébastien
Pierre Mosnay de Boishéraud

*

<femme archive>=

Adélaïde Perrin de la Courbejollière
Marie-Antoinette Deslandes de Bagneux
Laurence de Boishéraud
Esther Mesnard de Toucheprest
Marie Anne Leray de la Clartais

*

<témoin>=

<fermier>

<artisan>

*

<lieu de naissance>=

à la Courbejollière
à la Guérivière
à Nantes
au château de clisson

*

<lieu de décès>=

à la Courbejollière
à la Guérivière
au siège de la Flocellière
à la bataille du Mans
à la bataille de Joué

*

Un programme qui raisonne

Où Syntaxikos rencontre d'abord Syllogismus Hypothéticus, puis Semantika et comment ils décident de faire route ensemble.

1.1 Présentation

On se propose d'emmagasiner les informations comprises dans des phrases en français puis de répondre à des questions sur ces affirmations, tout en effectuant des déductions. Pour cela, on se base uniquement sur la syntaxe des phrases, et grâce à des transformations simples on s'efforce d'obtenir la même représentation pour des phrases équivalentes.

1.2 Syntaxe des affirmations

Les affirmations peuvent prendre l'une des trois formes suivantes :

groupe_nominal verbe groupe_nominal
groupe_nominal verbe
celui qui verbe groupe_nominal verbe groupe_nominal

Exemples :

le chat noir mange la petite souris
celui qui mange une pomme verte tombe malade
toute souris aime le gruyère
aucun contribuable n'aime son percepteur

Le verbe peut être mis à la forme négative

ne verbe pas
n'verbe pas

Il peut aussi être mis à la forme passive

est verbé par
n'est pas verbé par

Par groupe nominal (Gn) on entend :

logique booléen article nom attribut
logique, c'est : tout ou aucun
booléen, c'est : non
article, c'est : le, la, les, un, une, des, mon, ton, son, ce, cet ...
attribut, c'est un adjectif

le groupe nominal doit contenir au moins un nom ou un adjectif
on peut le remplacer par : qui, quoi, ou, comment, quand, combien
le système répond alors à la question correspondante.

On peut de même remplacer le verbe par : fait

(ces « Qui », « quoi », « fait » réagissent comme les variables de prolog)

1.3 Syntaxe des questions :

les questions peuvent prendre l'une des formes suivantes :

gn verbe gn?
celui qui verbe gn verbe gn? (deux seuls cas où le ? est obligatoire)
qui verbe gn
gn verbe qui
qui verbe qui (qui, quoi, quand, comment, quel)
gn fait quoi
est ce que gn verbe (gn)
gn verbe-t-pronom gn
(ou quand comment combien) gn verbe
(ou quand comment combien) verbe gn
qui est ce qui verbe (gn)
qu est ce qui verbe (gn)
qu est ce que gn
qui est gn
qu est ce que verbe gn
qu est ce que gn verbe
que verbe gn
par qui est verbé gn
par qui gn est-il verbé
qui est verbé par gn
qui est verbé
qui fait quoi

donne la liste complète de toutes les affirmations

1.4 Relations entre les mots

On peut définir des relations entre les mots :

mot est l'inverse de mot (monter est l'inverse de descendre)
mot est synonyme de mot (manger est synonyme de bouffer)
mot est réflexif (frère, ami sont réflexifs)
mot est le symétrique de mot (père est le symétrique de fils)
donner est symétrique de recevoir

1.5 Moteur d'inférence

la logique suivie est celle bien connue des syllogismes :

si $a \implies b$ alors non $b \implies$ non a

si $a \implies b$ et $b \implies c$ alors $a \implies c$

si le programme ne connaît pas un mot, et ne peut en déduire son genre grammatical (par la présence d'un article) il en demande le genre grammatical, et le dictionnaire s'élargit.

(n : nom , a : adjectif , d : adverbe , v : verbe)

si on répond alors « return », la phrase est poubellifiée, poil au nez.

1.6 Exemples de dialogue

les phrases précédées de "?" sont celles frappées par le joueur.

? Marc est fou

? intelligent est l'inverse de fou

? tout fou est dangereux

? aucun programmeur n'est fou

? monte est l'inverse de descend

? Marc monte l'escalier

? *qui est fou*

Marc

? *est ce que Marc est dangereux*

oui

? *est ce que Marc est programmeur*

non

? *est ce que Marc descend*

non

? *Marc mange des pommes crues*

? *que mange Marc*

des pommes crues

? *Marc mange quoi*

des pommes crues

? *pierre mange des pommes excellentes*

? *qui mange des pommes*

pierre

Marc

? *appétissantes est synonyme de excellentes*

? *qui mange des pommes appétissantes*

pierre

? *qui mange quoi*

Marc mange des pommes crues

pierre mange des pommes excellentes

? *qu'est Marc*

(ou *qu'est ce que Marc ?*)

Marc est un fou

tout fou est dangereux

un fou n'est pas programmeur

Marc monte l'escalier

Marc mange des pommes crues

et maintenant, très fort :

? père est le symétrique de fils
? frère est réflexif
? Marc est le frère de Pierre
? Marc est fils de Bernard
? est ce que Pierre est le frère de Marc
oui
? quel est le père de Marc
Bernard

? donne est le symétrique de reçoit
? pierre donne du chocolat à Marc
? qui reçoit du chocolat
Marc

? le rat est mangé par le chat
? qui mange le rat
le chat
? par qui est mangé le rat
le chat
? le rat est mangé par qui
le chat
? qui est mangé
le rat

? un chien est un canidé
? un canidé est un mammifère
? un mammifère est un animal
? un chien est-il un animal
oui

? un chien est quoi
un canidé
un mammifère
un animal

? un canidé a des griffes
? un cheval n'a pas de griffes
? un cheval est il un canidé
non

encore plus fort (les axiomes de départ n'engagent que moi !!!)

? un schtroumpf vote mal
? celui qui vote mal est fou
? je suis malin
? malin est l'inverse de fou
? est ce que je suis schtroumpf
non

ouaouhhhh !!!!!

1.7 Commandes de gestion

il existe enfin quelques commandes spéciales :

*lire toto.txt	pour lire un fichier de déclarations
*help	pour avoir des indications
*raz	efface tout
*fin	fin du jeu

1.8 Fonctionnement du programme

1.8.1 Dictionnaire et valeur sémantique

Le dictionnaire est représenté en mémoire par un ensemble de tables parallèles contenant : le libellé du mot, sa valeur sémantique, la valeur éventuelle de son symétrique, son type grammatical et son sous-type. Pour une recherche rapide, les mots sont chaînés à l'aide d'un "hshcode" égal à leur première lettre. La valeur sémantique des mots est liée à l'utilisation des déclarations de synonymie et d'antonymie : si deux mots sont déclarés synonymes, ils ont même valeur et si ils sont déclarés antonymes, ils ont des valeurs dont la somme est nulle.

d'où les équivalences entre

x ne verbe pas y , aucun x ne verbe y , x ne verbe aucun y
et x ant_verbe y
avec verbe + ant_verbe = 0. (verbes antonymes)

de même si

un non x verbe y
cela est équivalent à
un ant_x verbe y
avec
 $x + \text{ant}_x = 0$ (noms ou adjectifs antonymes)

et (uniquement avec le verbe être) si

x n'est pas y
cela équivaut à
 x est non y
ou à
 x est ant_y
avec
 $y + \text{ant}_y = 0$ (nom ou adjectifs antonymes)

1.8.2 Stockage des informations

Une affirmation doit être stockée en mémoire, elle a la forme :

celui qui verbe' gn' verbe gn

la forme gn' verbe gn en étant un cas particulier

("celui qui est gn' verbe gn " ou l'on aurait retiré "celui qui est")

soit donc 2 verbes, et 2 Gn, chaque Gn étant composé d'un nom et d'un adjectif avec pour chaque mot éventuellement une marque de négation et un article pour les noms ou une préposition pour les verbes.

Si la phrase est à la forme passive, elle est stockée à la forme active

le rat est mangé par le chat

devient donc, après une transformation d'inversion :

le chat mange le rat

ou plus exactement, comme on vient de le voir :

celui qui est le chat mange le rat

Chaque phrase est alors stockée sous sa forme directe et sous sa forme inversée laquelle est équivalente à :

celui qui ne verbe pas gn ne verbe' pas gn'

soit donc pour la forme réduite :

celui qui ne verbe pas gn n'est pas gn'

dans notre exemple, cela donne donc :

celui qui ne mange pas le rat n'est pas le chat

ce stockage inversé pourrait être évité, mais il simplifie le moteur d'inférence.

1.8.3 Fonctionnement du moteur d'inférence

Après analyse de la question, celle ci est mise sous une forme semblable à celle d'une affirmation. la seule différence étant qu'un ou plusieurs éléments sont des pronoms (qui, que, quoi, où, quand, comment ...) ou le verbe faire. ces éléments prennent une valeur sémantique particulière (-1).

Ainsi on conçoit bien que les phrase suivantes sont comparables :

Marc mange une pomme

qui mange une pomme

qui mange quoi

qui fait quoi

Marc mange quoi

De même, après une petite transformation consistant en une inversion :

que mange Marc

Il peut aussi arriver que la question soit complète :

est ce que gn verbe gn'

ou gn verbe gn' ?

Pour tenir compte efficacement des règles "celui qui ...", et des affirmations du genre : "A est B", on commence par établir une liste des équivalents du premier membre de la question.

par exemple si l'on a déjà stocké :

celui qui Verbe Gn Verbe' Gn'

et Gn est Gn"

alors la question :

Gn V Ggn ?

calcule d'abord la liste d'équivalence :

Gn, Gn"

et la question :

celui qui Verbe Gn Verbe" Gn"

calcule la liste :

celui qui Verbe Gn, celui qui Verbe' Gn'

Cet algorithme est itératif, une équivalence trouvée pouvant être elle même source d'une autre équivalence. Il suffit ensuite de parcourir toutes les phrases stockées et de les comparer à la question en tenant compte de ces équivalences. Une phrase stockée répond à la question (ou confirme ou contredit l'affirmation) si les deux verbes, les deux noms et les deux adjectifs ont une valeur sémantique égale ou opposée, en tenant compte du fait qu'une valeur sémantique égale à -1 correspond avec n'importe quoi et que la valeur sémantique 0 (absence d'adjectif ou de nom) dans la question correspond aussi avec n'importe quoi.

La comparaison se fait en plusieurs temps :

- comparaison des deux phrases telles quelles

- si le 2ème nom a un réflexif déclaré, transformation puis comparaison

x est frère de y

entraîne donc (si "frère est réflexif" a été affirmé) la comparaison avec

y est frère de x

- si le 2ème nom a un symétrique déclaré, transformation puis comparaison

x est le patron de y

devient (si "patron est symétrique de employé" a été affirmé)

y est l'employé de x

- si le verbe a un symétrique déclaré, transformation puis comparaison

x donne y à z

devient (si "donne est symétrique de reçoit" a été affirmé)

z reçoit y de x

Une même phrase peut subir successivement plusieurs de ces transformations. lorsqu'une réponse correspond, il n'y a plus qu'à afficher le résultat. On n'affiche que les mots correspondants à une valeur sémantique égale à -1 (ou tous si le verbe est seul différent de -1). On passe alors à la phrase stockée suivante. Si une phrase stockée contredit la question, on répond "non" et on arrête. si aucune phrase ne répond à la question, on répond "je ne sais pas". Avant de stocker une phrase, il est nécessaire de vérifier sa cohérence avec les affirmations précédentes. On la considère donc d'abord comme une question et on ne la stocke que si on est amené à répondre : "je ne sais pas".

2 Fichier de données de moslog - moslog.txt

Le dictionnaire de départ est très réduit. Il est réparti en quelques groupes et sous-groupes ne définissant que des mots outils :

* groupe , sous-groupe

par exemple "est" est un élément du sous groupe du verbe être (e) qui est lui même dans le groupe des verbes (v).

on note donc :

*v,e est

les mots séparés par des virgules sont traités comme des synonymes.

*r,r le la les un une des l ma mon mes sa son ses ta ton tes cet cette ces

*k,k du de en d à

*w,w s se n ne c ca ce

*v,e est,suis,es,sont

*v,a a,as,ont

*v,? fait,font

*n,# inverse,opposé,contraire,antonyme

*n,= synonyme,équivalent

*n,f reflexif

*n,d transitif

*n,c symétrique

*v,e implique,entraîne,signifie,entraîne,appelle

*n,n je,j tu

*n,& il ils elle elles

*l,l tout,toute,tous,toutes

*z,z aucun,aucune

*b,b non

*p,p pas

*x,x quel quels quelle quelles

*o,o ou quand comment combien que

*q,q qu

*i,i qui quoi

*a,1 verbe

*a,2 adjectif

*a,3 nom

*,(celui celle ceux

*:, par

1.11 Limitations

Le programme n'utilise qu'un dictionnaire très réduit ne contenant que les verbes être et avoir, les articles et les quelques mots clés du langage. Il n'a aucune connaissances sémantiques. Il ne conjugue pas les verbes et ne connaît pas les règles d'accord. Il vaut donc mieux tout mettre au masculin, au singulier et à la 3ème personne pour que cela fonctionne au mieux. Tant pis si certaines phrases choquent les grammairiens. La négation n'est vraiment efficace que sur des groupes nominaux sans adjectif, de même pour les raisonnements. Les opérateurs logiques "et" et "ou" ne sont pas gérés, la construction "si P alors Q" non plus, pas plus que les propositions relatives du genre "le gn qui v gn v gn" ou "le gn v le gn qui v gn", mais elles ne perdent rien pour attendre, Prologistos est sur le coup.

Scène 4

Un jeu de devinettes

Où Syntaxikos, piqué par la mouche toute-folle, retombe en enfance et joue aux devinettes avec son vieil ami Basic.

1.1 Principe de la déduction

Le programme fonctionne en autoapprentissage et devine ce à quoi on pense en posant des questions auxquelles on ne répond que par oui ou par non. L'analyse de la phrase n'est donc pas trop compliquée ! Il est écrit en basic, c'est un petit programme qui utilise un fichier de données contenant des connaissances de départ. (ce fichier peut être vide). Les connaissances augmentent lorsque, n'ayant pas trouvé, le programme donne sa langue au chat, il faut alors lui donner la solution et la distinguer de celle qu'il a proposée. On peut utiliser ce programme sérieusement, c'est à dire non pas pour jouer aux devinettes mais pour rechercher des pannes par exemple. Les questions posées sont alors du genre :

le voyant rouge est-il allumé
est ce que la prise est bien enfoncée

Mais alors, si il faut écrire des programmes qui servent à quelquechose ou va t-on? c'est pas croyable, mais que fait le gouvernement ? je vous le demande madame bichou.

1.2 Exemple de dialogue

tu vas penser à quelqu'un ou à quelquechose, et je vais le deviner

ca y est, tu as choisi ?

.oui

c'est une personne

.non

c'est un animal

.non

c'est un végétal

.non

c'est un objet

.oui

ca se boit

.oui

c'était : un verre de rouge

c'était bien ca ?

.non

je donne ma langue au chat, c'était quoi ?

.un pastis

pose moi une question qui permette de distinguer un verre de rouge
d'un pastis

.on y met de l'eau

qu'aurai tu répondu si je t'avais posé cette question :

un pastis, on y met de l'eau ?

.oui

c'est un végétal

.non

c'est un objet

.oui

ca se boit

.oui

on y met de l'eau

.oui

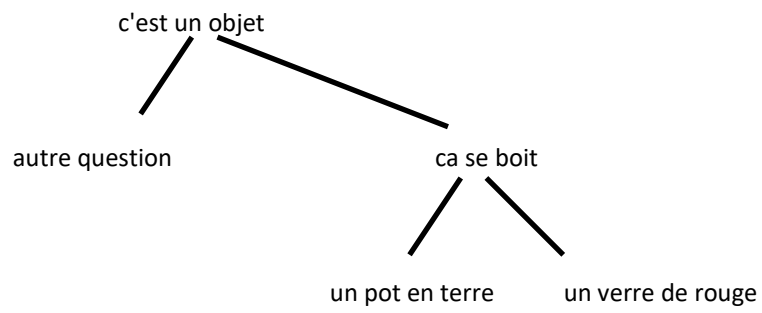
c'est un pastis

c'était bien ca ?

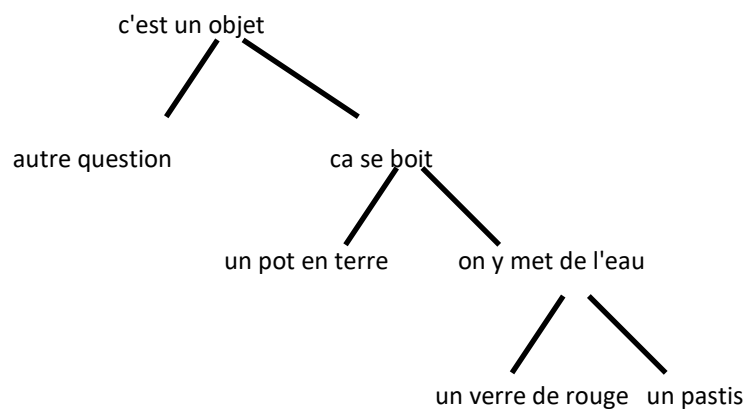
.oui

1.3 Fonctionnement

Les questions sont chaînées entre elles par deux chaînages : l'un correspond à la réponse OUI et l'autre à la réponse NON. selon la réponse du joueur il suffit donc de suivre le bon chaînage et d'afficher la nouvelle question. Si il n'y a plus de chaînage, la question est en fait ce que le programme propose comme réponse. Si le joueur s'en contente, tant mieux, sinon il doit fournir une nouvelle question que l'on va intercaler dans le chaînage.



devient donc :



1.4 Listing de Sphinx

```
10 DEFINT A-Z
20 TAILLE=100
30 DIM COD(TAILLE,2)
40 DIM RESULT$(TAILLE,2)
60 DIM QUEST$(TAILLE)
70 DIM T(TAILLE):DIM N(TAILLE)
90 NB=2
100 COD(1,1)=2
110 COD(2,1)=0
120 COD(2,2)=0
130 GOSUB 2010
160 NIV=2
164 PRINT "tu vas penser à quelq'un ou quelquechose, et je vais le deviner"
165 INPUT "ca y est, tu as choisi";C$
170 PRINT QUEST$(NIV)
171 INPUT " ";C$
180 IF C$="" GOTO 170
190 IF C$="oui" THEN REP=1:GOTO 370
210 IF C$="non" THEN REP=2:GOTO 370
220 IO=1-IO
230 IF IO=0 THEN PRINT "he, je te cause" ELSE PRINT "si je t'enmerde faut l'dire"
240 GOTO 170
370 IF COD(NIV,REP)<>0 THEN NIV=COD(NIV,REP):GOTO 170
380 REX=REP
390 PRINT "C'était ";RESULT$(NIV,REX)
400 INPUT "c'était bien ca ";C$
410 IF C$="oui" GOTO 160
420 IF C$="non" GOTO 600
430 IIO=1-IIO
440 IF IIO=0 THEN PRINT "mon coco, il faut répondre par oui ou par non"
      ELSE PRINT "oui ou non, ca me semblait simple pourtant"
450 GOTO 400
600 LL=LL+1
610 IF LL=1 THEN PRINT "je donne ma langue au chat, c'était quoi"
620 IF LL=2 THEN PRINT "j'abandonne, c'est quoi"
630 IF LL=3 THEN PRINT "qu'est ce que tu as encore été chercher"
640 IF LL=4 THEN PRINT "tu m'as collé, c'était quoi":LL=0
650 INPUT " ";C$
660 IF C$="" GOTO 600
670 PRINT "pose moi une question qui permette de distinguer ";C$;
      " de ";RESULT$(NIV,REX)
680 INPUT " ";Q$
690 IF Q$="" GOTO 670
700 PRINT "qu'aurai tu répondu si je t'avais posé cette question:"
710 PRINT C$;" , ";Q$
720 INPUT " ";R$
730 IF R$="oui" THEN REP=1:GOTO 900
740 IF R$="non" THEN REP=2:GOTO 900
750 OUIE=OUIE+1
```

```

760 IF OUIE=1 THEN PRINT "j'en étais sur, t'as rien compris"
770 IF OUIE=2 THEN PRINT "oulala, le petit poid va chauffer"
780 IF OUIE=3 THEN PRINT
      "tourne ton doigt sept fois sur le clavier et reponds moi calmement"
790 IF OUIE=4 THEN PRINT "on a tout notre temps":OUIE=0
800 GOTO 700
900 NB=NB+1
920 COD(NB,1)=0
930 COD(NB,2)=0
935 QUEST$(NB)=Q$
940 RESUL$(NB,REP)=C$
950 IF REP=1 THEN REP=2 ELSE REP=1
960 RESUL$(NB,REP)=RESUL$(NIV,REX)
965 RESUL$(NIV,REX)=""
970 REM cod(0,1)=nb
980 COD(NIV,REX)=1
990 COD(NIV,REX)=NB
1000 GOTO 160
2010 X=0 ;NIV=1;REP=1
2020 OPEN "sphinx.txt" FOR INPUT AS #1
2030 IF EOF(1) THEN RETURN
2040 LINE INPUT #1,C$
2050 QQ=INSTR(1,C$,"*")
2055 RR=INSTR(1,C$,"$")
2056 IF RR<>0 THEN QQ=RR
2060 IF QQ<>0 GOTO 3000
2070 NIV=NIV+1
2080 X=X+1
2090 T(X)=NIV
2100 QUEST$(NIV)=C$
2105 Y=T(X-1)
2110 COD(Y,REP)=NIV
2130 N(Y)=N(Y)+1
2135 REP=1
2140 GOTO 2040
3000 Y=T(X)
3010 COD(Y,REP)=0
3020 RESUL$(Y,REP)=C$
3030 N(Y)=N(Y)+1
3040 IF REP=1 THEN REP=2 :GOTO 2040
3050 X=X-1
3055 IF X=0 GOTO 4000
3070 IF N(T(X))<>2 GOTO 2040
3080 REP=2
3090 GOTO 3050
4000 CLOSE #1
4005 RETURN

```

Il y en a une version en pascal dans le jeux suivant (pmais), avec un autre tipe de fichier sphinx.txt

1.5 Fichier de données de Sphinx - sphinx.txt

dans ce fichier, les décalages par des tabulations sont significatifs. ils reflètent l'organisation des données dans la mémoire. des lignes également décalées correspondent à la réponse NON. les réponses se trouvent donc être dans chaque groupe les lignes les plus décalées.

```
c'est une personne
  une personne vivante
    c'est une personnalité du spectacle
      *la mère Denis
    c'est une personnalité politique
      c'est un scandaaaaaaaaaaaaale
        *marchais
        *chirac
        *toi
  c'est un personnage historique
    c'était un roi
      *louis et des poussières
    il a inventé l'école
      *charlemagne
      *napoléon
  c'est un personnage de légende
    *mélusine
    *asterix
c 'est un animal
  il a des poils
    il a des griffes
      des griffes rétractiles
        *un chat
      c'est un animal domestique
        *un chien
        *un lion
    ça a des cornes
      *une vache
  on monte dessus
    *un cheval
  il a une trompe
    *un éléphant
  ça a un long cou
    *une girafe
    *un dinosaure
  ça a des plumes
    c est un oiseau de proie
      *un aigle
      *un rossignol
  ça a des écailles
    ça nage
      c'est gros
```

*une baleine
 ça vit dans la mer
 *une sardine
 ça vit en eau douce
 *une truite
 \$alors ducon, ça vit en eau trouble
 ça rampe
 c'est un serpent
 *une vipère
 *un lézard
 \$et mon pote tu réponds n'importe quoi
 c'est tout petit
 ça vole
 ça pique
 ça fait du miel
 *une abeille
 c'est rayé jaune et noir
 *une guêpe
 *un moustique
 *une mouche
 *une bactérie
 \$ça a une tête d'abruti comme toi
 c'est un végétal
 c'est un arbre
 *un chêne
 c'est une fleur
 *une rose
 \$il faudrait répondre d'une manière cohérente, dur pour toi
 c'est un objet
 ça se boit
 c'est jaune
 *un pastis
 *un verre de rouge
 *un pot en terre cuite
 c'est une abstraction ou une idée
 une qualité
 *la beauté
 un défaut
 *l'orgueil
 une religion
 *l'islam
 *le sexe
 *le néant

Et puis le programme s'est étoffé, et voici l'organisation du fichier sphinx.txt utilisé dans le programme pmais qui suit :

racine : Est-ce une créature ou un personnage imaginaire ?

oui : créature imaginaire

non : non imaginaire

créature imaginaire : Est-ce un personnage mythologique (grec, romain, ...) ?

oui : personnage mythologique

non : fantastique ou roman

fantastique ou roman : Est-ce un personnage fictif (roman, bd, ...) ?

oui : personnage de roman ou bd

non : personnage fantastique

personnage de roman ou bd : Est ce un personnage de roman ?

oui : roman

non : bd

bd : Est-ce un gentil gaulois ?

oui : astérix

non : tintin

roman : Est-ce un naufragé ?

oui : robinson crusoé

non : don quichotte

personnage fantastique : apparence animale ?

oui : apparence animale

non : apparence humaine

apparence animale : il crache le feu ?

oui : un dragon

non : un vampire

apparence humaine : il est très grand ?

oui : un géant

non : un fantôme

personnage mythologique : c'est le roi des dieux

oui : zeus

non : mars

non imaginaire : Est-ce une personne humaine précise (pas une fonction ni un métier) ?

oui : personne humaine

non : non personne précise

voir pmais/win64/debug/sphinx.txt pour la suite du fichier

un jeu d'aventure

Où Syntaxikos se perd dans les méandres du labyrinthe préparé par Semantika et comment il apprend le javanais, le pilotage des tapis volants et l'arabe.

1.1 Analyse syntaxique.

1.1.1 Dictionnaire

L'un des plaisirs du jeu est de découvrir ce qu'il faut faire, dans le cadre d'un "labyrinthe" de 120 lieux et en utilisant les mots d'un dictionnaire qui en contient environ 650. La description du labyrinthe et du dictionnaire est entièrement symbolique et est analysée par un automate spécifique, ce qui permet de rajouter facilement des lieux, des verbes, des objets, des monstres et des messages, le noyau du jeu lui-même restant inchangé. Le noyau du jeu, écrit dans les années 1970, n'utilise pas les techniques liées aux grands modèles de langage comme chatgpt, qui n'existait pas, mais un dictionnaire contenant des informations syntaxiques et sémantiques permettant une analyse par un noyau indépendant du jeu. La phrase est donc analysée syntaxiquement et sémantiquement par des attributs et pas par des statistiques.

Chaque mot a donc :

- **une liste d'attributs grammaticaux** : verbe, nom commun, nom propre, adjectif, genre masculin/féminin, mot vide (article, préposition ...), prépositions avec/dans/sur/sous. Certains mots ont un attribut spécifiant qu'ils peuvent avoir plusieurs formes finales. cela permet de reconnaître par leur racine les verbes mêmes conjugués.

- **une liste d'attributs sémantiques** : arme, puissance de l'arme, objet précieux, objet magique, petit/grand, léger/lourd, solide/fragile, liquide, objet pouvant en contenir d'autres, nombre et taille de ces objets, nom abstrait, monstre, monstre bénéfique, monstre malfaisant, force du monstre, nombre....

- **une liste d'attributs syntaxiques** :

Pour les noms, possibilités d'être complément d'objet direct de tel et tel classe de verbes : prendre/poser, ouvrir/fermer, casser, boire, manger, attacher, chanter, jouer, danser, monter/descendre, creuser.

pour les noms encore, possibilités d'être instrument (complément d'objet indirect, « avec la hache ») de tel et tel verbe noté au participe présent : ouvrant, défonçant, allumant, attachant, creusant, coupant.

ex: porte nom, féminin, passage, prendre, ouvrir, défoncer
 chocolat nom, masculin, manger, prendre, petit, léger, bon
 clé nom, féminin, prendre, petit, léger, ouvrant

pour les verbes, constructions possibles avec, chacune, la préposition éventuelle et le ou les compléments. chaque construction est annoncée du caractère "\$" et contient, séparés par des virgules, la liste des attributs que doit avoir chaque mot composant la phrase..

ex : mange \$manger
 donne \$donner, à,monstre
 ouvre \$ouvrir,\$ouvrir, avec, ouvrant

Pour montrer que le complément du verbe prendre est soit un objet prenable (un chocolat, une hache) ou un passage (une porte, un chemin), on indique que le verbe a deux constructions, ayant des sens totalement différents.

ex : prend \$prendre, \$passage
 tire \$à,arme,sur,monstre,\$sur,monstre,avec,arme,\$objet

1.1.2 Fonctionnement de l'analyse

Dans un premier temps, les mots sont extraits de la phrase en supprimant les mots dits "vides" de sens. Le premier mot restant doit alors être un verbe, sinon la phrase est réputée non reconnue. Si la phrase est à l'impératif, c'est vrai. sinon il risque d'y avoir un pronom devant le verbe; qu'à cela ne tienne, les pronoms sont mis avec les articles dans la liste des mots vides. Les mots suivants sont recherchés dans le dictionnaire. lorsqu'un mot n'est pas reconnu, il y a deux solutions : ou bien rejeter la phrase, ou bien la conserver en en supprimant la fin..

Pour chaque mot reconnu, la présence de l'objet ou du monstre est vérifiée dans le lieu courant, à moins que le mot ait l'attribut abstrait, qui le rend accessible en tous lieux.

Il faut alors mettre en correspondance les constructions du verbe avec cette liste de mots. Lorsque l'on trouve correspondance mais qu'il reste des mots non analysés en fin de la phrase, il y a là aussi deux solutions : ou bien rejeter la construction, ou bien ignorer les mots supplémentaires; pour un jeu, la 2ème méthode donne de meilleurs résultats. le passage d'une méthode à l'autre est simple.

Pour les verbes ayant plusieurs constructions, en cas d'échec de la mise en correspondance avec une construction, on essaye la construction suivante

Si un complément de la construction n'est pas présent dans la phrase frappée, le programme le demande avec une question appropriée aux attributs du complément attendu. l'analyse du complément est stricte, le mot doit être reconnu. On ne note pas les adjectifs dans les constructions car tout nom peut avoir ou ne pas avoir d'adjectif.

Le résultat de l'analyse est donc :

- le code du verbe
- le numéro de la construction du verbe
- le code du 1er nom, ou -1
- le code de son adjectif, ou -1
- le code de la préposition
- le code du 2ème nom, ou -1
- le code de son adjectif, ou -1

Si un nom est un nombre, sa valeur est traduite par le programme.

une phrase complète a donc la structure :

verbe nom adjectif préposition nom adjectif
verbe préposition adjectif nom préposition adjectif nom

d'où les phrases comprises :

.mange le chocolat
.ouvre la porte
.prend le chocolat
.prend le chemin
.donne le chocolat au gentil dinosaure
.ouvre la porte sud avec la clé
.tire au fusil sur le monstre
.tire sur le monstre avec le fusil

.ouvre phrase incomplète, entraîne la réponse :
je veux bien, mais quoi ?
la porte
si il y a plusieurs portes fermées
il n'est toujours pas content et demande :
laquelle ?
celle du sud et si , vous avez vraiment pas de bol,
elle est fermée à clé, il demande :
avec quoi ?
avec la clé
.ouvre la porte du sud avec la clé

.ouvre la hache phrase incohérente entraîne la réponse :
ouvrir ou fermer une hache, ça doit pas être facile.

1.2 Jeux de mots

Selon l'occasion, le programme peut comprendre aussi la même phrase en javanais :

avouvrave lava pavorte davu savud avavec lava clavé

ou en arabe :

élc al ceva dus ud etrop al ervuo

Dans le genre jeux de mots, il ne manquera pas l'occasion de vous faire le coup du comment vas-tu yau de poêle, ni du j'en ai marre, marrabout de même vous aurez droit à quelques poil au ... et à quelques ... de cheval. bon d'accord, vous allez me dire : y a mieux. M'enfin quand même ça surprend, au premier rabord. D'autre part, il n'aime pas être traité de nom d'oiseau et dans ce cas attend des excuses et finit même par boudier et arrêter de jouer. Enfin, au bout d'un certain temps de jeu, des messages sont envoyés, et puis le programme simule une erreur à l'exécution et semble rendre la main au DOS. par exemple, à la commande :

dir *.*

il répond :

no file found

D'où généralement une grande angoisse, et comme c'est un tendre

il ne repart que lorsqu'on l'appelle par son petit nom en faisant : pmais, lacourbe ou run pmais

1.3 Scénarios du jeu

Lorsqu'une phrase n'est pas reconnue, on passe dans le programme PSY, déjà décrit, pour répondre un peu n'importe quoi, et parfois on affiche une page web de gag. Lorsqu'elle est reconnue, et éventuellement complétée, le module d'analyse rend le code du verbe, le numéro de sa construction (1,2, ...) les codes des 2 noms et des deux adjectifs. un nom ou un adjectif absent ayant le code -1. Après cela, il n'y a plus qu'à inventer des scénarios désopilants et des gags époustouflant. le jeu étant donc un gigantesque aiguillage sur le code du verbe et une flopée de procédures pour traiter chaque verbe. Il y a évidemment une autre floppée d'indicateurs pour se rappeler de ce qui a été fait. Depuis une de ces procédures il est possible de réactiver le module d'analyse pour poser des questions particulières.

1.4 Fonctionnement

1.4.1 Initialisation

Au début du jeu, le programme recopie le fichier de description des lieux dans un fichier qui va contenir l'état évolutif de tous les lieux. c'est à dire la liste des objets qui s'y trouvent, le monstre éventuel du lieu et son état, la liste des moyens de sortie du lieu et leurs états.

1.4.2 Gestion des objets

A chaque lieu est associé une liste d'objets présents en ce lieu. Une autre table contient la liste des objets que le joueur transporte sur lui. Enfin à certains objets (sac, coffre, ...) est associée une table contenant la liste des objets qu'ils contiennent. Toutes ces tables d'objets sont modifiés par les verbes du genre prendre, poser, mettre, manger, boire, lancer,....

1.4.3 Gestion du score

Pour éviter de se répéter et pour calculer le score, lequel s'affiche grâce aux verbes : "score", et « détaille » une table d'indicateurs est mise à jour dès que le joueur fait une action réputée intéressante comme par exemple de traverser le miroir pour aller dans la quatrième dimension ou trouver et jouer de l'accordéon, ou parvenir à quitter le tapis volant sans tomber dans les douves. Le simple emploi d'un mot du dictionnaire et la simple visite d'un lieu font aussi évoluer le score. Si on est bien sage, le grand maître de ces lieux apparaît parfois et condescend à vous donner des bribes du dictionnaire ou même, mais alors là il faut être bien vu, il peut vous susurrer dans le creux de l'oreille quelques actions à faire, extraite de la table des actions intéressante, et il apprécie qu'on le remercie pour l'indice. Cet indice dépend du lieu où l'on se trouve.

1.4.4 Passage d'un lieu à l'autre

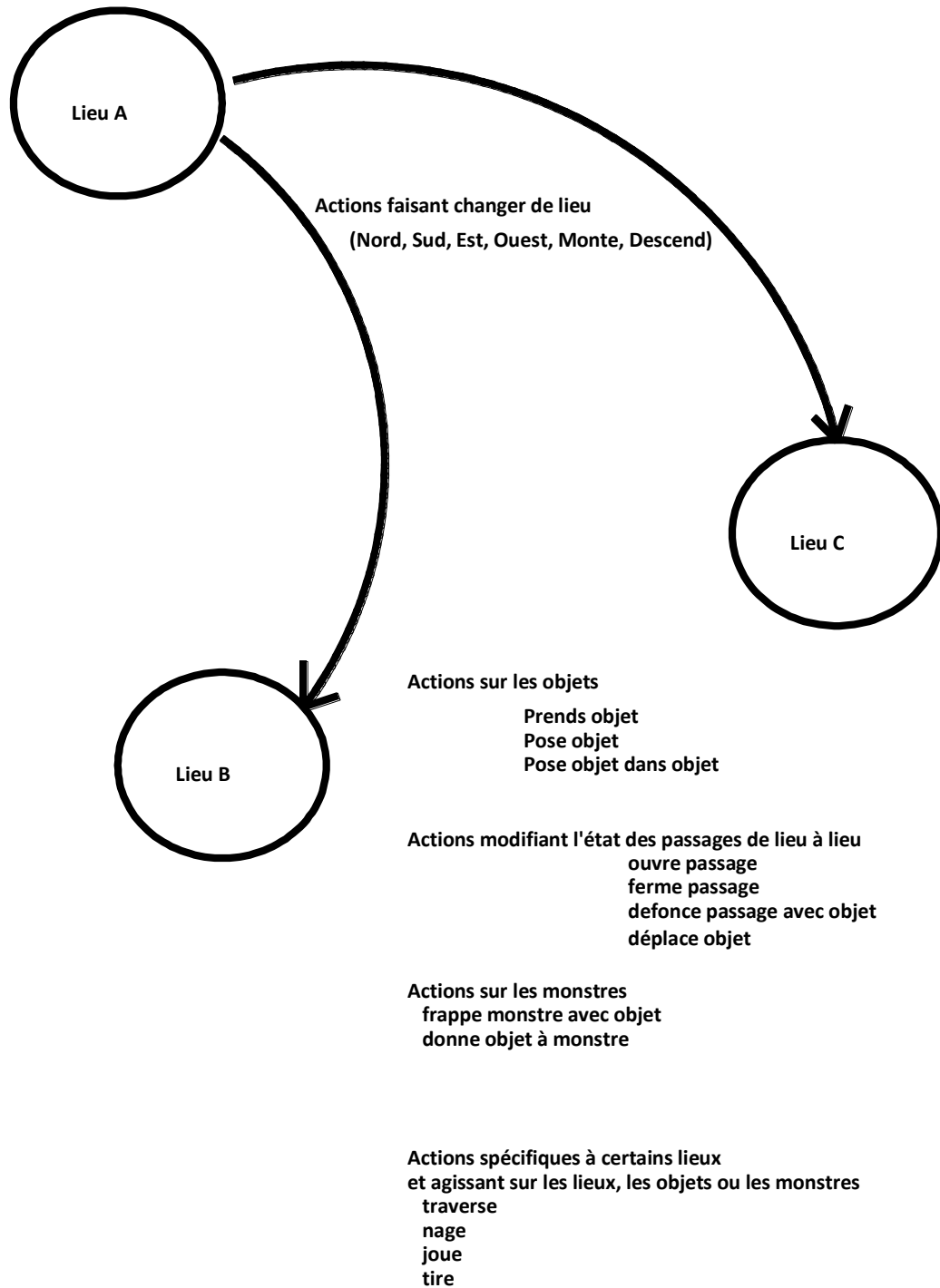
L'état des moyens de passage d'un lieu à l'autre varie par les verbes : ferme, ouvre et défonce ou par d'autres propres à certains passages particuliers ou secrets.

Les déplacements d'un lieu à l'autre se font par les verbes du genre : va, monte, descend, avance, droite, nord... et par d'autres propres à des passages particuliers comme par exemple de suivre les parallèles pour arriver dans l'espace dual ou de placer l'échelle contre le mur pour entrer par la fenêtre du 1^{er} étage, ou déplacer la stère de bois pour faire apparaître un passage souterrain.

1.4.5 Gestion des monstres

La gestion des monstres n'est pas très élaborée, Pour la plupart on ne peut pas leur faire grand chose à part les frapper, les tuer, leur tirer dessus, ... ou leur donner des trucs divers qu'on a trouvé par ci par là. Il y en a un qui aime bien jouer et rejouer avec vous à toutes sortes de jeux, il est très gentil; mais en général on a plutôt intérêt à décaniller vite fait, si on le peut. On peut aussi les charmer avec des instruments de musique ou leur chanter quelque chose.

L'espace de jeux se décrit par le graphe :



Objets pouvant en contenir
armoire, sac

Objets utiles à ramasser car pouvant servir dans d'autres lieux
pelle, drap, hache, couteau, anneau, clé

Objets utiles dans le lieux ou ils sont
lit, vanne, téléphone, bateau

1.4.6 Sauvegarde

A certains moments critiques, il est intéressant de sauvegarder l'état actuel du jeu pour pouvoir le reprendre plus tard, soit tout simplement le lendemain, soit après une action ayant entraîné un résultat non désiré. Cela se fait par le verbe "sauve", lequel provoque la recopie de toutes les variables dans un fichier, et par le verbe "restaure" qui permet de les relire.

1.4.7 Description faite au joueur

Tout ce qui se trouve dans la description du lieu courant et qui n'est pas secret est affiché : une phrase générale de description, une phrase éventuelle dépendant de l'état du lieu, la liste des moyens visibles de sortie, la liste des objets présents, la liste des objets que le joueur porte sur lui, et la présence éventuelle d'un monstre. Ce qui est secret à un moment ne le sera plus lorsque l'action correspondante aura été faite, comme de déplacer la stère de bois pour voir le souterrain, ou comme défaire le lit pour voir apparaître le drap, le prendre, l'attacher au lit, défoncer la fenêtre et lancer le drap par la fenêtre pour pouvoir sortir de la prison.

1.4.8 Messages

Les messages envoyés au joueur sont de plusieurs types :

- description générale d'un lieu
- description précise de l'état d'un lieu
- description d'un monstre
- phrase non comprise
- phrase incomplète, demande de précision
- action absurde, le verbe et ses compléments ne vont pas ensemble
- action impossible, l'objet n'est pas là ou une autre action doit être faite au préalable
- résultat d'une action

certains des messages ont une partie variable : le nom d'un monstre, d'un objet ou d'un moyen de passage.

1.4.9 le hasard

Dans les mêmes conditions, les réactions du programme ne sont pas toujours les mêmes : les monstres ne sont pas toujours à leur place, les coups qu'ils donnent ou qu'on leur donne portent plus ou moins, les messages d'encouragement ou d'échec varient eux aussi, grâce à l'utilisation d'un nombre aléatoire qui permet de déterminer l'action à réaliser ou le message parmi les N possibles.

1.4.10 les indices

Des indices sont distillés au fur et à mesure du jeu. Au delà de certains nombre de points :

- Un texte d' « aide » peut être demandé
- Lors de la description d'un lieu, afficher, associés à chaque passage, les noms des lieux voisins déjà visités
- Détailler le score
- Demander un indice – un élément de la table des actions intéressantes, lié ou non au lieu courant.
- Demander où est tel objet
- Demander à quoi sert tel objet, extrait les lignes de la table des actions intéressantes
- Demander le schéma de la zone courante, ou d'une zone particulière (site, ferme, château, étage, douves, métro, labyrinthe, souterrain, espace).

1.4.11 L'histoire familiale

Dans la description des lieux, est ajouté une phrase relatant un fait familial. Chaque lieu est associé à une ou plusieurs photos nommée du numéro du lieu : x.jpg (ou 3 nommées x_1.jpg et x_2.jpg) qui sont affichées dans 2 fenêtres à l'arrivée au lieu, et extraites du répertoire images. S'il n'y en a pas ou s'il n'y en a qu'une seule, la fenêtre du haut est remplie par une photo prise d'un autre répertoire (images_quizz), et celle-ci est légendée.

Il est possible de demander des précisions :

qui est ce (photo du haut)

Ou qui est ce en haut, ou qui est ce en bas

Le fichier liste_infos contient les informations nécessaires pour répondre à la question :

qui est prénom nom

chaque ligne de liste_infos contient le nom d'un personnage (ou d'un lieu) une tabulation et une information le concernant.

famille\prénom nom prénom nom est fils de x et y, époux de z

famille\prénom nom (2) s'il y a plusieurs fichiers jpg pour ce personnage

et

famille\prénom nom (x) s'il n'y a pas de fichier jpg ou plus d'infos que de fichiers

Dans quelques cas, il y a des homonymes, qu'on distingue par un numéro : pierre 3 mosnay de boishéraud

Il peut y avoir un nom seul, lorsqu'il y a un jpg sans info

Le fichier liste_inf contient les quelques légendes des fichiers du répertoire images

un répertoire « famille » n'est visible que aux utilisateurs qui ont choisi cette ligne de descendance au début du jeu. Certaines photos doivent être visibles par plusieurs descendances, il a donc fallu faire des regroupements. Et certaines personnes apparaissent dans deux regroupements : kerviler, mosnay, rossulien, blois, petit, paris, albin, roussel, perelle, laroque, lhéritier, bignan

Deux jeux utilisent ces données :

Un quizz invente, à l'aide du programme journal, des fausses histoires familiales et glisse une information exacte, qu'il faut retrouver.

Un autre jeux propose la photo d'un personnage ou d'un lieu, qu'il faut identifier parmi une dizaine de noms.

1.4.11 Les autres jeux

Les uns se déroulent hors du jeu principal, il faut terminer le jeu pour revenir à l'aventure elle-même. Ainsi un jeu de devinettes, un jeu d'énigmes, les deux jeux familiaux.

Les autres fonctionnent par des verbes déclarés comme ceux de l'aventure.

Il est enfin possible de demander au jeu de raconter un conte, un article de fait divers, une pièce de théâtre, une archive familiale, un proverbe.

raconte moi un conte

1.4.11 Les autres programmes

En plusieurs occasions, un programme extérieur est lancé dans une autre fenêtre. Par exemple pour afficher plein écran une photo (les schémas des zones du jeu) ou une vidéo, ou une page web, ou pour lancer un programme de fractales. Ceux-ci sont en java, et évidemment java doit être installé.

Il y a deux versions du jeu : lacourbe et rossulien. Les fichiers de images_quizz sont communs (et ne sont physiquement que dans le répertoire courbe). les fichiers liste_infos sont les mêmes, les 2 fichiers liste_inf et les 2 répertoires images sont différents. Les fichiers mot, vide et attrib

sont les mêmes ou presque. Nœud est évidemment différent, ainsi que le directory images, lié aux noeuds. phrase est presque le même. (les douves sont devenues l'odet et le duc de mercoeur est devenu le montagnard et quelques actions diffèrent).

1.5 Fichiers de données - dictionnaire

Plusieurs fichiers décrivent les mots, les lieux et les phrases. certaines entités sont définies par ce que l'on affiche au joueur et par un symbole auquel on fait référence ailleurs (éventuellement dans un autre fichier et même dans le programme par le biais du fichier de déclarations "consmais.pas") Les mots y sont séparés par des espaces, des virgules ou des tabulations. la tabulation et la virgule séparent les entités, l'espace sépare les mots d'une même entité. Une ligne commençant par le caractère \$ spécifie des attributs communs à tous les mots définis par la suite. une ligne commençant par # spécifie un symbole à déclarer dans le fichier de constantes consmais.pas, pour y faire référence dans le programme.

Tous ces fichiers textes sont lus par le programme cmais, qui construit la description informatique (fichiers .mai) et crée le fichier consmais.pas, qui doit être recopié dans courbe (ou rossulien) et permettre une compilation de ce dernier.

1.5.1 Mots vides - vide.txt

Ce sont les mots dont la présence ou l'absence dans la phrase est indifférente. Certains pourraient facilement être déclarés en adjectif, d'autres plus difficilement en préposition, mais il faudrait alors reconsidérer les constructions des verbes. Certains sont en fait des bribes de mots composés pour simplifier leur gestion. (guêpes pour nid de guêpes, conserve pour boîte de conserve, duc pour duc de mercoeur, on peut donc ne parler que de boîte, de nid ou de mercoeur))

```
*
* *****
* * definition des mots vides *
* *****
*
$    vide
*
il    ils    elle    elles    celle    pas    celui    mais    es
êtes  donc   le     la     les     un     une     des    de d    ça vais
du    l      hep?   au     a à     aux    alle?   va     et     depuis quoi
vas   vers   en     tas    paire   morceau    dis    dites  répon?
abominable    méchant    mère    père    vous    te     ta     tes    tu
je     j      t      sa     son     ton     notre   votre   vos ce  que    programme
amour
leur   leurs   ses     nos    ma     mon    mes    lui     me duc vieux  secret
m      par    petit?  grand? long?  large? mince? étroit? calcule?
toi    moi    en     poudre conserve    vélo    photo    pompe  décerveler    bois
vide   guêpe? salamandre    travers contre musqué?
```

a et à ont été déclarés, abusivement, vides donc on a comme construction, plutôt que celles données plus haut en exemple théorique :

donne \$donner,monstre

tire \$arme,sur,monstre,\$sur,monstre,avec,arme,\$objet

1.5.2 Passages d'un lieu à l'autre - pass.txt

Ce fichier établit la correspondance entre les mots utilisés dans la description des lieux et les mots à utiliser pour générer les phrases en français, en particulier lors de la description des lieux. Le premier mot de chaque couple indique par un code l'action à faire sur le passage pour pouvoir l'utiliser.

```
*
* *****
* * Définition des passages *
* *****
*
$      pass,masc
meporte,metro      taxi,taxi      plafond,passage
pepierre,petit? pont de pierre      débarc,debarcadere
pont,pont      embarc,embarcadere      grillage,grillage
couloir,couloir      passage,passage      chemin,chemin
jeu,passage      escalier,escalier popierre,pont de pierre
souponrail,souponrail      tableau,passage
$ pass,masc,secret
miroir,miroir
$      pass,fem
armo, couloir      carpette,trappe
porte,porte      clporte,porte      enporte,porte
rue,rue      drfenetre,fenetre      pelevi, pont levis à contrepoids
sotrappe,trappe      pltrappe,trappe      grille,grille
clgrille,grille      herse,herse      clherse,herse
fenetre,fenetre      enfenetre,fenetre      stere,souponrail
$      pass,secret
panier,panier      barque,barque      arbre,arbre
douve,douve      parallele,parallele      cheminee,cheminee
renvers,renvers      accordeon,accordeon      madrier,madrier
echelle,echelle      tapis,tapis      dehors,dehors
```

1.5.3 Attributs des mots - atrib.txt

Ce fichier définit les attributs que l'on peut utiliser pour définir les mots dans le dictionnaire et les messages d'erreurs correspondants. (le caractère # sera remplacé par un nom et son article)

Le mot clé est séparé du message qui lui correspond par une tabulation.

```
*
* *****
* * Définition des attributs *
* *****
#phatri
masc
fem
plur
```

partde	
monstre	# n'est pas un monstre
verbe	
nom	
adj	
dir	# n'est pas une direction
noeud	
vide	
pass	# n'est pas un moyen de passage
conj	
prend	prendre ou poser # ne me semble pas aisé, surtout pour toi
prenp	prendre # me semble curieux
ecr	écrire avec # n'est pas dans mes cordes
petit	
leger	
grand	
lourd	
ouvre	ouvrir ou fermer #, on aura tout vu
ouvrant	ouvrir avec #, ça doit pas être facile
defonce	défoncer # c'est au dessus de tes possibilités
defoncant	défoncer avec #, c'est pas très efficace
facil	
fragil	
solide	
deplace	déplacer #, c'est l'heure de ta crise
danse	danser #, allons bon
doux	
fort	
arme	#, c'est pas une arme bien terrible
mauvais	
nompropre	
montre	# n'est pas sujet à un plan
corps	
contienttu as déjà vu quelqu'un mettre quelquechose dans #	
abstrait	
precieux	# ne me semble pas très précieux
magique	# n'a aucun effet magique
ami	
aleat	
mange	voilà que tu veux manger #, bon appétit
bois	boire # !! et allez donc, voilà que ca te reprend
lire	lire # n'est pas évident
bon	
avec	
dans	
allume	allumer # ne doit pas être très facile
monte	monter #, si ca t'amuse
creuse	creuser #, oh du calme !

creusant	ça va pas être facile de creuser avec #
plonge	
traverse	traverser # ça doit être dur, tu crois pas ?
chante	on ne chante pas #, à ma connaissance
coupant	# pour couper, c'est pas terrible
attache	attacher # tu crois que ça va faire quelque chose ?
attachant	attacher avec # ça va tenir?
jouant	tu as déjà joué avec # ? moi non.
remplir	remplir # est bizarre
coupe	couper # va être dur, surtout pour toi
allumant	allumer avec #, je ne crois pas que ça fasse beaucoup d'effet
nage	
objet	
nombre	ce n'est pas un nombre ça
lieu	
sur	
sous	
con	
lieu	
tourne	tourner # c'est comme pisser dans un violon
toujours	
avant	
apres	
random	
couleur	
pejoratif	
jeux	jouer #, c'est pas terrible
tout	
jeuuuuuuu	
vide	
ennemi	
f	
unique	
pas	
partir	
secret	
eau	
avion	
devoile	
pour	
fractal	
ramasse	ramasser # me semble curieux

1.5.4 Construction des verbes - verbe.txt

Ce fichier donne les constructions possibles de chaque verbe. le caractère \$ signale le début d'une construction. Les mots qui suivent sont les attributs que doivent avoir les mots correspondants de la phrase analysées. Si une construction n'est pas reconnue, le programme essaye la suivante comme de bien entendu !! Si il n'y a pas de "\$" ou si il est seul, cela veut dire que le verbe doit être seul dans la phrase, aux mots vides près. Le "?" signale que la fin du verbe est variable, la comparaison s'effectue alors uniquement sur les premiers caractères du mot, et cela permet de reconnaître les formes conjuguées du verbe. ce processus est aussi valable pour le fichier des mots, qui suit. Si il y a plusieurs verbes sur une ligne, cela signifie qu'il sont synonymes.

```
*
*
* *****
* * Définition des adjectifs et verbes de direction *
* *****
*
$      dir,adj,verbe
*
n,nord,milieu,devant,avance?
s,sud,arriere,derriere,recule?
e,est,droite
o,ouest,gauche
ne,nord-est
no,nord-ouest
se,sud-est
so,sud-ouest
h,haut,air
b,bas,descend?
*
* *****
* * Définition des verbes *
* *****
*
$      verbe
*
qui
qu
pren?,emporte?,emmène?      $prend,$prenp
ramasse?      $ramasse
pose?,dépose?,place?,laisse?,met?      $prend,dans,contient,$prend
score,points
suivre,suit      $objet
regarde?,observe?,fouille?      $objet,$conj,objet,$
détail?
ouvr?      $ouvre,avec,ouvrant,$ouvre
sézame
ferme?      $ouvre
enfonce?,défonce?,démoli?,casse?      $defonce,avec,defoncant
```

tue?	\$monstre,avec,arme
poignarde?	\$monstre
frappe?,attaque?,cogne?	\$monstre,avec,arme
claque?	\$objet
acheter,achète	\$prend
scanne?	\$lire
imprime?	\$lire
transcri?	\$lire
photocopie?	\$lire
rédige?	\$lire
attache?,accroche?	\$attachant,objet
raccroche?	\$objet
détache?,décroche?	\$objet
soulève?,déplace?,roule?,enlève?,ote?	\$deplace
mélange?	\$objet,avec,objet,\$objet,objet
crie?,hurle?	\$monstre,\$
chante?,danse?,siffle?	\$chante,\$
entre?	\$dans,lieu,\$dir,\$
embarque?	
débarque?	
accoste?,aborde?	
mosnay,boishéraud	
boi?	\$bois,nombre,\$bois
mange?	\$mange
cherche?,recherche?,trouve?	
couche?	
défai?	
embrasse?	\$monstre
coupe?,taille?	\$coupe,avec,coupant
creuse?,bèche?	\$creuse,avec,creusant,\$avec,creusant
alle?,va,vas	\$dir
donne?,offrir?	\$abstrait,\$prend,monstre
lève,lever,élève?,dresse	\$objet
allume?	\$allume,avec,allumant
éteint?	\$allume
excuse?,pardon	
jette?,lance?	\$fractal,\$arme,monstre,\$prend,dans,objet,\$arme,conj,monstre,\$prend,pass
lire,lis	\$lire
montre?,affiche?	\$abstrait,montre,\$objet,monstre,\$objet
nage?,plonge?	\$dans,nage,\$
pend?	\$prend,avec,attachant
photographie?	\$objet
saute?	\$dans,objet,\$sur,objet,\$
passe?,traverse?	\$traverse,\$prenp
enjambe?	\$objet
merde,con,idiot,bête,stupide,enfoiré,foutre,fouttre,gueule	
cornegid?	
merdre	

chandelle
 zut
 suis
 es,êtes
 dis,dites
 arrête?
 rejoue? \$jeux,\$chante,avec,jouant,&
 joue? \$jeux,\$chante,avec,jouant,\$chante,sur,jouant,&
 azerty
 qwerty
 triche \$nombre
 retourne?,téléporte? \$objet
 raz
 teste
 appuye?,pousse? \$objet
 tourne? \$dir,\$tourne
 monte?,grimpe?,escalade? \$dans,monste,&monste,\$
 descend? \$dans,monste,&monste,\$
 oui
 non
 test?
 inconnu
 bonjour
 merci,remercie? \$pour,abstrait,\$monstre,\$
 rempli? \$contient,prend,\$remplir,avec,ecr
 fin,lo,log,salut, revoir,bonsoir,stop,bye,quit
 boot,reboot
 atterri?
 fai?,appelle?,téléphone?,compose? \$nombre,\$objet
 sauve?
 restaure?
 abaisse? \$objet
 manoeuvre? \$objet
 tire? \$arme,sur,monstre,\$sur,monstre,avec,arme,\$objet
 sert \$objet
 ou,où \$monstre,\$objet
 rame?,godille?
 yau
 comment
 poil?
 rate?
 pisse?,urine?
 speak
 up,look,save,take,down
 effeuille? \$objet
 abracadabr?
 garçon,commande ? \$mange
 taxi

marche?
 pompe?
 compare?,pèse? \$objet
 assied?
 transforme?,rédui?,intégrer,intègre?,transcende?,projete? \$objet
 help,sos,aide
 additionne?,ajoute?,somme \$nombre,nombre,\$nombre
 soustrai?,retranche?,différence \$nombre,nombre,\$nombre
 multiplie?,produit \$nombre,nombre,\$nombre
 divise?,quotient \$nombre,nombre,\$nombre
 demande? \$abstrait
 raconte?,récite?,invente? \$abstrait

*

* *****

* * Conjonctions *

* *****

*

*

\$ conj,avec
 avec,ec \$objet
 \$ conj,dans
 dans \$objet
 \$ conj,sur
 sur \$objet
 \$ conj,pour
 pour \$objet
 \$conj
 dessus \$objet
 sous,dessous \$objet
 plus

1.5.5 Les mots - mot.txt

Ce fichier contient les mots et leurs attributs syntaxiques et sémantiques. Lorsqu'il y a plusieurs mots sur la ligne, cela signifie qu'ils ont les mêmes attributs mais ils ne sont pas synonymes. Lorsqu'il y a un caractère "&" cela indique le nom complet à afficher.

* *****

* * definition des mots *

* *****

*

\$ nom
 *
 porte fem,ouvre,defonce,prenp,pass,facil
 glace,vitre fem,ouvre,defonce,pass,fragil
 fenêtre fem,ouvre,defonce,pass,fragil
 carreau masc,ouvre,defonce,pass,fragil
 bouteille? fem,bois,prend,petit,leger,defonce,fragil,contient

carafe fem,bois,prend,petit,leger,defonce,fragil
 chocolat? masc,mange,prend,petit,leger,bon,precieux
 patates plur,ramasse,mange
 haricots plur,ramasse,mange
 choux plur,mange
 champignons plur,ramasse,mange
 mures plur,ramasse,mange
 noisettes plur,ramasse,defonce,mange
 crêpe fem,mange
 feu masc,abstrait,allume
 chlorate masc,partde,allume,prend,petit
 boîte,&boîte? de conserve vide fem,prend,petit,leger,contient,allume
 tube,&tube? de pompe à vélo masc,prend,petit,leger,contient,attachant,allume
 auge fem,contient
 sucre,&sucré? en poudre masc,partde,mange,prend,petit
 pétard masc,prend,petit,allume
 buche fem,prend
 ordinateur masc,prend
 imprimante fem,prend
 photocopieuse fem
 briquet masc,prend,petit,allumant
 bougie fem,prend,petit,allumant
 couteau masc,prend,petit,arme,creusant,coupant
 pelle fem,creusant,arme,grand,defoncant,prend
 pioche fem,arme,fort,creusant,coupant,grand,defoncant,prend
 danse fem,abstrait,chante,danse
 drap masc,prend,leger,defonce,facil,attachant,attache
 rat?,rat? musqué masc,monstre
 escalier masc,prenp,pass,monte
 saucisson,paté masc,prend,petit,leger,mange,bon
 jambon masc,prend,petit,leger,mange,bon
 main,poing? masc,arme,doux,corps,creusant,defoncant,tout
 pied? masc,arme,doux,corps,defoncant,tout
 hache fem,prend,grand,arme,fort,coupant,defoncant
 bêche fem,grand,leger,prend,arme,fort,creusant
 lampe fem,prend,petit,defonce,allume
 mur masc,abstrait
 débarcadère masc
 ciseaux plur,prend,petit,leger,arme,doux
 grillage masc,ouvre,defonce,pass,facil
 carte,&carte?s plur,prend,petit,leger,defonce,allume
 appareil,&appareil? photomasc,prend,petit,leger,defonce
 dent,&dent?s plur,corps,coupant,tout
 plafond masc,pass,defonce
 machine,&machine? à décerveler fem
 téléphone masc,petit,leger,prend,defonce
 accordéon masc,petit,leger,prend,defonce,precieux,jouant,allume
 cornemuse fem,petit,leger,prend,defonce,precieux,jouant,allume

chant	masc,abstrait,chante
musique	fem,abstrait,chante
air	masc,abstrait,chante
pont	masc,prenp,pass,ouvre
bois	masc,partde,prend,petit,leger
rue	fem,pass,prenp
grille	fem,ouvre,pass,solide
herse	fem,ouvre,pass,solide
métre	masc,monte,prenp,pass,montre
soupirail	masc,ouvre,pass,solide
passage	masc,prenp,pass
couloir	masc,prenp,pass
trappe	fem,ouvre,defonce,pass
stère,&stère?	de bois masc,grand,lourd,deplace,defonce,devoile
tableau	masc,prend,grand,leger,defonce,facil,devoile
armoire	fem,contient,deplace,lourd,devoile,ouvre
carpette	fem,prend,petit,leger,defonce,devoile
madrier	masc,grand,prend,defoncant,arme,devoile
échelle	fem,prend,grand,lourd,monte,pass,defoncant,devoile
chemin	masc,prenp,pass
eau	fem,bois
embarcadère	masc
barque	fem,defonce,solide,prenp,pass,monte
bateau	masc,defonce,solide,prenp,pass,monte
douve,&douve?s	plur,bois,pass,plonge,traverse,mauvais,eau,montre
cheminée	fem,monte,pass
tapis	masc,prend,grand,lourd,defonce,solide,monte
score	masc,abstrait
panier	masc,prend,defonce,facil,monte,pass
polka	fem,danse,chante,abstrait
gavotte	fem,danse,chante,abstrait
valse	fem,danse,chante,abstrait
scottisch	fem,danse,chante,abstrait
mazurka	fem,danse,chante,abstrait
bourrée	fem,danse,chante,abstrait
gigue	fem,danse,chante,abstrait
branle	masc,danse,chante,abstrait
laridé	masc,danse,chante,abstrait
rondo	masc,danse,chante,abstrait
romance	fem,danse,chante,abstrait
andro	masc,danse,chante,abstrait
avantdeux	masc,danse,chante,abstrait
rame	fem,arme,prend,grand,lourd
clé?	fem,prend,petit,leger,precieux,ouvrant
vanne	fem,ouvre,solide
contrepoids	masc
ficelle	fem,prend,petit,leger,attachant,attache
marguerite	fem,prend,petit,leger,precieux

fiole fem,prend,petit,leger,precieux,defonce,fragil
 buste masc,prend,precieux
 moule masc,prend,precieux
 tirages masc,prend,precieux,plur
 formulaire masc,prend,lire,remplir,precieux
 questionnaire masc,prend,lire,remplir,precieux
 encre fem,prend,ecr,partde
 plume fem,prend,ecr
 armure fem,prend,grand,lourd,solide
 statue fem,solide,prend
 malle fem,contient,deplace,lourd,ouvre
 tête fem,corps,tout
 jambe fem,corps
 balance fem,prend,petit,leger,defonce
 chateau masc
 miroir masc,prend,petit,leger,defonce,fragil,transpose
 anneau masc,prend,petit,leger,precieux,arme,magique
 taxi masc,prend,pass,monte
 dictionnaire masc,prend,petit,leger,defonce,lire,facil,allume
 stylo masc,prend,petit,leger,defonce,facil
 puit masc,pass,monte,eau
 vin masc,partde,bois,prend,petit,leger,bon
 arbre masc,monte,pass,coupe
 fusil,carabine masc,prend,petit,leger,arme,fort
 revolver masc,prend,petit,leger,arme,fort
 sac masc,prend,petit,leger,contient,defonce,ouvre
 livre masc,prend,petit,leger,lire,defonce,facil,ouvre,allume
 livret?,&livret? d'archives familiales masc,lire,prend
 journal masc,prend,petit,leger,lire,defonce,facil,ouvre,allume
 balai masc,prend,grand,leger,arme,doux
 lit masc,deplace,lourd
 buffet masc,deplace,lourd,contient,ouvre
 placard masc,contient,ouvre,defonce
 coffre masc,deplace,lourd,contient,ouvre
 coffret masc,prend,petit,contient,ouvre
 poignard masc,prend,petit,leger,arme,fort,creusant,coupant
 bouton? masc,jeux
 terre fem,creuse,tout,contient
 sol masc,creuse,tout,contient
 corde fem,prend,petit,leger,attache,attachant,precieux
 allumette,&allumette?s plur,prend,petit,leger,defonce,facil,allume,allumant
 parallèle,¶llèle?s plur
 perpendiculaire fem
 nombre masc,nombre,abstrait
 résultat masc,abstrait,nombre
 tout prend,abstrait
 mosnay abstrait
 boishéraud abstrait

courbejollière abstrait,fem
 monstre masc
 pierre fem,prend,petit,leger,arme,jeux
 écu? masc,jeux,prend
 sphinx masc,jeux
 mot? masc,jeux
 quinze masc,jeux
 tictactoe masc,jeux
 dame masc,jeux
 énigmes fem,jeux
 rizièrè fem
 honte fem
 or masc
 calme masc,avion,deplace
 avion masc,avion,deplace
 averse fem,avion,deplace
 bruine fem,avion,deplace
 crachin masc,avion,deplace
 cyclone masc,avion,deplace
 pluie fem,avion,deplace
 tornade fem,avion,deplace
 vent masc,avion,deplace
 neige fem,avion,deplace
 nuage masc,avion,deplace
 stratus masc,avion,deplace
 cyrus masc,avion,deplace
 tempête fem,avion,deplace
 givre masc,avion,deplace
 fontaine fem,eau
 bouche fem,ouvre,tout
 tangente fem,prenp,abstrait
 barrique fem,defonce,deplace
 modif? fem,abstrait
 chaise fem,grand,prend,arme,defoncant
 usine? fem
 nurmidie fem
 chasse,&chasse? d'eau fem
 bave,&bave? de crapaud fem,prend,precieux
 langue,&langue? de salamandre fem,prend,precieux
 table fem,deplace
 philtre,&philtre? d'amour masc,prend,precieux
 sort? masc,abstrait
 nid,&nid? de guèpes masc,ouvre,defonce,contient
 frelon? masc
 canon? masc,arme,fort,lourd,prend
 siège masc,prend
 cadran,&cadran? solaire masc
 trousseau,&trousseau? de clefs masc,prend,petit,ouvrant

indice masc,abstrait
 plan masc,abstrait
 schéma masc,abstrait
 conte masc,abstrait
 poème masc,abstrait
 archive? fem,lire,prend,petit,jeux
 papier?,fakenews masc,lire,prend,petit,jeux
 famille?,quizz masc,abstrait,jeux
 proverbe masc,abstrait
 pièce fem,abstrait
 histoire fem,abstrait,jeux
 synthèse,chronique,généalogie fem,lire
 mandelbrot masc,fractal,abstrait
 newton masc,fractal,abstrait
 ifs masc,fractal,abstrait
 lsystem masc,fractal,abstrait
 attracteur masc,fractal,abstrait
 aide,help fem,abstrait
 chateau masc,abstrait,montre
 ferme fem,abstrait,montre
 site masc,abstrait,montre
 étage masc,abstrait,montre
 labyrinthe masc,abstrait,montre
 souterrain masc,abstrait,montre
 espace masc,abstrait,montre
 * *****
 * * les monstres *
 * *****
 *
 \$ nom,monstre
 *
 ankou masc
 dupond masc,nompropre,toujours,ennemi,paspartir
 denis masc,nompropre,random,ami,f
 mélusine fem,nompropre,ami,unique,f
 joueur masc,ami,toujours,f
 rate,&rate? passoncoup masc,nompropre
 tire,&tire? pasmal masc,nompropre
 soudards masc,plur,ennemi,toujours,paspartir
 hydre fem,fort,random,ennemi,paspartir
 cuistot masc,unique,ami,f
 jardinier masc,toujours,ami,f
 griffon masc,fort
 elf masc,unique,ami,f
 lutin masc,unique,ami,f
 momie fem,random,ennemi,paspartir
 pieuvre fem,fort,random,ennemi,paspartir
 squelette masc,random,ennemi,paspartir

dinosaure	masc,random,ennemi,paspartir
guet	masc,fort,random,ennemi,paspartir
mercoeur	masc,fort,toujours,ennemi,paspartir
dragon	masc,fort,random,ennemi,f
crapaud	masc,toujours,ami,f
bureaucrate	masc,toujours,ennemi
mouleur	masc,toujours,ami

*

* *****

* * adjectifs *

* *****

*

\$ adj

joie

gros,grosse,enorme avant

petit,petite avant

grand,grande avant

joli,jolie,beau,bel,belle avant

horrible,abominable avant,pejoratif

laid,laide,affreux avant,pejoratif

doux,douce,aimable avant

désagréable,déplaisant avant,pejoratif

bleu,bleue couleur,apres

blanc,blanche couleur,apres

rouge couleur,apres

vert,verte couleur,apres

noir,noire couleur,apres

solaire

1.5.6 La description des lieux - noeud.txt

Ce fichier contient la liste des moyens visibles ou cachés pour passer d'un lieu à l'autre (notés "."), la liste des objets (notés "*") et le monstre (notés "\$") présents au départ dans chaque lieu. Le numéro figurant près du nom du lieu n'est là que pour aider à la mise au point. Le symbole "<" introduit une description du lieu, de même le caractère '&' introduit une description du monstre hantant le lieu.

La liste des moyens de passage est organisée en triplet séparés par une tabulation :

type de passage, nœud d'arrivée, verbe de passage

* *****

* * DEFINITION DES NOEUDS *

* *****

*

\$ noeud,nom

*

boudoir boudoir 1

.porte,entree,o .fenetre,pejardin,e

*chocolat,carte,briquet,encre

<ici on aime les jeux de sphinx, fakenews, archive, énigmes, quizz, pierre
 <tictactoe, dame, écu, mot, histoire, quinze
 <et on y joue on y rejoue sans s'en lasser
 \$joueur
 &Le joueur est assis à une table, il semble très absorbé
 entree entree 2
 .porte,boudoir,e .porte,samanger,n
 .porte,toilettes,s .porte,grjardin,o .escalier,couloir_echo,h
 *plume
 toiletteschiottes 3
 .porte,entree,n
 *chasse
 samanger salle à manger 4
 .porte,pejardin,e .porte,entree,s .porte,couloir1,no
 .passage,cuisine,n .arroi,planque,o .cheminee,grchambre,h
 *armoie,cheminée
 planqueplanque 5
 .passage,lab1,o .passage,samanger,e
 <vous avez reçu le questionnaire pour assoir votre taxe d'habitation locative
 *questionnaire
 cuisine cuisine 6
 .passage,samanger,s
 \$cuiatot
 &Le cuistot
 *balance
 couloir1 couloir 7
 .porte,samanger,s .porte,pejardin,e .clporte,grcuisine,no
 .porte,passerel,so .escalier,palier1,h
 *bouteille,boite,chlorate,tube
 passerelpasserelle d'embarquement 8
 .porte,couloir1,ne .porte,espion,s
 avion avion 9
 .renvers,boudoir,saute
 espion avion espion 10
 .renvers,carrefour,saute
 .jeu,avion,h
 grcuisine grande cuisine 11
 .porte,couloir1,e .porte,sagardes,o
 *ciseaux
 sagardes salle des gardes 12
 .porte,grcuisine,e .porte,grentree,o
 .porte,nif,n .fenetre,grjardin,s
 *fusil
 grentreegrande entrée 13
 .escalier,palier2,h .porte,grjardin,s
 .porte,grsalon,o .porte,sagardes,e .dehors,étang,n
 *sac
 grsalon grand salon 14

.porte,grentree,e .porte,nif,n
 .porte,pesalon,no
 *buste
 \$dupond
 &l'abominable dupond la joie te tombe sur le poil
 &t'es pas près de t'en débarrasser
 pesalon petit salon de la tour 15
 .porte,grsalon,se
 <le petit salon est le lieu des veillées, où on joue à des jeux
 <de lettre et à des énigmes
 chmaitre chambre du maitre de céans 16
 .porte,cabinet1,o .porte,couloir_echo,no
 *appareil,journal
 cabinet1 cabinet de toilette 17
 .porte,chmaitre,e .porte,couloir_echo,ne
 couloir_echo couloir d'écho 18
 .escalier,entree,b .porte,cabinet1,s
 .porte,chfleur,n .porte,chmaitre,se .clporte,grchambre,ne
 grchambre grande chambre 19
 .porte,couloir_echo,o .tableau,pechambre,n
 *tableau
 pechambre petite chambre 20
 .porte,couloir_echo,o .passage,grchambre,s .accordeon,nirvana,joue
 <dans les tiroirs d'une commode, il y a plein de vieux papiers
 *accordéon,pétard,papier
 saubu chambre du père ubu 21
 .sotrappe,oubliettes,b
 *machine
 <Tu es actuellement dans la chambre du père Ubu
 nirvana grand nirvana 22
 .accordeon,pechambre,arrête
 <Il y a des petits oiseaux partout et des petites fleurs bleues
 <d'autres aventuriers sont là avec leurs instruments
 <qui un violon, qui une vielle, qui une cornemuse, qui sa voix
 <et vous jouez avec eux de douces mélodies à l'accordéon
 chfleur chambre à fleurs 23
 .porte,couloir_echo,s
 .porte,palier1,n .porte,saubu,e
 *table
 palier1 palier 24
 .clporte,chfleur,s .porte,chdenis,n
 .clporte,chenfants,o .escalier,couloir1,b
 chdenis chambre de la mère Denis 25
 .porte,palier1,s
 *téléphone
 \$denis
 &La mère Denis surgit comme une furie en hurlant :
 &mais c'est pas possible, vous laissez toutes les portes ouvertes !

&elle part, on entend les portes claquer
 chenfants chambre des enfants 26
 .porte,chgrise,o .porte,palier1,e
 chgrise chambre grise 27
 .porte,chenfants,e .porte,cabinet3,ne
 .porte,couloir3,n
 *cornemuse
 cabinet3 cabinet de toilette 28
 .porte,chgrise,s .porte,couloir3,o
 couloir3 couloir 29
 .porte,chgrise,se .porte,prison,s
 .porte,cabinet3,e .passage,palier2,o .passage,chrose,so
 prison prison 30
 .drfenetre,grjardin,s .plafond,gre nier2,h
 *chaise,lit
 chrose chambre rose 31
 .porte,couloir3,n
 *livre
 palier2 palier 32
 .jeu,gre nier2,h .escalier,gre ntrees,b
 .passage,couloir3,e .enporte,chroi,o
 *bouteille
 chroi kalifat 33
 .porte,palier2,s .clporte,debarras,o .porte,cabinet2,n
 <Chambre du kalife Haround al Rachid
 <des vapeurs d'encens et de myhrre et des mélopées tambourinesques
 <emplissent le volume spacio-intemporel.
 debarras débarras 34
 .porte,chroi,e .tapis,tapisvol,h
 *tapis
 tapisvol tapis volant 35
 .renvers,ndouves,saute .jeux,,vabracadabra
 <tu volettes gaiement au dessus des douves
 <(je suppose que tu vois ou je veux en venir !!!!)
 <et tu aperçois enfin le chateau dans son ensemble
 <il est constitué de 3 îles : le bosquet, le château, la ferme
 cabinet2 cabinet de toilette 36
 .passage,palier3,e .porte,chroi,s
 palier3 palier 37
 .passage,cabinet2,o .carpette,oubliettes,b .escalier,chtour,h
 *carpette
 oubliettes oubliettes 38
 .passage,souter3,n
 chtour chambre de la tour 39
 .escalier,palier3,b .porte,gre nier2,e
 .jeu,sallemiroir,donne
 \$mélusine
 &mélusine est là

<Il y a trois marguerites à effeuiller dans un vase
 grenier2 grenier 40
 .porte,chtour,e
 pejardinpetit jardin 41
 .embarc,dabrev,e .clporte,samanger,o .enfenetre,boudoir,so
 .chemin,grjardin,s .chemin,balançoire,n .porte,couloir1,no
 .douve,abreuvoir,plonge
 <le soir il y a des rats musqués qui passent et repassent dans les douves
 *nid,rat
 grjardin grand jardin 42
 .echelle,chgrise,h
 .pont,courferme,s .douve,bosquet,plonge .chemin,pejardin,se
 .passage,tocharbon,o .clporte,entree,e .clporte,grcuisine,n .chemin,puit,no
 .clporte,couloir1,ne .clporte,grentree,no
 <il y a une fenêtre ouverte au premier
 polevis petit pont levis 43
 .madrier,bosquet,h
 .pelevis,bosquet,s .chemin,deversoir,o .chemin,fabosquet,n
 .douve,bosquet,saute
 <les douves sont plus étroites à hauteur du pont levis
 coinlioncoin des lions 44
 .chemin,bardot,o .chemin,abreuvoir,s
 balançoire balançoire 45
 .chemin,pejardin,s .chemin,nif,o
 nif if 46
 .panier,npanier,h
 .porte,grsalon,so .porte,sagardes,se
 .chemin,balançoire,e .douve,bardot,plonge
 .pont,bosquet,no
 <Il y a de la lumière à la fenêtre du haut de la tour
 <et on entend un chant mélodieux et suave
 npanier panier 47
 .fenetre,chtour,o.renvers,ndouve,saute
 <tu te balance doucement au dessus des douves
 <et Mélusine, car c'est elle, est là dans la pièce
 <elle te fait signe de la rejoindre
 ndouve douves 48
 <L'eau est vraiment infecte, mais heureusement la rive n'est pas loin
 tocharbon tour à charbon 49
 .stere,souter1,b
 .passage,grjardin,e
 *échelle,stère
 souter3 souterrain 50
 .passage,souter1,no .passage,souter2,o .passage,souter4,e
 souter4 souterrain 51
 .passage,souter2,o .passage,champ,h
 souter1 souterrain 52
 .passage,tocharbon,h .passage,souter3,se .passage,souter2,so

souter2 souterrain 53
 .passage,souter1,ne .passage,souter3,e
 bardot écluse 54
 .chemin,coinlion,e .chemin,fabosquet,s .douve,nif,plonge
 *vanne
 fabosquet faux bosquet 55
 .chemin,bardot,n .chemin,polevis,s .douve,nif,plonge
 .chemin,deversoir,so
 *canon
 \$soudards
 &voila la troupe de soudards avinés que le duc de mercoeur
 &a rassemblés pour protéger son canon
 &et il prépare un mauvais coup pour la Courbejollière
 deversoir déversoir 56
 .chemin,ruisseau,s .chemin,fabosquet,ne .douve,bosquet,plonge
 .chemin,polevis,e
 *madrier
 ruisseauruisseau57
 .grille,potager,n .chemin,deversoir,ne .chemin,charmilles,e
 .douve,butte,plonge
 \$lutin
 &Le lutin te salue bien bas
 potager potager 58
 .grille,ruisseau,s.enporte,appentis,n
 *hache
 \$jardinier
 &Le jardinier est javanais et il te dit :
 &bavonjavour,jave avous avatavendavais
 appentis appentis 59
 .porte,potager,s
 *pelle,pioche,corde
 carrefour carrefour 60
 .chemin,caroue,n .chemin,vigne,ne
 *pioche
 <Salut à toi, temeraire aventurier
 fontainefontaine61
 .chemin,caroue,o .chemin,vigne,s
 *fontaine
 abreuvoir abreuvoir 62
 .chemin,caroue,s .chemin,coinlion,n
 .chemin,champ,e .douve,pejardin,plonge
 vigne vigne 63
 .chemin,fontaine,n .chemin,étang,e
 .chemin,carrefour,o .porte,taverne,s
 \$crapaud
 &Un crapaud énorme te tire une langue grande comme ça, bave
 &comme un crapaud seul sait le faire et s'éloigne à reculon
 étang étang 64

.chemin,vigne,so .chemin,champ,n
taverne taverne des aventuriers 65
.porte,vigne,n
&des groupes bruyants sont attablés en sirotant
&mais ton arrivée semble jeter un froid, et les conversations
&continuent à voix basse
champ champ 66
.stere,souter4,b .chemin,abreuvoir,n
.chemin,étang,s .chemin,culsac,se
.chemin,caroue,o
*stère
culsac cul de sac 67
.chemin,champ,no
bosquet bosquet 68
.chemin,butte,o .pelevis,polevis,n
.douves,grjardin,plonge
.pont,nif,ne .pepierre,tilleul,se
butte butte 69
.chemin,bosquet,b
caroue quatres chemins 70
.chemin,carrefour,s .chemin,ruisseau,o .chemin,abreuvoir,ne
.pepierre,grporte,n .arbre,nidpie,h .chemin,champ,e
*arbre,anneau
\$elf
&Un elf te salue bien bas et te tend un anneau en disant:
&le chateau de la Courbejollière que tu vois là bas contient
&d'immenses trésors protégés par des monstres maléfiques, prends
&l'anneau suprême que voici et garde le bien car trois fois seulement
&il te protégera
nidpie nid de pie 71
.arbre,caroue,b
*clé
courferme cour de la ferme 72
.chemin,cour,s .chemin,tilleul,o
.grillage,jaferme,ne .escalier,chvalet,h
.porte,saferme,e
jaferme jardin chinois 73
.grillage,courferme,no .porte,saferme,o .douves,abreuvoir,plonge
.passage,tourferme,s .porte,orangerie,so
\$dragon
&Un dragon énorme et crachant un feu d'enfer s'approche en ricanant
&il semble bien décidé à te cramer les moustaches
embarcadère embarcadère 74
.embarc,dembarc,embarque .chemin,tilleul,s
*barque
saferme salle de la ferme 75
.porte,jaferme,e .porte,chambre,s .porte,courferme,o
chvalet chambre des tortures 76

.escalier,courferme,b
 <un tas d'objets affreux te regardent d'un sale oeil rigolard et
 <rougeoyant,un conseil: ne traîne pas ici car si le bourreau revenait
 chambre chambre de la ferme 77
 .porte,saferme,n
 pressoir pressoir 78
 .porte,cour,o .clporte,orangerie,n
 \$squelette
 &Un squelette tintinabulant de tous ses os s'avance vers toi, menaçant
 orangerie orangerie 79
 .jeu,cave,o .porte,pressoir,s .porte,jaferme,n
 *bouton
 cave cave à vin 80
 .porte,orangerie,e
 *barrique
 grporte grande porte 81
 .clherse,cour,n .popierre,caroue,s
 cour cour 82
 .herse,grporte,s .chemin,tilleul,no
 .porte,toentree,so .chemin,courferme,n
 .clporte,pressoir,e
 *auge,buche
 tilleul tilleul 83
 .chemin,cour,s .popierre,bosquet,o .pont,grjardin,n
 .chemin,embarcadère,no .chemin,courferme,e
 \$dinsaure
 &Le fameux dinosaure que personne n'avait jamais vu broute une
 &mandragore dans un coin. te voyant, il arrête de brouter et
 &piticlop, piticlop, il fonce sur toi.
 toentreetour d'entrée 84
 .porte,cour,n .dehors,ruisseau,ne
 \$guet
 &La patrouille du guet t'aperçoit, te course et te raccompagne
 &dehors sans ménagement et en rugissant
 dpont barque 85
 .barque,dbaraque,n .renvers,jaferme,saute
 <Tu es en barque près d'un pont de pierre
 <qui obstrue les douves
 dabrev barque 86
 .barque,dcoinblan,s .barque,dcoinlion,n
 .debarc,pejardin,o .renvers,abreuvoir,saute
 <Tu es en barque près de l'abreuvoir
 dembarc barque 87
 .barque,dcoinblan,e .barque,dponpierre,o
 .debarc,embarcadère,s .renvers,courferme,saute
 <Tu es mené en bateau (par moi !) près du pont
 dbardot barque 88
 .barque,dcoinlion,e .barque,dpelevis,s .renvers,bardot,saute

<Tu vogues tranquillement au large de l'écluse
 \$hydre
 &L'eau bouillonne sous le frêle esquif, des formes sombres et
 &hideuses apparaissent à la surface de l'eau. ah malediction
 &c'est l'hydre à trois têtes qui cherche à renverser la barque
 dpelevis barque 89
 .panier,npanier,h .barque,dbardot,n .barque,druisseau,o
 .barque,dponpierre,s .debarc,grjardin,so .renvers,nif,saute
 <tu navigue près du pont levis, le long de la tour
 <il y a de la lumière à la fenêtre du haut de la tour
 <et on entend un chant mélodieux et suave
 dponpierre barque 90
 .barque,dcharmilles,s .barque,dembarc,e
 .barque,dpelevis,n .renvers,bosquet,saute
 <Tu passes sous un petit pont de pierres
 druisseau barque 91
 .barque,dcharmilles,s .barque,dpelevis,e .renvers,ruisseau,saute
 <Tu es en barque près du ruisseau qui remplit les douves
 \$pieuvre
 &Tu sens comme un tapotement amical sur ton épaule gauche
 &et en te retournant, tu serres la tentacule visqueuse gluante
 &et abominable d'une pieuvre infecte qui cherche à t'entraîner
 &sous l'eau
 metro1 metro 92
 .meporte,station4,b
 <Tu es dans une rame de métro complètement vide
 metro2 métro 93
 .meporte,station4,b
 station1 station de métro 94
 .meporte,metro1,h .couloir,station3,e
 station2 station de métro 95
 .meporte,metro1,h .couloir,coul1,s .couloir,coul2,so
 <La station est déserte
 <Toutes les plaques semblent avoir été enlevées
 station3 station de métro 96
 .meporte,metro2,h
 station4 station de métro 97
 .couloir,coul3,s .chemin,carrefour,h
 coul3 couloir 98
 .couloir,station4,no .escalier,rue1,h
 coul1 couloir 99
 .couloir,station3,ne .couloir,coul1,s .couloir,station2,n
 coul2 couloir 100
 .couloir,coul1,e .escalier,rue1,h
 ntaxi taxi 101
 .porte,boudoir,b
 ruel rue déserte 102
 .taxi,ntaxi,h .escalier,coul1,b.chemin,lab9,n

.rue,rue2,s .rue,magazin,o .escalier,impots,h
 rue2 rue déserte 103
 .taxi,ntaxi,h .rue,rue1,n .rue,atelier,s
 quatredim quatrieme dimension 104
 .parallele,dual,suivre
 *parallèle
 <Deux parallèles discutent dans le coin gauche en s'éloignant lentement
 dual espace dual 105
 .jeu,fractal,intègre
 *parallèle
 <qu'est ce qu'on peut bien faire dans un espace dual ?
 lab1 labyrinthe 106
 .porte,planque,o .porte,lab2,no
 .passage,labb,s .passage,labb,so
 lab2 labyrinthe 107
 .passage,lab3,n .passage,lab1,s
 lab3 labyrinthe 108
 .passage,lab2,e .passage,lab4,h .passage,lab5,s
 lab4 labyrinthe 109
 .passage,lab3,b .passage,lab6,ne .passage,lab8,e
 \$momie
 &Une momie éternue dans sa niche et ouvre de grands yeux glauques
 &à ton approche. elle se redresse soudain et cherche à
 &t'enrouler avec ses bandelettes
 lab5 labyrinthe 110
 .passage,lab6,s .passage,lab3,n .passage,lab4,o
 lab6 labyrinthe 111
 .passage,lab4,ne .passage,lab8,no
 .passage,lab7,o .passage,lab5,e
 lab7 labyrinthe 112
 .passage,lab8,e .passage,lab6,ne
 lab8 labyrinthe 113
 .passage,lab8,e .passage,lab8,ne .passage,lab9,n
 .passage,lab7,s .passage,lab6,so
 lab9 labyrinthe 114
 .passage,lab8,s .couloir,rue1,n
 laba labyrinthe 115
 .passage,lab5,se .passage,labb,ne
 labb labyrinthe 116
 .passage,lab1,e .passage,lab4,ne .passage,lab5,s
 atelier atelier du mouleur 117
 .porte,rue2,e
 \$mouleur
 &le mouleur sait mouler et faire des tirages de tout et n'importe quoi
 impots centre des impots 118
 .escalier,rue1,b
 \$bureaucrate
 &le bureaucrate demande le questionnaire

magazin magasin 119
 .porte,rue1,s
 *photocopieuse

sallemiroir salle du miroir 120
 .miroir,quatredim,transverse
 .porte,chtour,n .couloir,station1,e
 *miroir
 <les murs de la pièce resplendissent de mille reflets

dbaraque barque 121
 .barque,dpont,s .barque,dcoinblan,n .renvers,jaferme,saute
 .debarc,jaferme,o
 <il y a une baraque qui abrite un débarcadère

dcoinblan barque 122
 .barque,dbaraque,s .barque,dabrev,n .renvers,jaferme,saute
 .barque,dembarc,o
 <ta barque est au coin de la ferme

dcoinlion barque 123
 .barque,dabrev,s.barque,dbardot,o .renvers,nif,saute
 <ta barque est au coin des lions

dcharmilles barque 124
 .barque,druisseau,e .barque,dponpierre,n .renvers,ruisseau,saute
 <la barque longe le chemin des charmilles

charmilles charmilles 125
 .chemin,ruisseau,o .chemin,caroue,e
 <ce chemin longe les douves vers le potager

tourferme tour de la ferme 126
 .passage,jaferme,e

puits 127
 .porte,grentree,n.chemin,grjardin,e

fractal espace des fractales 128
 .escalier,boudoir,b .escalier,nirvana,h
 <ici on baigne dans mandelbrot, ifs, lsystem, attracteur, newton

lax lax 129

1.5.6 La description des photos du répertoire images_quizz et les informations généalogiques de liste_infos.txt

Ce fichier donne la liste des photos du répertoire images_quizz décrivant un personnage (ou un lieu), et une liste d'informations le concernant. S'il y a plusieurs photos, elle sont nommées prénom nom.jpg, prénom nom (2).jpg, prénom nom (3).jpg, ...

S'il y a plus de lignes d'informations que de photos, on utilise la notation prénom nom (x)

Quelques lignes extraites de ce fichier :

```
mosnay\sébastien goguet de boishéraud (2) sébastien goguet de boishéraud est sculpteur,
propriétaire de la courbejollière puis de port joli
mosnay\sébastien goguet de boishéraud sébastien goguet de boishéraud est fils d'antoine gogiet
de boishéraud et marie antoinette deslandes de bagneux
mosnay\jean goguet de boishéraud (x) jean goguet de boishéraud a émigré en allemagne
pendant la révolution, et son frère à jerzey, la mère et les soeurs ont suivi l'armée catholique
mosnay\louis goguet de boishéraud (x) louis goguet de boishéraud était curé de la renaudière,
propriétaire de la courbejollière
mosnay\jean goguet de boishéraud (x) Jean Goguet de Boishéraud a émigré en Allemagne
puis en Russie en 1793
mosnay\louis jean goguet de boishéraud (x) Louis Jean Goguet de Boishéraud a émigré à Jerzey en
1793
mosnay\alphonse goguet de boishéraud (x) Alphonse Goguet de Boishéraud hérite de la
Guérivière de son frère curé
mosnay\julienne goguet de boishéraud (x) Julienne Goguet de Boishéraud a suivi l'armée de
Vendée et a écrit ses mémoires
```

Recompilation

Pour ceux intéressés par les fichiers sources de pmais (courbe et rossulien), ne pas modifier les fichiers textes associés. Seuls liste_infos.txt, journal.txt et sphinx.txt sont modifiables sans recompilation.

Ne pas rajouter de nouvelles lignes dans les autres fichiers .txt, sinon il est nécessaire d'exécuter cmais.exe qui recalcule consmais.pas puis il faut recompiler courbe (ou rossulien) en y ayant recopié ce consmais.pas.

Pour cette recompilation il faut installer delphi (chez embarcadero, version gratuite), charger les projets x.dproj puis voir les unités.pas

on peut rajouter des jpg dans images_quizz

dans images, il faut nommer les images du numéro du lieu où on veut les voir apparaitre, ex 30.jpg, 30_1.jpg ou 30_2.jpg

Acte II :

SERIOSO

if you = serioso or no serioso serres les fesses, ca va arracher

Près du palais du grand syntaxiqueur.
deux courtisans, le grand syntaxiqueur, des chambellans.

Les chambellans portent respectueusement le grand
syntaxiqueur vers les jardins.

Le 1er courtisan (très embêté et se demandant ou je veux en
venir) :

- Comment pensez-vous que tout cela se termine ?

Le second courtisan (pas vicelard mais voulant dévoiler le fond
de sa pensée sans s'engager tout en se plaçant) :

- sale histoire

Moi (à part) :

- comment ? toi, je vais pas te louper.

Analyseur de thesaurus

1.1 Relations du réseau sémantique

Pour cette première version de dictionnaire, on n'utilise pas d'attributs syntaxiques, on ne définit que des noms communs ou des noms propres, et on en définit des attributs, formant le champ sémantique, comme un ensemble de concepts reliés entre eux par des liens qui sont de trois types :

- Lien de propriété

on notera ce lien : propriété = valeur (ou liste de valeurs séparées par des virgules)
oiseau
déplacement=vol,marche

- Lien de la partie au tout

La France est constituée de régions
(et donc la Bretagne est une région de France)
on notera ce lien ::= liste des constituants
France
région::=Bretagne,limousin,corse,...

- Lien d'ensemble à élément de l'ensemble

Les animaux se classent en plusieurs ordres
(et donc un oiseau est un animal et en hérite les propriétés)
on notera ce lien := liste des éléments de l'ensemble
animal
ordre := mammifère,oiseau

Ce dernier lien est fondamental, il permet de faire hériter à un concept les propriétés et les parties d'un autre concept. Par exemple un oiseau hérite des propriétés d'un animal.

1.2 Fichier de définition

```
france
  habitant=français
  capitale=paris
  région::=
    bretagne
      département::=
        finistère
          chef lieu=quimper
          arrondissement::=
            brest,morlaix,chateaulin
        morbihan
          chef lieu=vannes
          arrondissement::=lorient,pontivy
        côtes du nord
          chef lieu=saint brieuc
          arrondissement::=dinan,guingamp,lanion
    limousin
      instrument=cabrette,vielle
      danse=bourrée
      produit régional=gâteau creusois,charcuterie
      département::=
        corréze
          chef lieu=tulle
          arrondissement::=
            ussel,brive la gaillarde
        creuse
          chef lieu=guéret
          arrondissement::=aubusson
  province::=
    bretagne
      département::=
        finistère
        côtes du nord
        morbihan
        ile et vilaine
        loire atlantique
    aquitaine
    languedoc
bretagne
  habitant=breton
  langue régionale=breton,gallo
  instrument=bombarde,biniou,vielle
  danse=an dro,laridé,gavotte,avant deux,plinn,fisel
  chant=kan a diskan
  produit régional=crêpe,gâteau breton,cidre,chouchen
boisson
  boisson alcoolisée:=
    cidre
```

```

                                base=pomme
        chouchen
                                base=cidre,miel
                                couleur=jaune
cheval blanc d'henri IV
        couleur=blanc
bataille de marignan
        date=1515

animal
    embranchement:=
        mollusque
        vertébré
            classe:=
                batracien
                reptile
                oiseau
                    déplacement=vol,marche
                    espèce:=
                        autruche
                            déplacement=marche
                            faucon
                                mammifère
                                    super ordre:=
                                        carnivore
                                            famille:=
                                                canidé
                                                félidé
                                                    ordre:=
                                                        cétacé
                                                            espèce:=
                                                                baleine

canidé
    griffe=non_rétractile
    molaire=broyeuse
    genre:=
        canis
            espèce:=
                chien
                loup

félidé
    griffe=rétractile
    genre:=
        felis
            espèce:=
                chat

```

Ce fichier entraîne la définition du mot France et de tous les noms de départements, de chefs lieux, d'arrondissements, de produits régionaux, d'animaux, ... Le décalage des lignes les unes par rapport aux autres est significatif et correspond justement à ces déclarations implicites. les attributs sont notés "=", les mots définissant les éléments d'un ensemble, qui entraînent la transmission des attributs supérieurs, sont notés ":@" et les mots définissant les constituants d'un système sont notés "::="

1.3 Exemples d'interrogation de la base de données

On se propose d'analyser un fichier de ce type et de répondre à des questions concernant le champ sémantique qu'il décrit :

- ? qu'est ce que 1515
date de la bataille de marignan
- ? quelle est la couleur du cheval_blanc_d_henri_IV
blanc
- ? quels sont les instruments
cabrette, vielle, bombarde, biniou, vielle
- ? quels sont les instruments de bretagne
bombarde, biniou, vielle
- ? décrire le chouchen
** produit régional de bretagne de région de france*
** boisson alcoolisée de boisson*
base = cidre, miel
couleur = jaune
- ? attribut du chouchen
couleur, base
- ? quelle est la base du chouchen
cidre, miel
- ? quel est le produit_régional dont la base est le cidre
chouchen
- ? quel est la boisson dont la base est la pomme ou la couleur jaune
cidre, chouchen
- ? quels sont les départements dont une danse est la gavotte
ou dont un produit_régional est le chouchen
finistère, ...
- ? lister les animaux dont les griffes sont rétractiles
félidé, felis, chat
- ? lister les oiseaux dont le déplacement n'est pas le vol
autruche

On notera que l'analyse des mots composés est rudimentaire, ils sont notés avec « _ »

1.4 syntaxe de la question

La question comporte trois parties :

- **la commande**, qui peut avoir 5 significations :

attribut	on recherche le nom des attributs du mot
valeur	on recherche le nom des attributs et leurs valeurs
tout	on recherche toutes les informations sur le mot :
	mots définissants
	attributs et valeurs
	mots induits par la classification
quel	on recherche les mots ayant telle valeur d'attribut
lister	on recherche la liste de tous les mots induits par
	la classification

- **un nom** définissant l'entité recherchée
(... départements de bretagne ...)

- **une proposition relative** comportant des couples d'attributs et de valeurs amenés par la conjonction "dont" ou le verbe "avoir" et enchainés par les conjonctions "et" et "ou".
(dont la base est le miel et qui a la couleur jaune)

La négation peut porter sur le verbe : « n'a pas » ou sur la valeur « non jaune »

les deux parties de la question sont séparées par les conjonctions "dont", "ayant" et "à". la deuxième partie, filtrante, peut être absente.

1.5 les listes

La représentation interne des données est construite autour d'une gestion de listes chaînées entre elles et constituant le réseau sémantique. Chaque mot est représenté par ses caractères et sa liste de définition, et est référencé dans les autres définitions par son numéro.

Chaque élément d'une liste contient

- une valeur d'élément
- une liste d'attributs ou de mots classificateurs
- un numéro d'élément suivant de la liste ou 0
- un numéro d'élément englobant.

La liste de définition d'un mot est une liste de valeurs ou une liste d'attributs ayant chacun une valeur. A tous ces attributs et valeurs est associé aussi une liste de définition, dans laquelle le lien avec le mot définissant est noté dans un élément de valeur 0.

1.6 fonctionnement de la réponse

Pour répondre à une question concernant un mot, il suffit de parcourir sa liste de définition et d'éditer le ou les éléments de cette liste qui correspondent. A chaque élément, on construit la liste des mots englobant jusqu'au niveau N et si tous les mots de la question sont dans cette liste, l'élément répond à la question. Cela permet de répondre correctement aux questions :

? quels sont les départements de bretagne
et ? quels sont les départements du limousin

La polysémie n'est pas gérée, ainsi pour le Maine, il faudrait éliminer l'état du Maine aux USA, si on y a défini une rubrique "chef_lieu" :

 ? quel est le chef_lieu du maine en france
ou ? quel est le chel_lieu du département du maine

Cette dernière formulation levant l'ambiguïté dans la mesure où le seul Maine à avoir un définissant "département" est le nôtre, le leur ayant plutôt (et même surement) un définissant "état". De toute façon, en cas d'ambiguïté le programme donne toutes les réponses possibles. Et puis d'abord, il n'y a pas de département du Maine en France, nah ! Lorsque l'on a trouvé un mot correspondant à la définition, on cherche dans ces attributs si les propositions en "dont" sont satisfaites. Si un des attributs est classificateur, on essaye avec les mots induits. Cela permet de répondre aux questions :

 ? quel est l'animal dont les griffes sont rétractiles
 ? quel animal a des griffes rétractiles

le cas de l'autruche est délicat :

 ? quel est l'oiseau dont le déplacement n'est pas le vol

car on trouve d'abord l'attribut contraire pour les oiseaux, il faut passer outre et chercher quand même dans les oiseaux, c'est à dire qu'un mot peut ne pas répondre à une question mais l'un de ses mots induits peut y répondre.

grammaire sous forme de bakus

Où l'on voit se déchaîner ce bon dieu de Bakus,
pour la plus grande gloire de Syntaxikos et le salut des langages.

1.1 la forme de bakus

La grammaire se présente sous la forme d'une suite non ordonnée de règles. Chaque règle définit un symbole appelé "non terminal" comme étant composé soit d'une suite de symboles (terminaux ou non), soit d'un choix entre plusieurs suite de symboles. Un symbole terminal est un mot du langage.

c'est pas clair ? alors zieutez moi cet exemple :

```
<phrase>      := < sujet > < groupe verbal >
<sujet>       := < groupe nominal >
<groupe nominal>:= < article > < nom >    /    < nom propre >
<article>     := "le" / "la"
<nom propre>  := "pierre" / "cathy"
```

le choix est marqué par le caractère '/'

les espaces et les passages à la ligne sont sans significations.

"le", "la", "pierre" et "cathy" sont des mots du langage ce sont les symboles terminaux, les autres symboles sont les non terminaux.

Théoriquement, ça a du s'éclaircir, ou alors vous vous êtes trompé de bouquin, vous avez pris la version javanaise : check with your preferred libraire.

on peut utiliser les parenthèses :

```
<groupe verbal>      := < verbe > < complement >
<complement> :=
    ( < groupe nominal > < conjonction > < groupe nominal > ) / < groupe nominal >
```

certaines suites peuvent être facultatives :

```
<sujet> := < groupe nominal > [ "et" <sujet> ]
```

La plupart des langages ont la possibilité d'utiliser des nombres ou des mots terminaux dont la liste n'est pas figée : il est impossible par exemple de donner la liste de tous les noms propres.

la grammaire prévoit ce cas et 3 fonctions sont prévues pour permettre cela : +get identifier+, +get nombre+ et +get string+ on écrira donc :

`<nom propre> := +get identifier+`

Certains langages utilisent la notion de mots réservés, c'est à dire qu'ils ne peuvent pas être utilisé pour un identifieur.

'mot' est un mot clé

"mot" n'en est pas un

c'est le cas en pascal pour tous les mots du genre : 'for', 'begin', 'end', ... mais ce n'est sans doute pas le cas en français : "la" peut être article mais peut aussi être un nom (la note « la ») et même aussi un nom propre.

1.2 analyse de la grammaire

l'analyse de la grammaire consiste à constituer un dictionnaire des symboles non terminaux et d'avoir pour chacun une représentation sous forme d'arbre et/ou chaque nœud de l'arbre est une référence à un autre symbole non terminal, les feuilles sont les symboles terminaux.

un nœud à la structure suivante :

- nom du symbole
- type du symbole (terminal, règle, fonction)
- obligatoire ou facultatif
- pointeur sur la suite (et)
- pointeur sur l'autre choix (ou)

On s'arrange pour qu'un nœud n'aie pas un champ "et" et un champ "ou", et pour cela on ajoute automatiquement à la grammaire quelques règles nommés :

`<groupe nominal> := <article> <nom> / <pronom personnel> / <nom propre>`

se représente comme la règle :

`<groupe nominal> := (<article> <nom>)
 / <pronom personnel>
 / <nom propre>`

soit comme les deux règles :

`<groupe nominal> := <articienom> / <pronom personnel> / <nom propre>
<articienom> := <article> <nom>`

1.3 analyse d'une phrase

L'analyse est du genre top-down, elle part de la règle définissant la phrase dans la grammaire et cherche une décomposition de `<phrase>` qui s'unifie à la phrase donnée. En cas d'échec, l'analyseur procède à un retour en arrière jusqu'au point de choix ou de facultatif précédent.

1.3.1 analyse par descente récursive

l'arbre résultat de l'analyse de la grammaire est transformé en un programme pascal. Et chaque règle est transformée en une procédure :

```
function article_nom : boolean;
begin
  if article
  and nom
  then   article_nom := true
  else   article_nom := false;
  end;
```

En fait, pour assurer en cas d'échec le retour arrière dans le fichier à analyser, on utilise les procédures "note_position" et "retour_position"

```
function article_nom : boolean;
begin
  note_position;
  if article
  and nom
  then   article_nom := true
  else   begin
          article_nom := false;
          retour_position;
        end;
  end;
```

```
function groupe_nominal : boolean;
begin
  note_position;
  if article_nom
  or pronom_personnel
  or nom_propre
  then   groupe_nominal := true
  else   begin
          groupe_nominal := false;
          retour_position;
        end;
  end;
```

Le test des terminaux se fait en utilisant la procédure "cmp" située dans l'unité "ucompil" :

```
function article : boolean;
begin
  note_position;
  if cmp('le')
  or cmp('la')
  or cmp('les')
  then   article := true
  else   begin
          article := false;
          retour_position;
        end;
  end;
```

La saisie d'un mot quelconque se fait de la même manière en utilisant la fonction "get_identifier".

```
function nom_propre : boolean ;  
  begin  
    if get_identifier  
    then   nom_propre := true  
    else   nom_propre := false;  
  end;
```

Le cas des éléments facultatifs se traite avec l'aide d'une procédure suffixée "_fac".

< sujet > := < groupe nominal > ["et" < sujet >]

```
function p_1 : boolean;  
  begin  
    if cmp('et')  
    and sujet  
    then   p_1 := true  
    else   p_1 := false;  
  end;
```

```
function p_1_fac : boolean;  
  begin  
    if p_1 then ;  
    p_1_fac := true;  
  end;
```

```
function sujet : boolean;  
  begin  
    if groupe nominal  
    and p_1_fac  
    then   sujet := true  
    else   sujet := false;  
  end;
```

1.11 Application à la grammaire du français

```
<phrase> := [<introd>] <centrale> <finale>  
<discours> := <phrase> <discours>  
<introd> := <interjection> / <jonction de cause>  
<finale> := "." / "?" / ";" / "!"  
<centrale> := <affirmation> / <sans sujet> / <interrogation> / <ordre>  
<affirmation> := <assertion> / <inversion> / <extraction> / <impersonnel>  
<assertion> := <sujet> <groupe verbal>  
<sujet> := <nom propre>  
           / <pronom personnel>  
           / <pronom relatif> <relative>  
           / <groupe nominal>  
<complement> := <nom propre>
```

/ <pronom me>
 / <groupe nominal>
 <groupe nominal> := <determinant> <fin de groupe>
 / <quantitatif> [<determinant>] <fin de groupe>
 <fin de groupe> := [<adjectif>] <nom> [<adjectif>] [<relative>]
 <relative> := <de> <relative de>
 / "en" <relative en>
 / "qui" <relative qui>
 / <que> <relative que>
 / "dont" <relative dont>
 / <relative participe passe>
 / <relative participe present>
 / <relative pronom>
 / <parenthese>
 <parenthese> := "(" <ajout parenthese> ")"
 "," <ajout parenthese> ","
 <ajout parenthese> := <assertion>
 / <pronom relatif> <relative>
 / <groupe nominal>
 / <ajout>
 / <relative participe passe>
 / <relative participe present>
 / "comme" <groupe nominal>
 / "bien que" <assertion>
 / "mieux que" <groupe nominal>
 <relative en> := <matiere>
 <relative de> := <nom propre>
 / <attribut> <specification d attribut>
 / <expression de valeur>
 / <article> <attribut> <expr comparative> <element de comparaison>
 / <groupe nominal avec un nom>
 <relative pronom> := <conjonction> <pronom relatif> <sujet> <complement>
 <specification d attribut> := <adjectif d attribut>
 / <expr comparative> <element de comparaison>
 <expr comparative> := [<modifiant de comparatif>] <comparatif>
 <modifiant de comparatif> := "au moins" / "au plus" / "au maximum"
 / "au minimum"
 / "bien" / "de beaucoup" / "de peu" / "un peu"
 / "environ" / "a peu pres" / "approximativement"
 / "non"
 <comparatif> := "egal a" / "identique a"
 / "superieur a" / "inferieur a"
 / "de"
 <element de comparaison> := <element relatif> <groupe nominal>
 / <expression de valeur>
 <element relatif> := <pronom relatif> <de>
 / <article> <attribut> <de>
 <expression de valeur> :=
 [<conjonction de valeur absolue>] <nombre avec unite de mesure>
 / <conjonction de valeur relative> [<element relatif>] <groupe nominal>

<conjonction de valeur absolue> :=
 "plus de"
 / "moins de"
 / "au plus"
 / "au moins"
 / "au minimum"
 / "au maximum"
 <conjonction de valeur relative> :=
 "plus que"
 / "moins que"
 / "autant que"
 / "le meme que"
 / "le double de"
 / "le triple de"
 / "la moitie"
 / "le quart"
 <relative qui> := <verbe attribut> <expression de valeur>
 / <verbe avoir> <article> <attribut>
 <comparatif> <element de comparaison>
 / <verbe avoir> <conjonction de valeur relative> <de> <attribut>
 <que> <groupe nominal>
 / <groupe verbal>
 <verbe avoir> := <avoir>
 / <ne> <avoir> <pas>
 <avoir> := "a" / "ont"
 <relative que> :=
 <sujet> <groupe verbal direct>
 / <groupe verbal direct> <sujet>
 <relative dont> := <article> <attribut> <etre vaut>
 <specification d attribut>
 / <assertion>
 <relative participe passe> := <participe passe> [<complement passif>]
 <relative participe present> := <participe present> [<complement direct>]
 / <verbe d attribut participe present> <expression de valeur>
 <groupe verbal> :=
 <groupe verbal direct>
 / <groupe verbal passif>
 / <groupe verbal pronominal>
 / <pronom me> <fin pronom me>
 / <pronom les><verbe direct> <complement direct>
 / <pronom en> <verbe direct> <complement direct>
 <fin pronom me> := <pronom les><verbe direct>
 / <verbe direct> <complement direct>
 <groupe verbal direct> := <verbe direct> <complement direct>
 / <ne> <verbe flechi> <pas> [<complement direct>]
 / <ne> <pronom me> <pronom les> <verbe flechi> <pas>
 / <ne> <pronom les> <verbe flechi> <pas> <complement direct>
 / <ne> <pronom les> <verbe flechi> <pas> <complement direct>
 <complement direct> := <complement>
 / <complement><conjonction><complement>

/ <conjonction><complement><complement>
 / <de> <verbe infinitif> <complement direct>
 / <que> <assertion>
 <verbe direct> := <verbe flechi>
 / <ne> <verbe flechi> <pas>
 / <verbe operateur> <verbe infinitif>
 / <ne> <verbe operateur flechi> <pas> <verbe infinitif>
 <groupe verbal passif> := <verbe passif> [<complement passif>]
 <complement passif> := "par" <complement>
 / "par" <complement> <conjonction> <complement>
 / <conjonction> <complement> "par" <complement>
 <verbe passif> := <etre> <participe passe>
 / <ne> <etre> <pas> <participe passe>
 <ne> := "ne" / "n"
 <pas> := "pas" / "aucunement" / "nullement" / "jamais"
 <verbe operateur> := "faire"
 / <verbe operateur de mode>
 / <verbe operateur de temps>
 / <verbe operateur de dire>
 <ajout> := <ajout de temps>
 / <ajout de duree>
 / <ajout de lieu>
 / "en" <participe present> <complement direct>
 / <adverbe>
 <ajout de temps> := <adverbe de temps>
 / <specification de date>
 / <date>
 <date> :=
 <jour> ["prochain" / "en 8"]
 / "le" <jour> <nombre> <mois> <annee>
 / "a" <nombre> "heure" <nombre> "minute" <nombre> "seconde" <nombre>
 "centieme"
 / "en" <mois> <annee> [("avant" / "apres") ("jesus christ" / "jc")]
 / "au" <nombrieme> [<unite de temps>]
 / <article> <unite de temps> <adjectif de temps>
 / <adverbe de temps> <article> <periode historique>
 <adjectif de temps> := "passe" / "precedent" / "suivant" / "dernier"
 / "avant" / "apres"
 <ajout de duree> := <conjonction de duree> <nombre avec unite de temps>
 / <nombre avec unite de temps> ["durant" / "de suite"]
 <unite de temps> :=
 "jour" / "mois" / "semaine" / "annee" / "siecle" / "millenaire"
 / <saison>
 <periode historique> := "moyen age" / "revolution" / "guerre" <relative de>
 / <unite de temps>
 <ajout de lieu> := [<conjonction de lieu>] <lieu>
 <ordre> := [<groupe nominal> ", "] <ordre effectif>
 / <ordre> [", " <groupe nominal>]

<ordre effectif> := <groupe verbal direct a l'infinitif ou a l'imperatif>
 (<verbe infinitif> / <verbe imperatif>) [<complement direct>]
 / "ne" <pas> <verbe infinitif> [<complement direct>]
 / <ne> <verbe imperatif> <pas> [<complement direct>]

<interjection> := "ah" / "oh"
 <jonction de cause> := "donc" / "mais" / "et" / "par consequent"
 <conjonction de cause> := "à cause de" / "pour" / "puisque"
 <conjonction de lieu> := "a" / "pres de" / "loin de" / "devant" / "derriere"
 / "sur" / "sous" / "au dessus de" / "au dessous de"
 <conjonction de temps> := "quand"
 <conjonction de duree> := "pendant" / "durant"
 <specification de date> := "hier" / "demain" / "aujourd'hui"
 <adverbe de temps> := "avant" / "apres" / "plus tard" / "plus tot"

<verbe operateur de mode> := "devoir" / "pouvoir" / "vouloir" / "risquer de"
 <verbe operateur de temps>:= "commencer a" / "venir de" / "finir de"
 / "aller"
 <verbe operateur de dire> := "dire" / "menacer de" / "ordonner" / "penser"
 <nom propre> := "pierre" / "marc"
 <pronom me> := "me" / "te" / "lui" / "nous" / "vous" / "leur"
 <pronom les>:= "le" / "la" / "les" / "l" ""
 <pronom en> := "en" / "y"
 <pronom personnel>:= "je" / "j" "" / "tu" / "il" / "elle"
 / "nous" / "vous" / "ils" / "elles"
 <pronom relatif>:= "celui" / "celle" / "ceux" / "celles"
 <determinant> := <article> / <demonstratif>
 <article>:= "le" / "la" / "les" / "un" / "une" / "des" / "l" ""
 <demonstratif> := "ce" / "cet" / "cette" / "ces"
 <quantitatif>:= "de" / "un peu de" / "peu de" / "beaucoup de"

<verbe infinitif>:= "manger" / "poser" / "etre"
 <participe passe>:= "mangé" / "mangée" / "mangés" / "mangés"
 <participe present>:= <participe_present>
 / <ne> <participe_present> <pas>
 <participe_present> := "mangeant"
 <verbe imperatif>:= "mange"
 <verbe flechi>:= "mange" / "pose" / "est"
 <nom> := "chat" / "souris" / "boite" / "pomme" / "pommes"
 <adverbe>:= "joliment" / "rapidement"

<groupe nominal avec un nom>:=<groupe nominal>
 <groupe nominal avec un nom derive d'un verbe>:=<groupe nominal>
 <verbe d attribut participe present> := "pesant"
 <verbe attribut> := <verbe_attribut>
 / <ne> <verbe_attribut> <pas>
 <verbe_attribut> := "pese" / "est"
 <attribut>:= "taille" / "forme" / "poids" / "duree" / "epoque"
 <adjectif> := "petit" / "petite" / "grand" / "grande" / "beau" / "belle"

/ "grandes" / "ovale"
 <adjectif d attribut>:=<adjectif>
 <matiere> := "bois" / "fer"
 <nombre avec unite de mesure>:=<vide>
 <groupe verbal pronominal>:= <se> <groupe verbal direct>
 / <se> <pronom les><verbe direct> <complement direct>
 <se> := "se" / "s"
 <groupe verbal infinitif>:=<groupe verbal>
 <conjonction>:=<conjonction de lieu>
 / <conjonction de temps>
 / <conjonction de cause>
 <verbe operateur flechi>:=<vide>
 <etre>:=<est" / "sont"
 <etre vaut>:= <ne> <etre_vaut> <pas>
 <etre_vaut> := "est" / "sont" / "vaut"
 <jour>:=<lundi" / "mardi"
 <nombre>:=<vide>
 <mois>:=<janvier" / "fevrier"
 <annee>:=<nombre>
 <nombre avec unite de temps>:=<nombre><unite de temps>
 <saison>:=<hiver" / "ete"
 <nomabrieme> := "premier" / "deuxieme" / "troisieme"
 <lieu>:=<vide>
 <groupe verbal direct a l'infinitif ou a l'imperatif>:=<groupe verbal>
 <vide>:=<xxx"

<sans sujet>:=
 <verbe supposons> (<groupe nominal> / <que> <assertion>)
 ("soit" / "soient") <groupe nominal>
 ("voici" / "voila") <conjonction voici> <assertion extraite>
 <verbe supposons>:=<supposons"
 <assertion extraite> := <assertion>

<conjonction voici> :=
 "que"
 / "a quoi"
 / "a qui"
 / "ou"
 / "d'ou"
 / "qui"
 / "ce dont"
 / "ce que"

<interrogation>:= <oui-non> / <question sujet> / <question complement>
 <oui-non> := [<oui_non>] <oui non> [<oui_non>]
 <oui_non> := "oui ou non" / "ou non"
 <oui non>:=
 <assertion> ", n'est-ce pas"
 / [<groupe nominal>] <ne> <verbe direct> <pronom-t> <pas> <complement>
 / "est-ce que" <assertion>

/ "est-ce" < sujet > "qui" < groupe verbal >
 / "est-ce" [< prepv >] < groupe nominal > < que > < assertion extraite >

< question sujet > := < particule question > < groupe verbal >
 < particule question > := "qui"
 / "qui est-ce qui"
 / "qu'est-ce qui"
 / "qui a t'il" < conjonction > < groupe nominal >
 / < quel > < est > < groupe nominal > "qui"
 / "combien y a t'il de" < groupe nominal >
 / "y a t'il" < groupe nominal >

< question complement > :=
 < conjonction interrogative > [< sujet >] < verbe direct >
 < pronom-t > < complement >
 / < prepv > ("qui" / "quoi") [< sujet >] < verbe direct >
 < pronom-t > < complement >
 / < quel > < groupe nominal > < assertion extraite >
 / < quel > < est > < groupe nominal > (< que > / "dont") < assertion extraite >
 / < lequel > [< de > < groupe nominal >] < groupe verbal >

< pronom-t > :=
 < pronom personnel >
 / "-t-" < il ou elle >

< de > := "de" / "d'"

< que > := "que" / "qu"

< quel > := "quel" / "quelle" / "quels" / "quelles"

< lequel > := "lequel" / "laquelle" / "lesquels" / "lesquelles"

< est > := "est" / "sont"

< prepv > := "de" / "a" / "pour" / "sur" / "sous" / "avec"

< il ou elle > := "il" / "ils" / "elle" / "elles"

< conjonction interrogative > :=
 "ou" / "d'ou" / "quand" / "comment" / "pourquoi"
 / < quel > / "de combien" / "auquel" / "a combien de"

< conj c est > := "quand" / "depuis que" / "alors que" / "parceque"

< conj nom c est > := "depuis" / "a cause de"

< extraction > :=
 < c'est > < sujet > "qui" < groupe verbal >
 / < c'est > [< prepv >] < groupe nominal > < que > < sujet > < assertion extraite >
 / < c'est > [< prepv >] < groupe nominal > < que > < verbe direct > < sujet >
 / "c'est" [< de >] < groupe verbal infinitif > "qui" < groupe verbal >
 / "c'est" < de > < groupe nominal > < que > < assertion extraite >
 / "c'est" < groupe nominal > (< que > / "dont") < assertion extraite >
 / "c'est" < conj c est > < assertion > < que > < assertion >

< c'est > := "c'est" / "ce sont"

< inversion > := [< prepv >] < groupe nominal > < assertion extraite >

< impersonnel > :=
 "il" < groupe verbal >
 / "il y a" < le fait >
 / "il" < verbe operateur > "y avoir" < le fait >

/ "il a ete" <participe passe> <complement>
 / "il a ete" <participe passe> <deque complet>
 / "c'est" <conj nom c est> <groupe nominal> <que> <assertion>
 / <c'est impersonnel> <adjectif bien> <deque>
 <le fait> := ["le fait"] <que> <assertion>
 / "le fait de" <groupe verbal infinitif>
 / <groupe nominal>
 <deque> := <de> <groupe nominal>
 / <que> <assertion>
 <deque complet>:=<deque>
 / "du fait que" <assertion>
 / "de ce que" <assertion>
 <c'est impersonnel> := "c'est" / "il est" / "il semble" / "il reste"
 <adjectif bien> := "bien" / "sur" / "certain" / "probable" / "bon"
 / "souhaitable" / "normal" / "bizarre" / "faux" / "vrai"

Arrivé là, je suis tombé sur la description des accords, et des contraintes sémantiques, et j'ai découvert le langage prolog. J'ai donc abandonné cette analyse d'une grammaire sous forme de Bakus. Déçu ? non, vous allez voir que j'ai repris tout cela en prolog, avec succès.

Entracte

Des fractales en tous genres sont distribuées aux spectateurs, qui en redemandent : Julia, Mandelbrot, Newton, Von Koch, Lindenmayer, Hutchinson, Barnsley, Peano, Sierpinski sont appelés au parloir pour expliquer comment on les réalise.

Pour commencer, on part d'un segment de droite à dessiner et on le remplace par une ligne brisée, constituée donc de plusieurs segments que l'on remplace à leur tour par des lignes brisées plus petites et ainsi de suite.

On peut aussi partir d'un polygone régulier, dont on remplace chaque côté par un autre motif, Là apparaît alors un détail important : selon que l'on dessine le motif remplaçant vers l'intérieur ou vers l'extérieur du polygone, on obtient deux figures très différentes.

Pour réaliser cela, il y a deux méthodes très voisines : la tortue Logo et les systèmes de Lindenmayer et puis il y a la méthode très surprenante des systèmes d'itérateurs (IFS), qui nous permettra de nous activer le neurone par des discussions théoriques, avec définitions et démonstrations.

On verra que toutes les fractales présentées peuvent se dessiner par l'une et l'autre des trois méthodes.

La tortue Logo

La tortue graphique Logo (le langage Logo a été inventé par Seymour Papert) évolue sur un écran où elle laisse la trace de son déplacement. Elle se programme à l'aide de quelques primitives :

```
forward(longueur)
backward(longueur)
turnright(angle)
turnleft(angle)
penup                pour ne plus laisser de trace
pendown
setposition(x,y)
```

voici comment la programmer en pascal pour dessiner un motif formé d'une ligne brisée :

```
procedure motif_koch(long : real ; profond : integer ; interieur : integer) ;
var   l : real ;
      k : integer ;
begin
  if profond=1
  then forward(long)      (* arrêt de la récursion : dessin d'un segment non remplacé *)
  else begin
    l := long / 4 ;      (* en fait long=2*l + 2*l*sqrt(3)/2
                          mais la longueur doit simplement diminuer
                          que la figure remplaçante ait la longueur du segment remplacé importe peu
                          par contre elle doit être bien orientée à la fin *)
    k := profond-1 ;
    motif_koch(l, k, interieur) ;      (* dessin d'un petit segment remplaçant *)
    turnleft(60*interieur) ;           (* interieur est supposé valoir 1 ou -1 *)
                                        (* turnleft(-60) c'est turnright(60) *)

    motif_koch(l, k, interieur) ;
    turnright(120*interieur) ;
    motif_koch(l, k, interieur) ;
    turnleft(60*interieur) ;
    motif_koch(l, k, interieur) ;
  end ;
end ;
```

à la première itération le segment de longueur L est remplacée par 4 segments de longueur $L/3$
 À la deuxième, on obtient 16 segments $L/9$... À chaque itération la longueur est donc multipliée par $4/3$, ce qui signifie que, contrairement à l'intuition première, la longueur d'une courbe de Koch tend vers l'infini pour un nombre d'itérations infini.

C'est ce qui fait dire que la mesure de la longueur d'une frontière maritime dépend de la longueur de l'instrument de mesure.

les angles sont exprimés en degré :

```
angle_radian=angle_degre*pi/180
```

```
if angle_radian > 2*pi then angle_radian := angle_radian - 2*pi ;
```

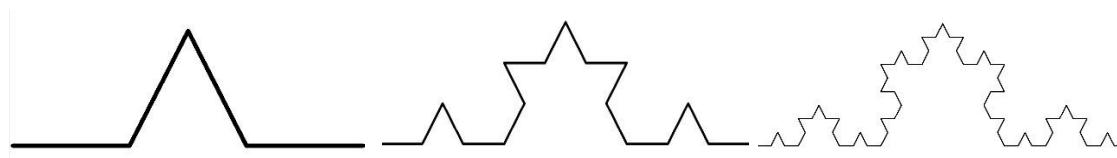
```
if angle_radian < -2*pi then angle_radian := angle_radian + 2*pi ;
```

Partant d'un segment on le remplace ainsi récursivement sur une profondeur de 2,3,4 en une ligne fractale :

```
motif_koch(100, 2, 1) ;
```

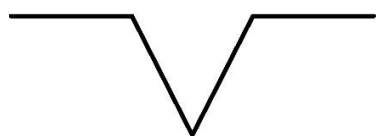
```
motif_koch(100, 3, 1) ;
```

```
motif_koch(100, 4, 1) ;
```



pour l'instant ici, l'option « intérieur » ne donne rien d'intéressant, patience.

```
motif_koch(100, 2, -1) ;
```



Pour le parcours d'un triangle (ou de toute autre figure formée de segments, on procède ainsi :

```
Procedure triangle(long :real ; profond :integer ; interieur :integer) ;
```

```
begin
```

```
  motif_koch(long, profond, interieur) ;
```

```
  turnleft(120) ;
```

```
  motif_koch(long, profond, interieur) ;
```

```
  turnleft(120) ;
```

```
  motif_koch(long, profond, interieur) ;
```

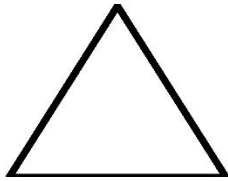
```
  turnleft(120) ;
```

```
end ;
```

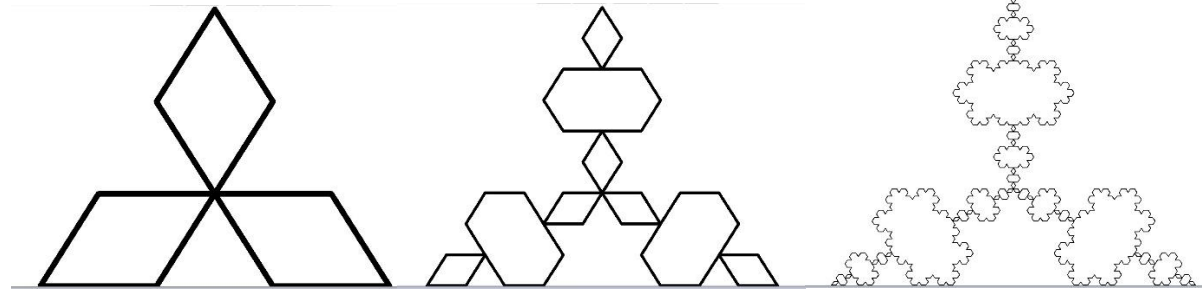
rappel : pour dessiner un polygone régulier, la valeur de l'angle entre chaque côté est :

$$360/\text{nb_cotés}$$

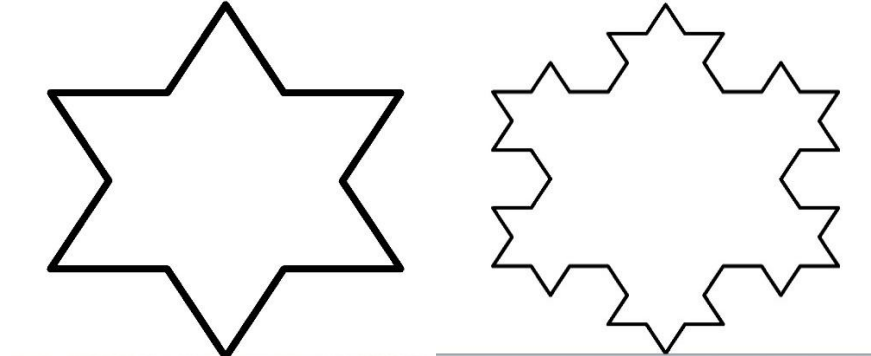
triangle(100,1,1) ; la profondeur de 1 n'est pas suffisante, on dessine les segments sans transformation.



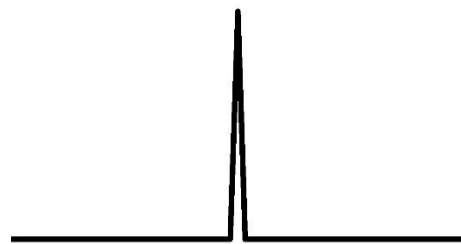
Puis, pour tracer la ligne brisée vers l'intérieur du triangle
`triangle(100, 2, 1) ; ... triangle(100,3,1) ;`



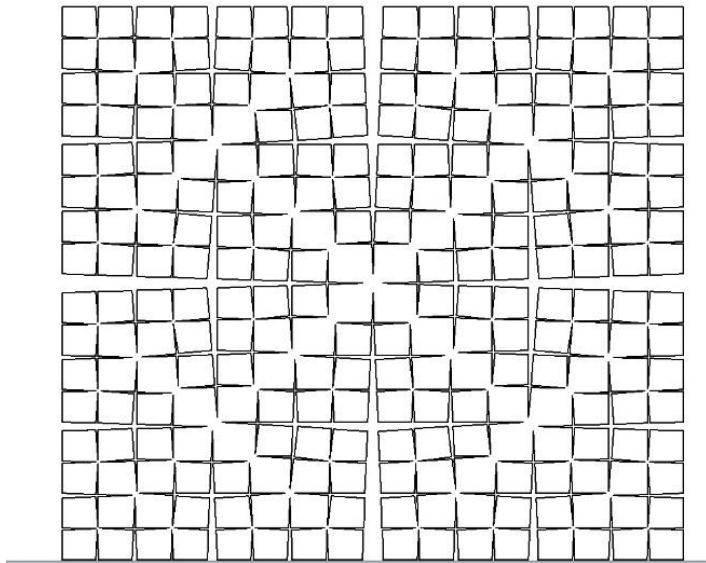
et pour la tracer vers l'extérieur
`triangle(100, 2, -1) ; ...`



on peut utiliser une brisure plus aigüe,



ce qui donne en parcourant un carré vers l'intérieur :



Grammaire de Lindenmayer

Les systèmes de Lindenmayer ont été créés pour modéliser la croissance des plantes.

On part d'une structure de départ constituée d'une ou plusieurs lettres (ou caractères spéciaux) et à chaque cycle on remplace les lettres en suivant la règle de production associée à chacune d'elles.

S'il y a plusieurs règles associées à la même lettre, on en choisit une au hasard.

Certaines des lettres sont des commandes, inspirées de celles du langage de la tortue graphique «logo»

liste des commandes constituant les règles de substitution

- + tourner à droite
- tourner à gauche
- | tourner de 180°
- # tourner à droite de 90°
- /n tourner à droite de n
- \n tourner à gauche de n
- ! inverser + et -
- [empiler les paramètres pour repartir plus tard de ce nœud de branches
-] dépiler les paramètres pour repartir depuis le nœud
- > ou >n ajouter 1 ou n à la couleur (de 0 à 256)
- < ou \$ ou <n retrancher 1 ou n à la couleur
- cn mettre n comme valeur de couleur
- @n multiplier la longueur du pas par n
- @In diviser la longueur du pas par n
- @Qn multiplier la longueur du pas par $\sqrt{n}$
- @IQn diviser la longueur du pas par $\sqrt{n}$
- : noter la position courante
- ; tracer un trait vers la position courante enregistrée -(utilisé uniquement dans le Z order)
- G avancer d'un pas sans dessiner
- F,A,B,D,l : avancer d'un pas en dessinant

En L-system, la courbe de Koch, déjà vue avec l'algorithme de la tortue logo, se définit par un angle, une structure de départ et des règles de substitution. Sur un segment :

```
angle=60  
axiom=F  
F+F--F+F
```



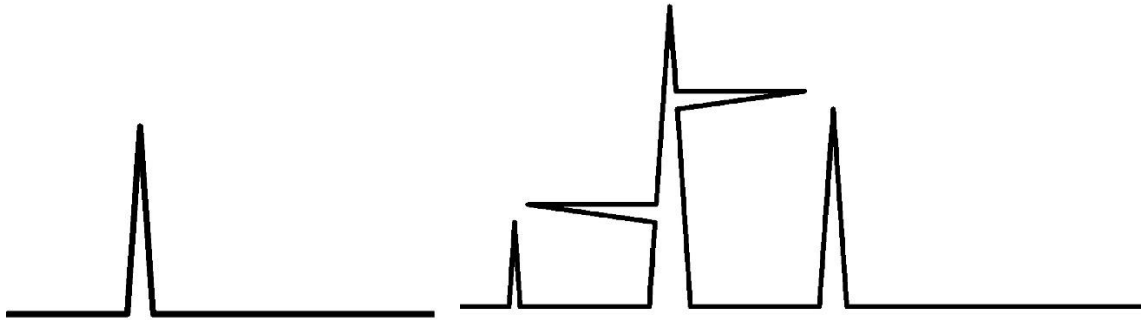
Et sur un triangle vers l'intérieur ça donne :

```
angle=60  
axiom=F--F--F--  
F-F++F-F
```

Ou, vers l'extérieur

```
angle=60  
axiom=F--F--F--  
F+F--F+F
```

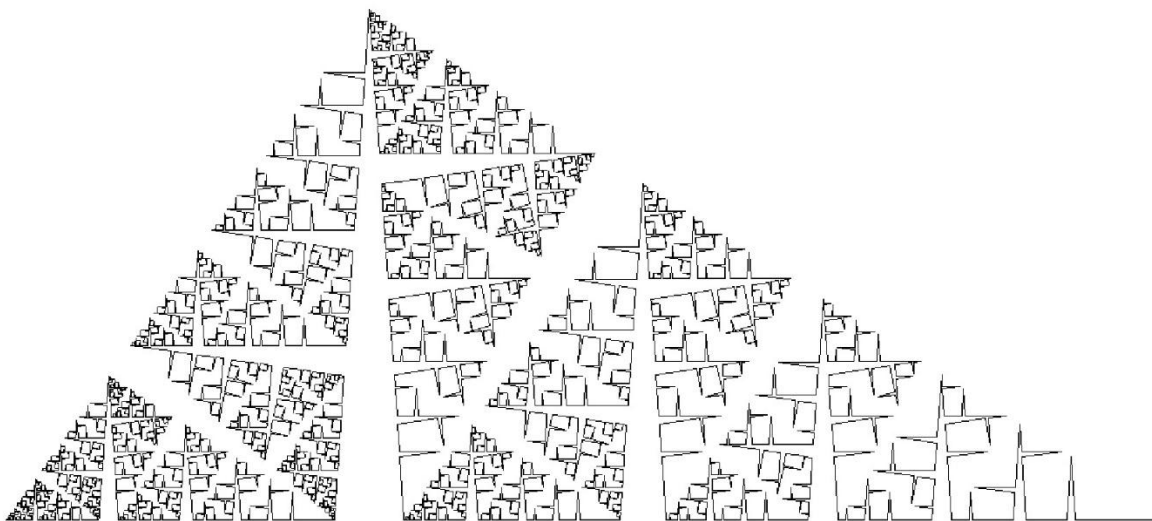
On peut rendre la brisure plus aigüe et la déplacer vers une extrémité du segment :



angle=120

axiom=D

D=@.3D\86@I.3@Q.21D/172D\86@IQ.21@.7D@I.7



A l'exception de F,A, B, et 1 toute lettre sans règle de production est supprimée à l'itération suivante, et donc ne sert à rien car n'ayant aucun effet

A chaque cycle, chaque lettre est remplacée par la règle de production lui correspondant, mais les lettres de cette règle ne sont pas expansées, et donc lorsque il y a de nombreuses règles l'effet des dernières n'est visible qu'après plusieurs cycles.

Le cadrage et le changement d'échelle se font automatiquement, par une première passe sans dessiner, servant à calculer les bornes de l'image et à la ramener dans la fenêtre par une règle de trois..

La structure de départ est souvent un segment (axiom=F), un triangle (axiom=F+F+F+) ou un carré (axiom=F+F+F+F+) (le dernier + pour s'orienter comme au début)

Parfois c'est un motif en lui-même.

Pour arrondir un angle, on double l'angle et on divise par 2 la taille pour dessiner le coin en plusieurs segments puis on remultiplie par 2 (voir le dragon)

Après avoir dessiné une branche d'un arbre, il faut revenir au nœud de la branche avec les paramètres tels qu'ils étaient au nœud, avant de continuer le tronc.

ainsi F[+F][-F]F dessine deux branches partant d'un même nœud et le tronc qui continue après le nœud de ces 2 branches

voir comme exemples le «détail» des L-systèmes proposés.

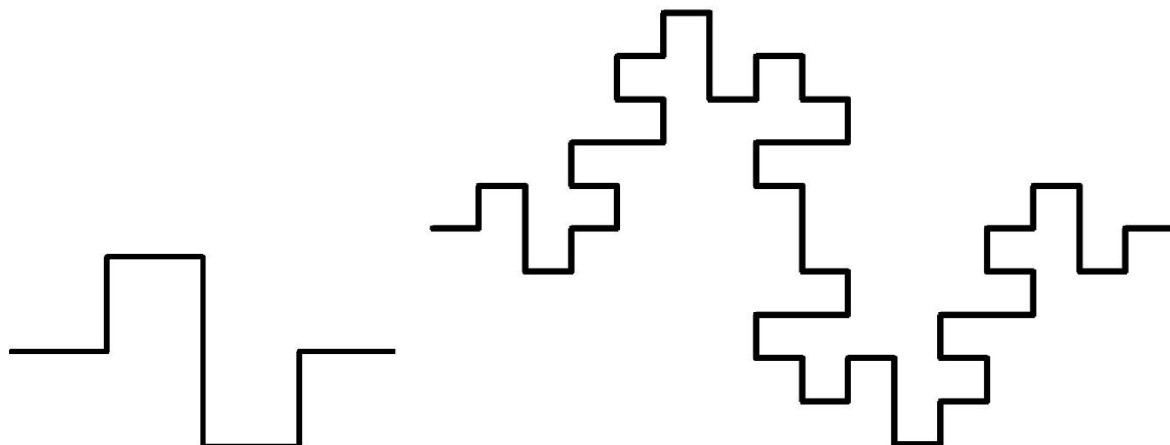
Au début, le programme est en option automatique, c'est à dire que toutes les secondes il affiche un niveau de plus, jusqu'à un maximum modifiable de 5

Cliquer sur «manuel» puis «cycle suivant» pour progresser cycle par cycle autant que désiré
l'affichage du niveau 0 correspond à l'axiome de départ

Si l'affichage est en noir et blanc, les commandes c, < et > sont ineffectives

< et > servent à passer d'une couleur à l'autre dans une liste brun, vert , bleu, indigo, violet, rouge, orange, et de nouveau brun

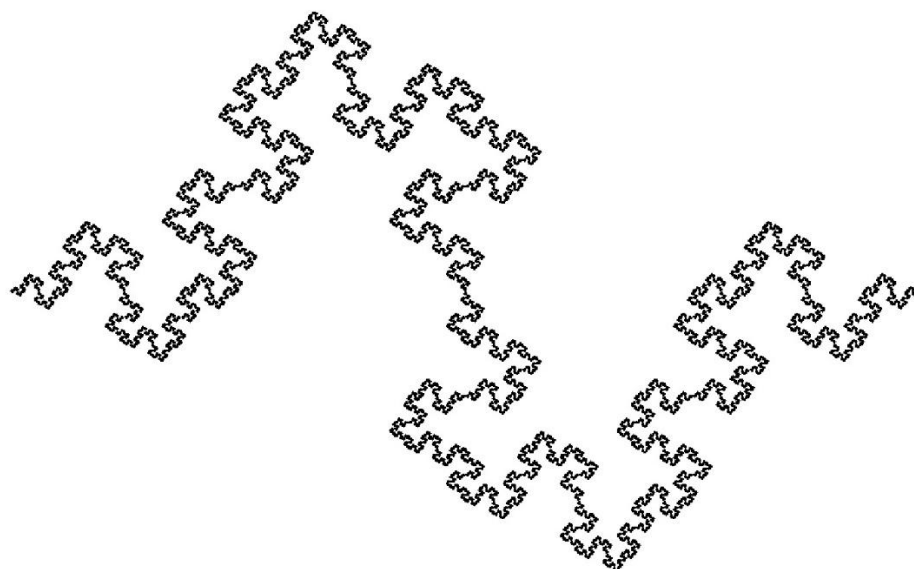
Saucisse de Minkovski

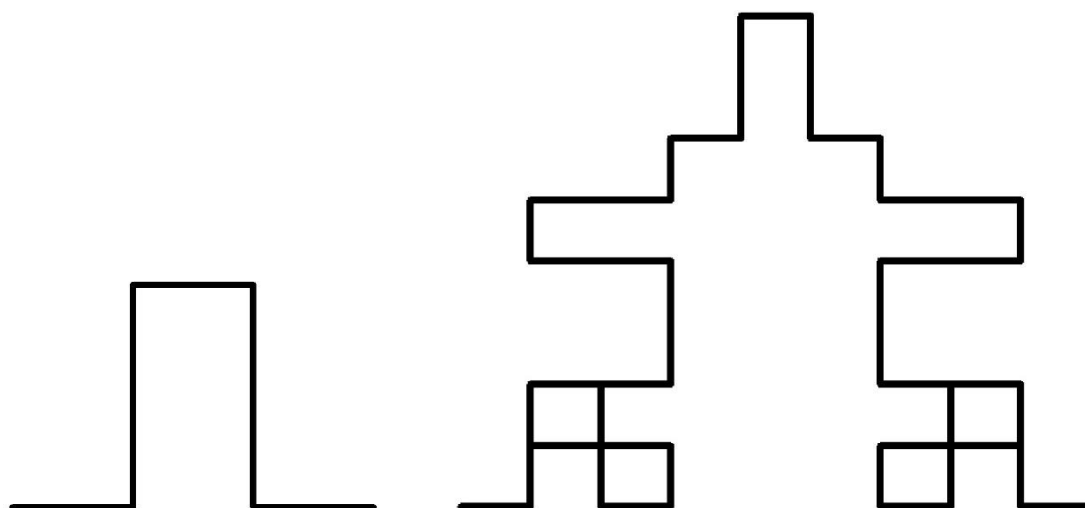


angle=90

axiom=F

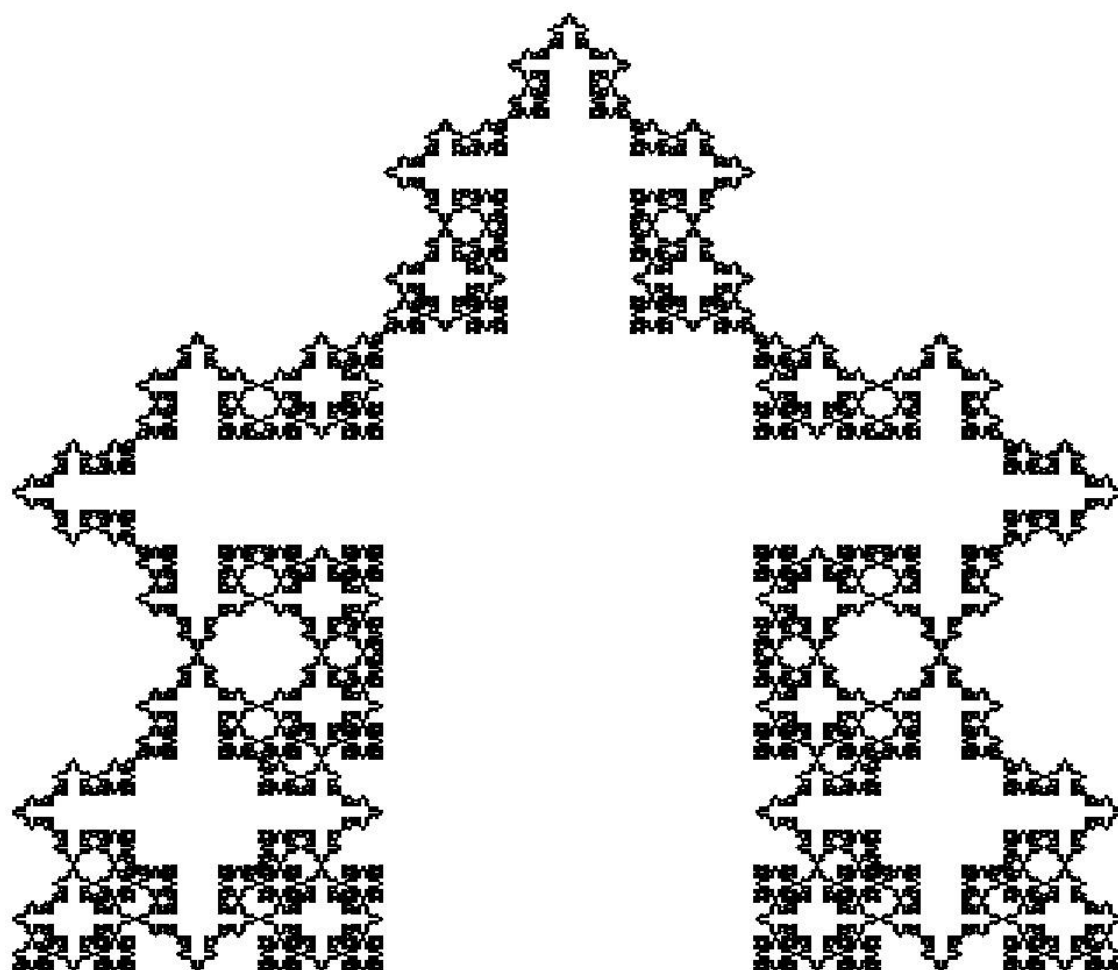
F=F-F+F+FF-F-F+F





Church

angle=90
 axiom=++F
 F=F+FF-F-FF+F

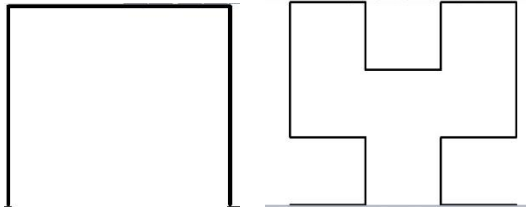


Courbes couvrant le plan

Pour cela, l'axiome est un simple segment. On découpe le plan en carreau et le motif doit parcourir les centres des n carreaux adjacents formant la base du pavage.

En général on divise un carré par un quadrillage, et le motif doit parcourir tous les carreaux. Chaque petit carré contiendra une réduction du motif.

La courbe de Hilbert



On est amené à écrire 2 règles, une X qui dessine dans le sens des aiguilles d'une montre et l'autre Y dans l'autre sens.

Lorsque l'on dessine le motif de base, il y a 4 motif à dessiner, 1 dans un sens, 2 dans l'autre sens et le dernier dans le 1er sens et il y a 3 joints à faire

on part du segment horizontal, et on oriente au début vers le haut par un $-$, on finit par un $-$ pour se remettre dans la direction du segment vu qu'on a fait 2 $+$

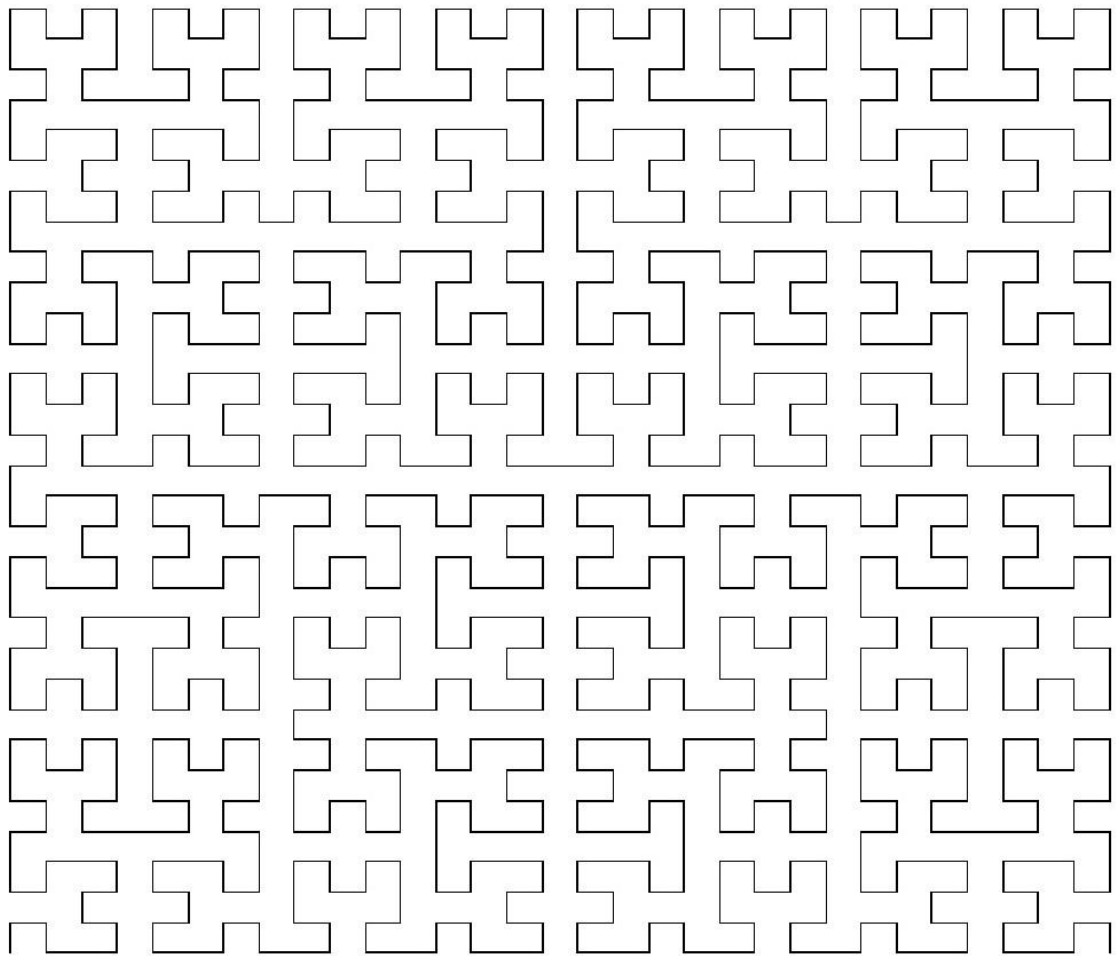
angle=90

axiom=X

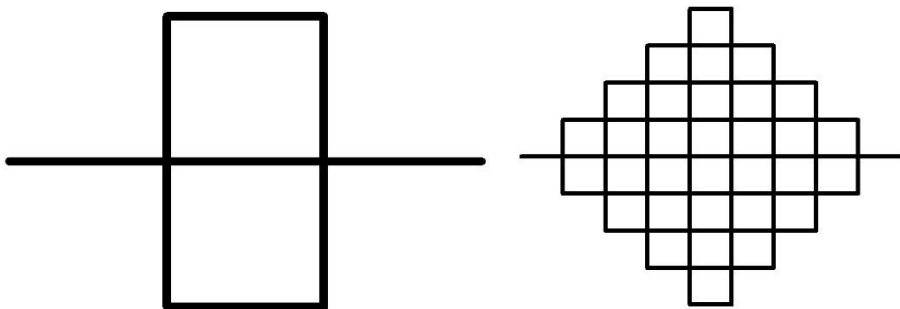
X=-YF+XFX+FY-

Y=+XF-YFY-FX+

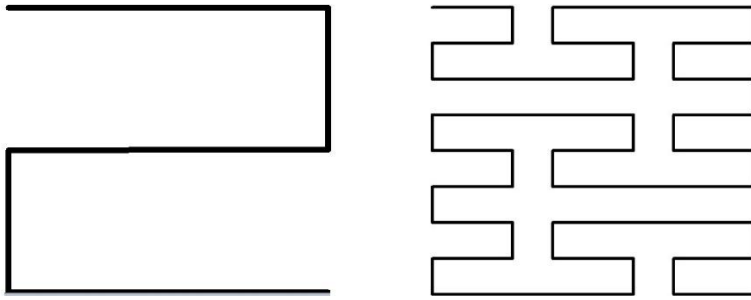
Les 3 F correspondent aux 3 joints



Les courbes de Peano

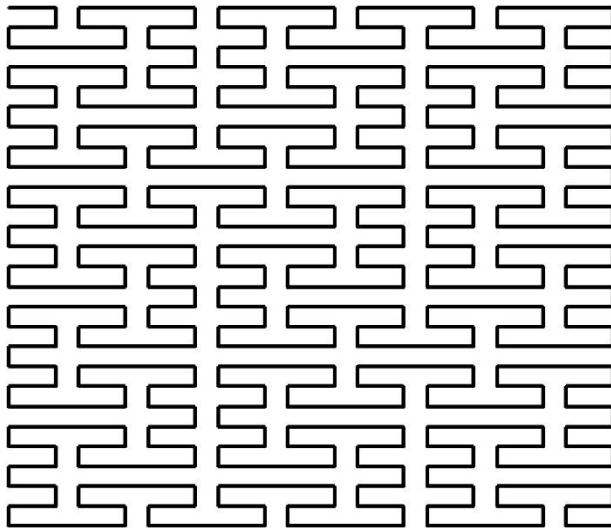


angle=90
 axiom=F
 F=F-F+F+F+F-F-F-F+F

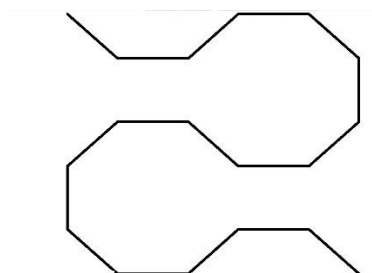


Même problématique pour le sens du dessin qui demande 2 règles, Y dans le sens des aiguilles d'une montre et X dans le sens contraire

angle=90
 axiom=X
 $X = XFYFX + F + YFXFY - F - XFYFX$
 $Y = YFXFY - F - XFYFX + F + YFXFY$



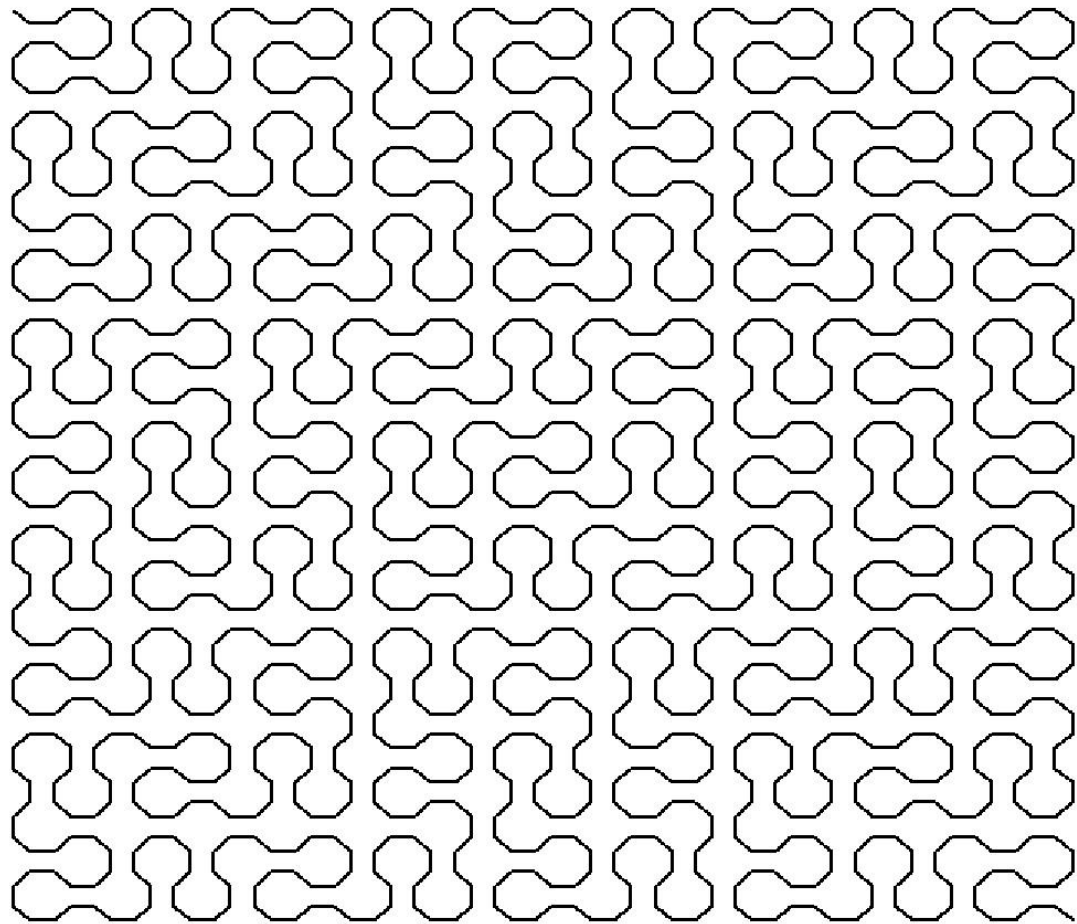
Et sa variante arrondie aux angles :



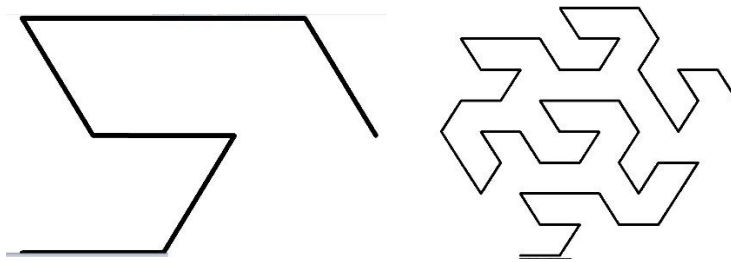
Paradoxalement, c'est plus simple car le motif se dessine toujours dans le même sens..
 Le motif est orienté diagonalement, il y a des joints partout pour tourner.

angle=45
 axiom=+z
 $z = fx$
 $f =$

$x = f_x - f_y - f_x + f_y + f_x + f_y + f_x + f_y + f_x - f_y - f_x - f_y - f_x - f_y - f_x + f_y + f_x$
 $y = f_y$



Le flocon de Gosper



Contrairement aux premiers cas où le motif était symétrique, ici le parcours dans les deux sens des aiguilles d'une montre est très différent

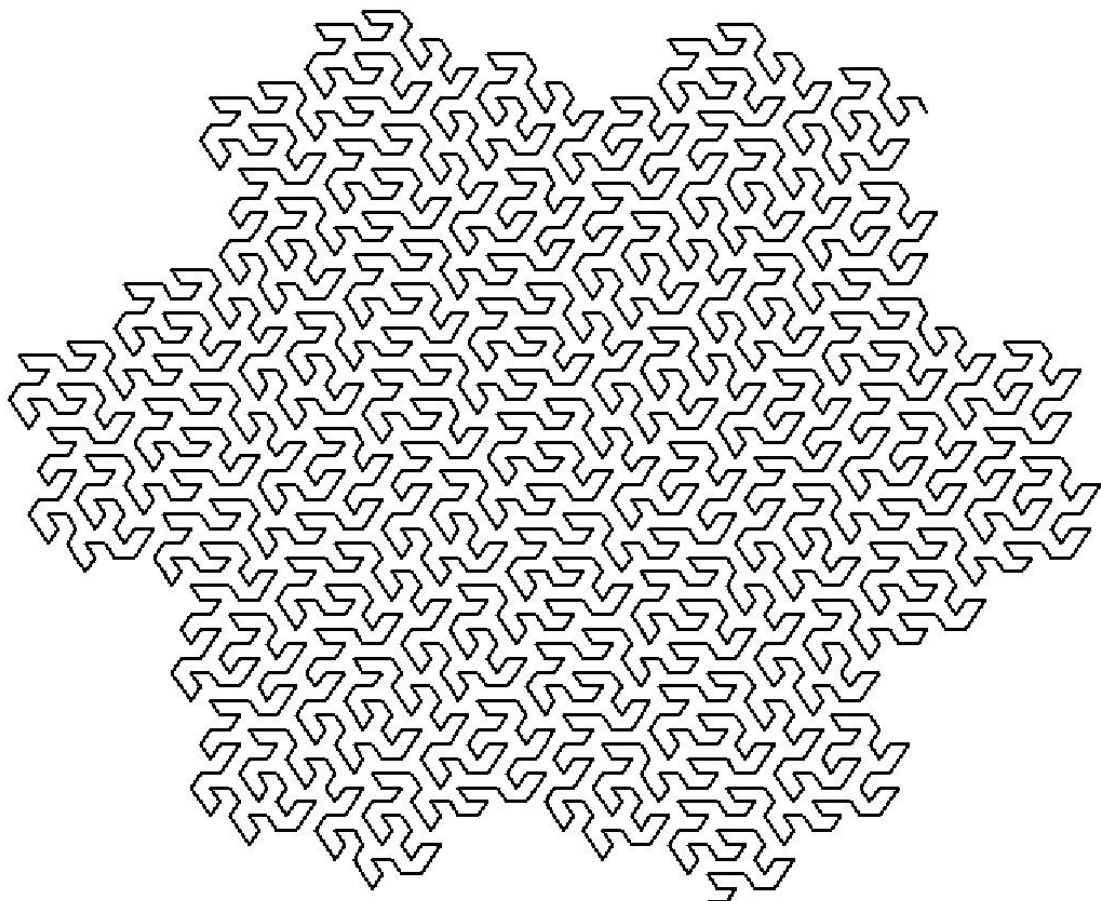
Heureusement le parcours inverse se déduit du premier, en commençant par la fin et en inversant les angles

Angle=60

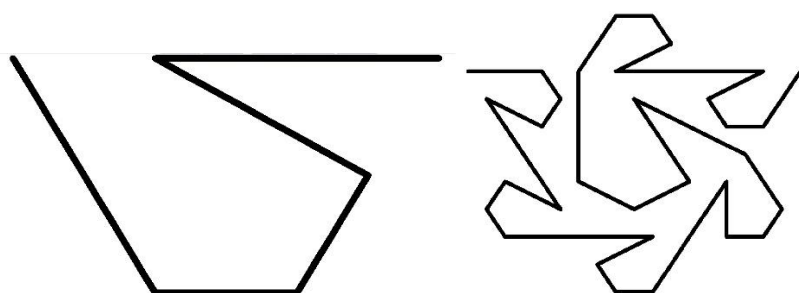
axiom=A

A=A-B--B+A++AA+B-

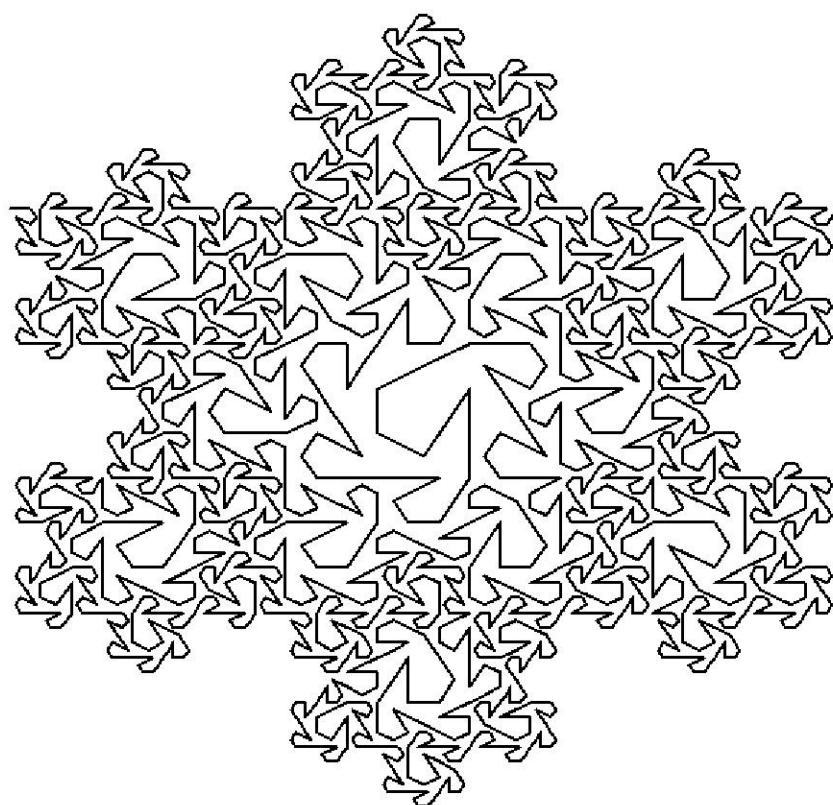
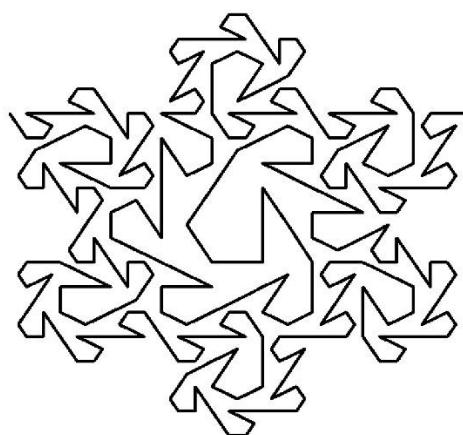
B=+A-BB--B-A++A+B



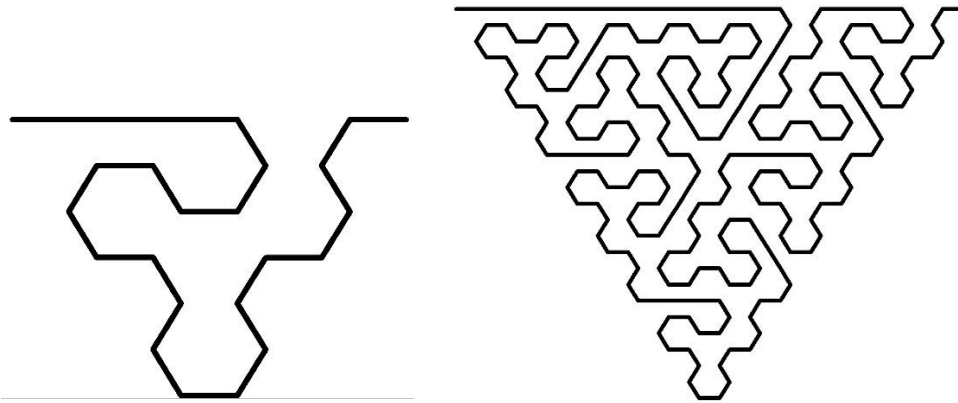
Snowflake



angle=30
 axiom=FR
 R=++!FRFU++FU++FU!---@Q3FU|~@IQ3!FRFU!
 U=!FRFU!|+@Q3FR@IQ3+++!FR--FR--FRFU!--
 F=



Un cas curieux : on ne passe pas d'un carreau à l'autre en suivant le motif mais en faisant une jointure



angle=45

axiom=z

z=ffffppppbb+fp+fpb+fp+fx-fp-fx-fp-fy+fp+fx-fp-fx-fp-fy+fp+fx-fp-fy+fp+fx

x=fpa+fp+fz-fp-fz-fp-fw+fp+fz

y=fpa-fp-fw+fp+fw+fp+fz-fp-fw

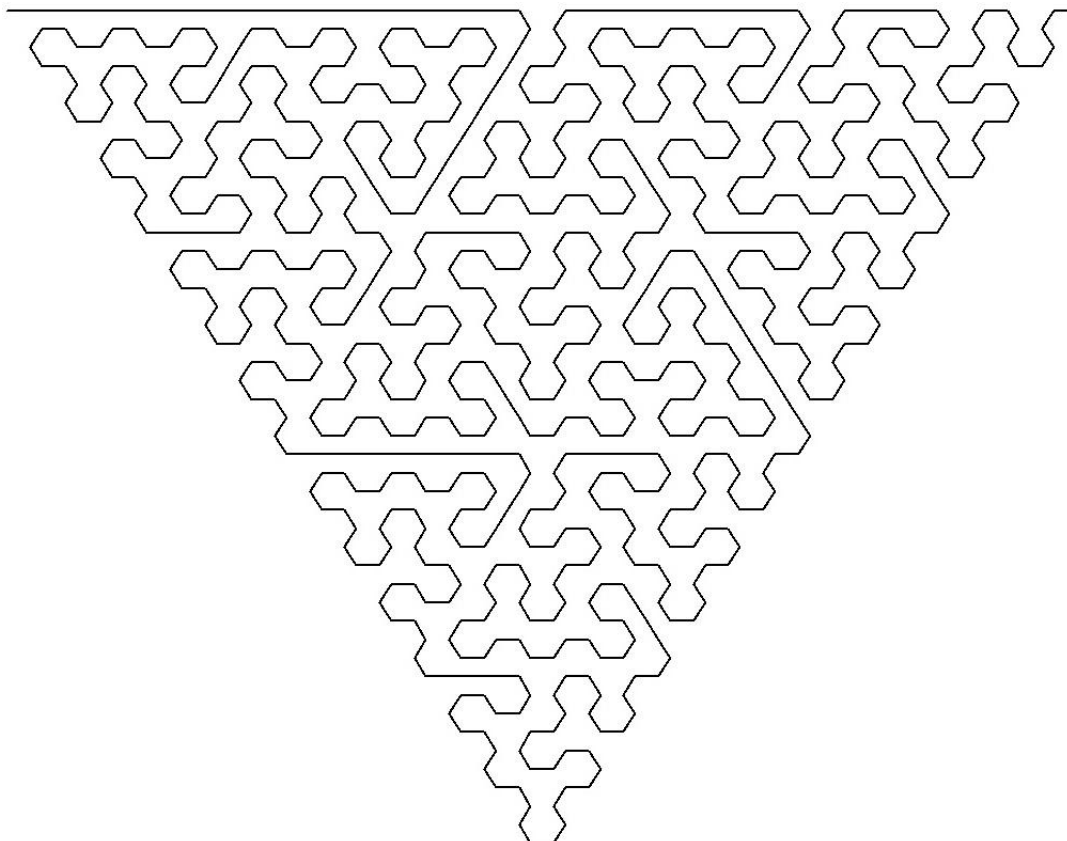
w=ffffppppbb-fp-fpb-fp-fy+fp+fy+fp+fx-fp-fy+fp+fy+fp+fx-fp-fy+fp+fx-fp-fy

a=fffffppppppbbb

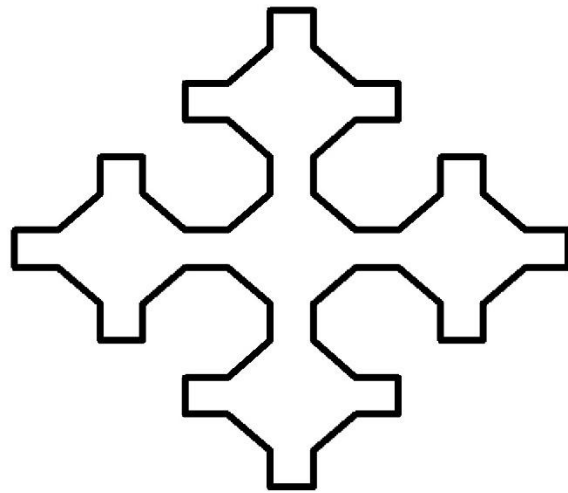
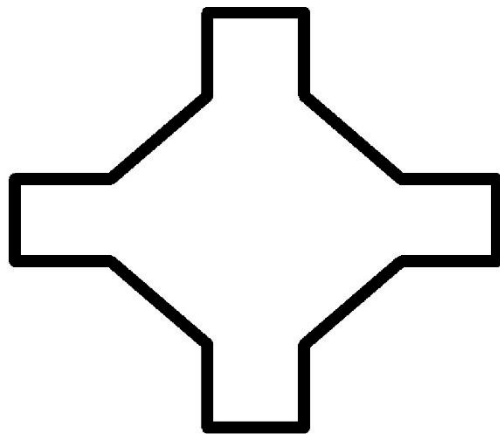
b=ffpppaa

p=fp

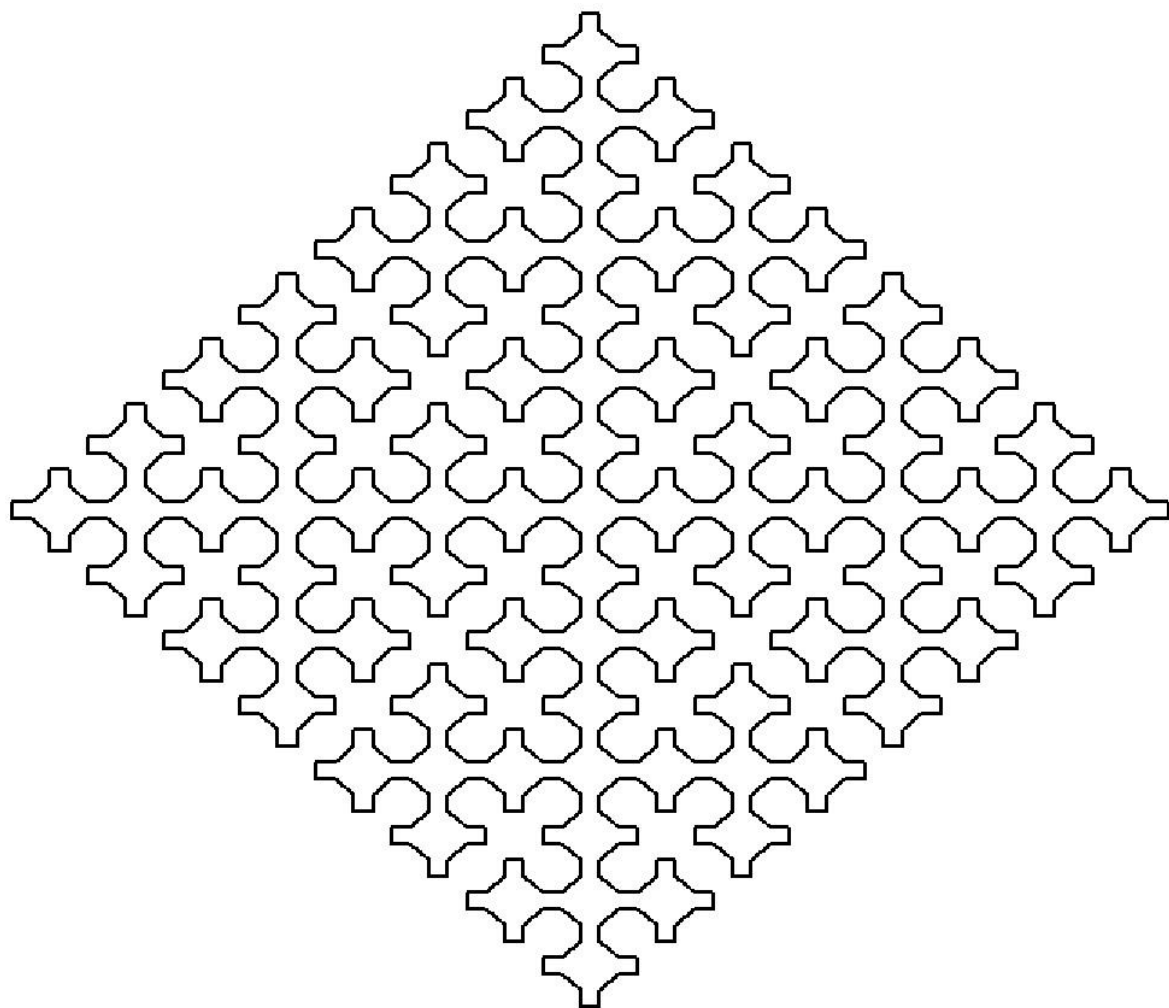
f=



La courbe de Sierpinski

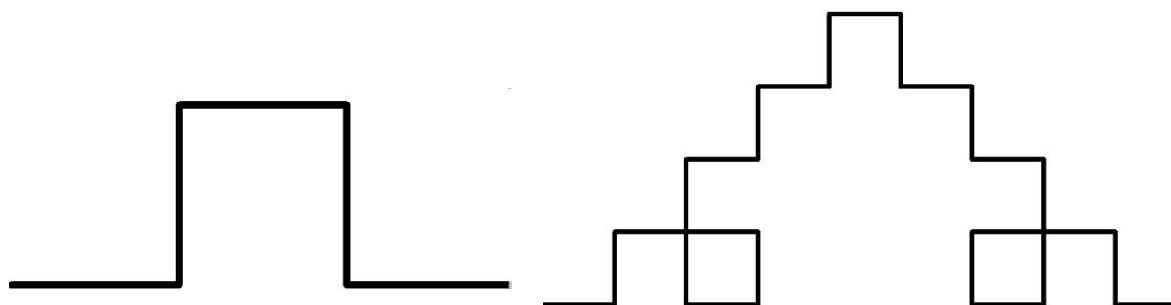


angle=45
 axiom=FXYY++F++FXYY++F (un carré)
 X=XY@Q2-F@IQ2-FXY++F++FXYY
 Y=-@Q2F-@IQ2FXYY



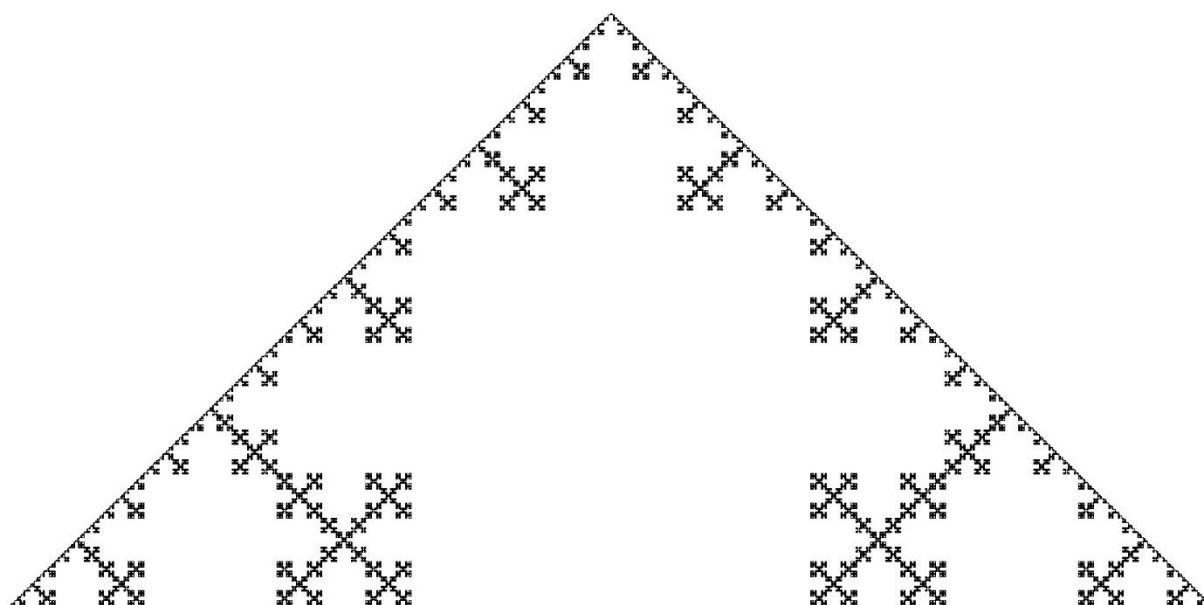
Courbes quadratiques de koch

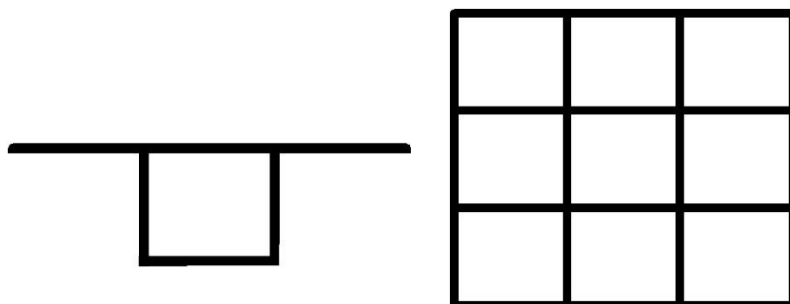
Elles sont basées sur un axiome de départ carré et le motif standard de koch, remplaçant un segment, est choisi pour créer un vide tout en autorisant un double parcours de certains éléments.



Snowflake

angle=90
axiom=F
 $F=F-F+F+F-F$





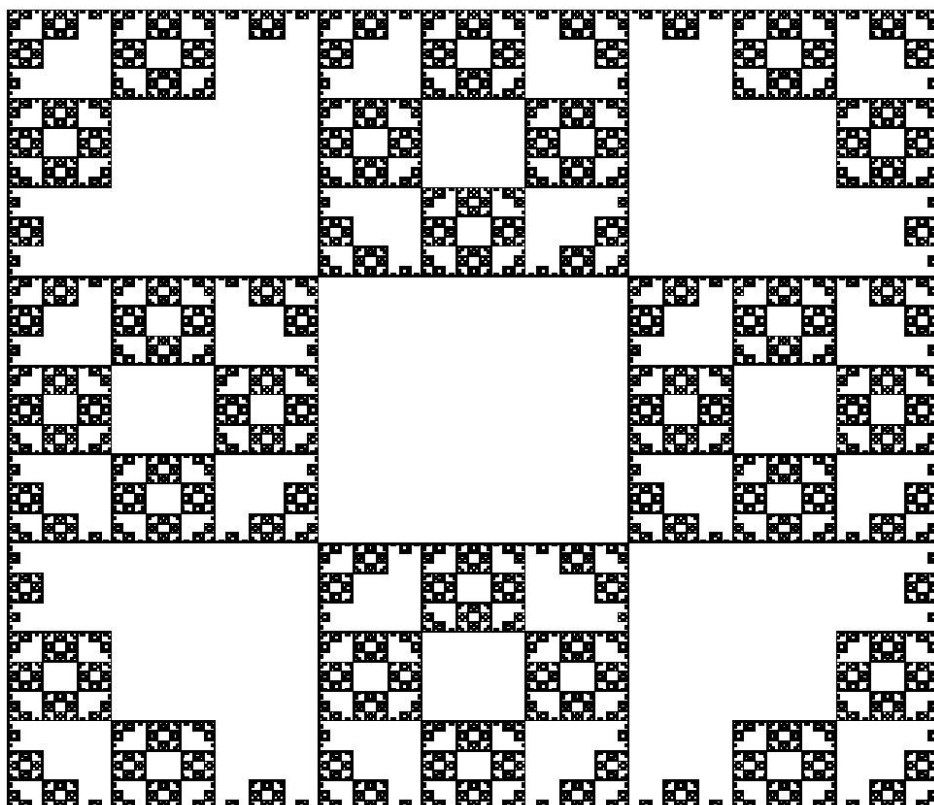
Koch carré 1

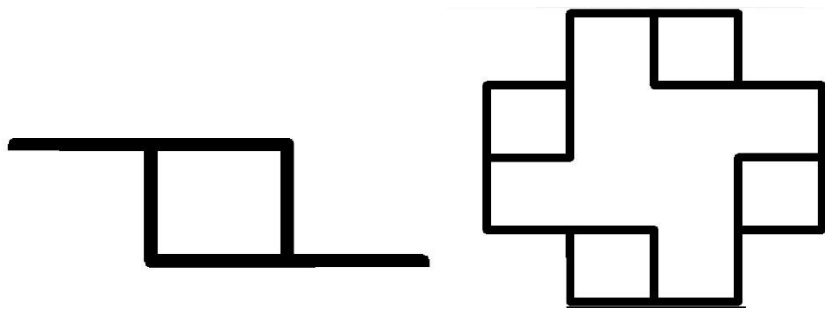
angle=90

axiom=F+F+F+F

F=FF+F+F+F+FF

Le carré central est parcouru par morceau d'un seul côté et dans un sens tel qu'il laisse un vide



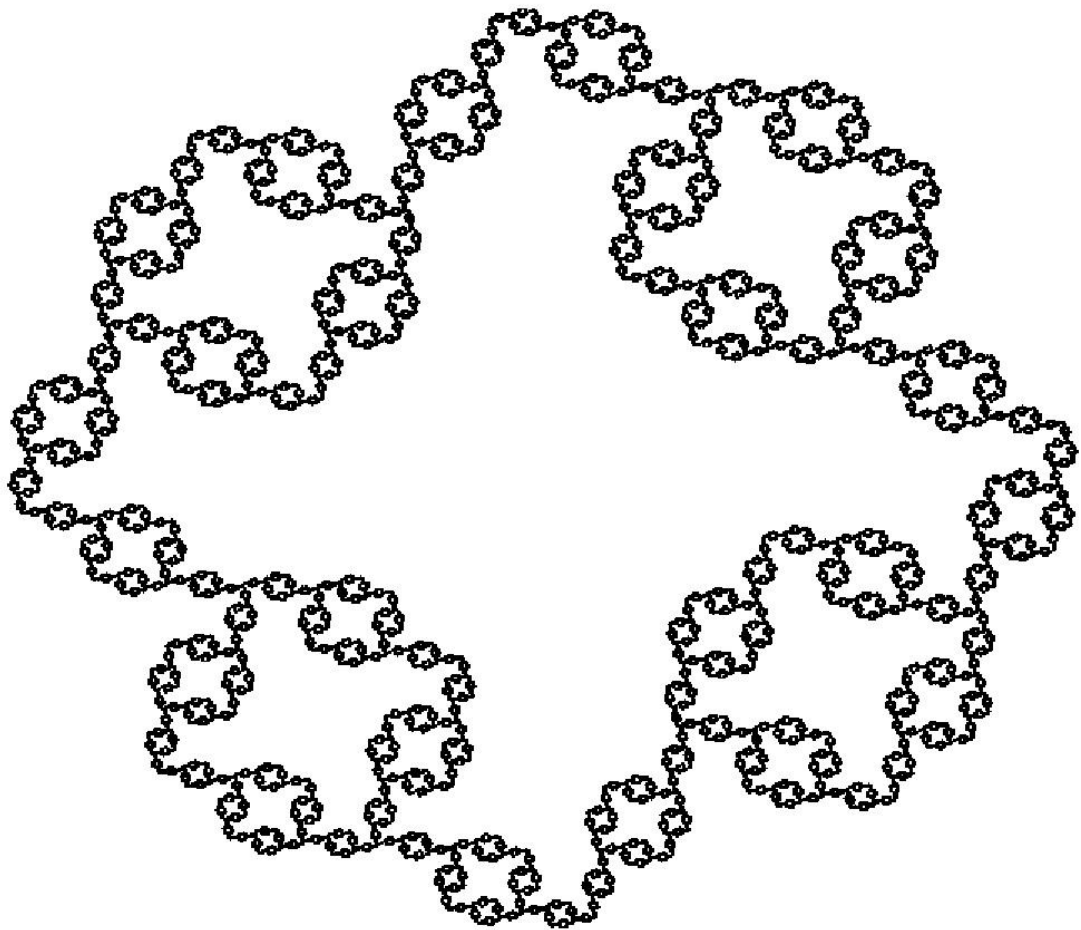


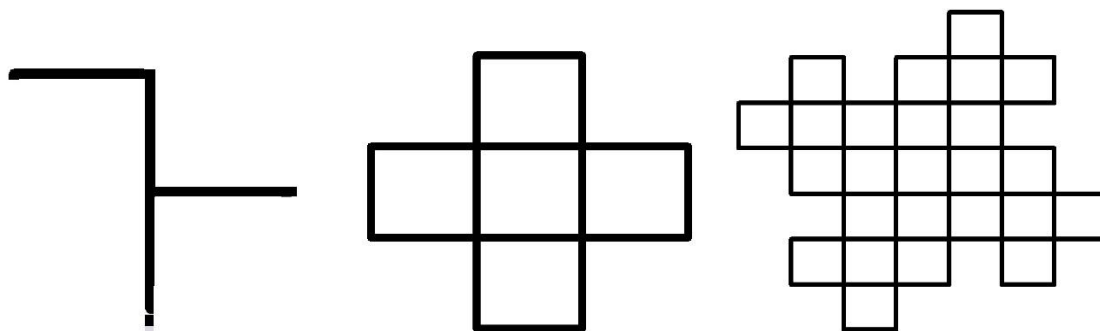
Koch carré

angle=90

Axiom=F+F+F+F

F=FF+F+F+F+F-F

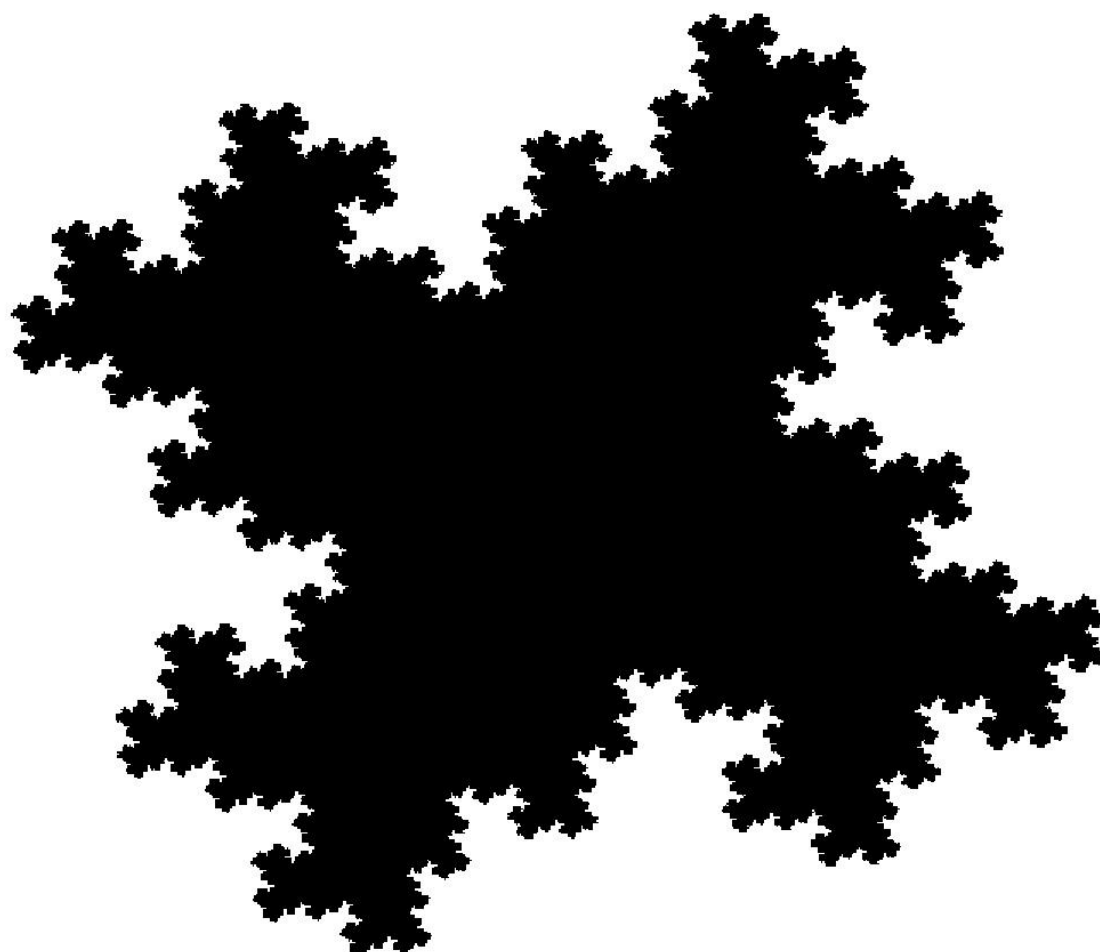


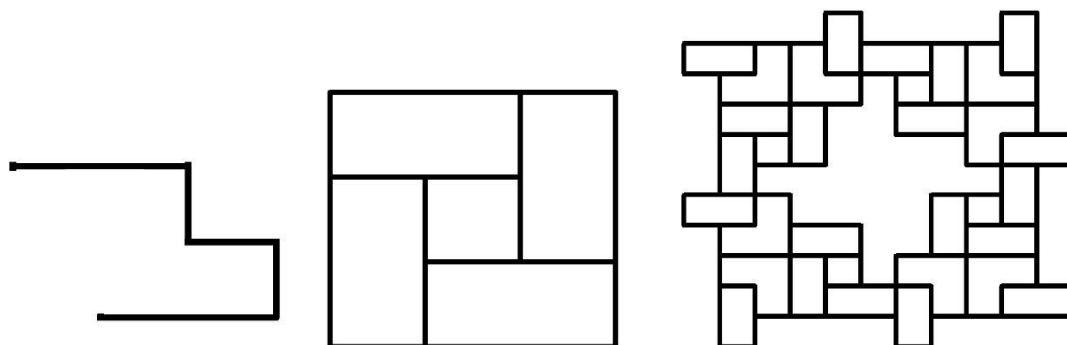


Koch carré 4

angle=90
 axiom=F+F+F+F
 F=F+FF++F+F

Le ++ fait demi tour en bas





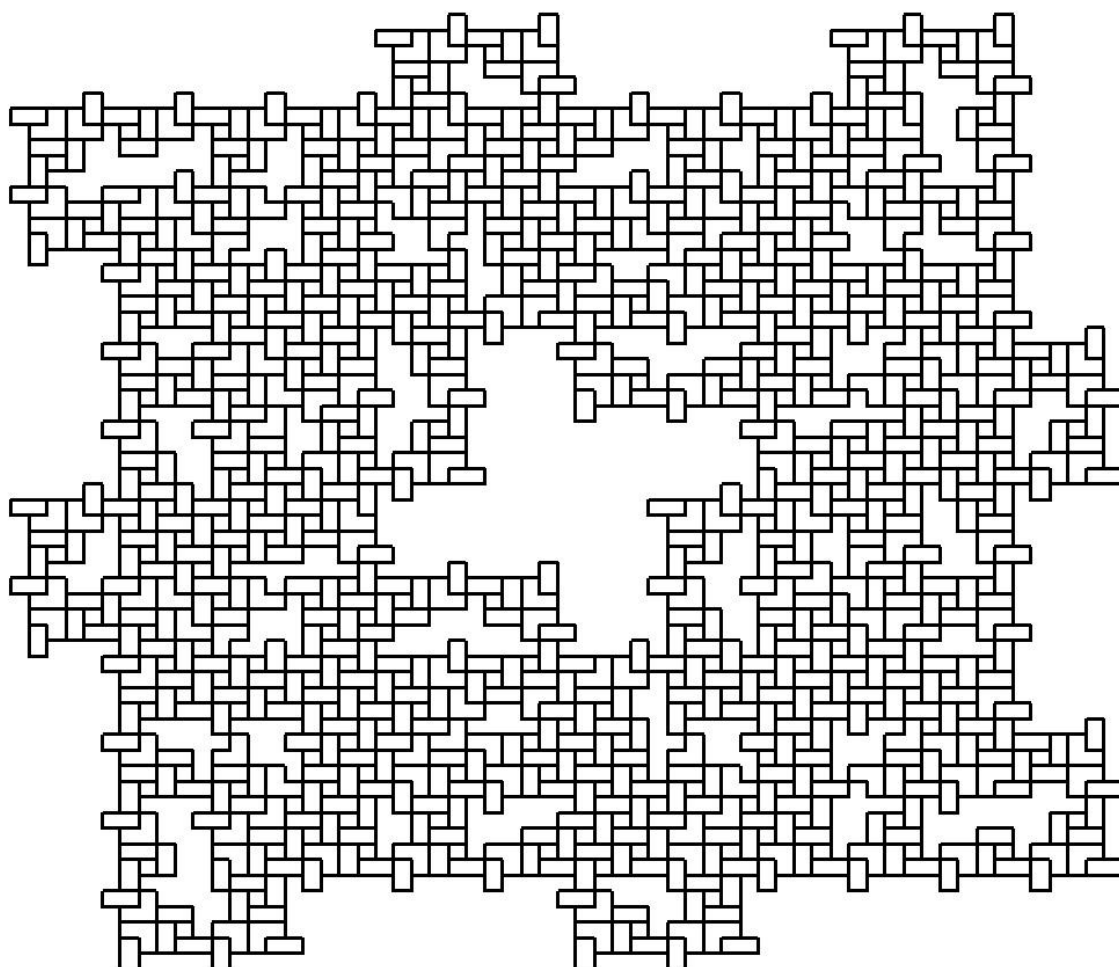
koch carré 2

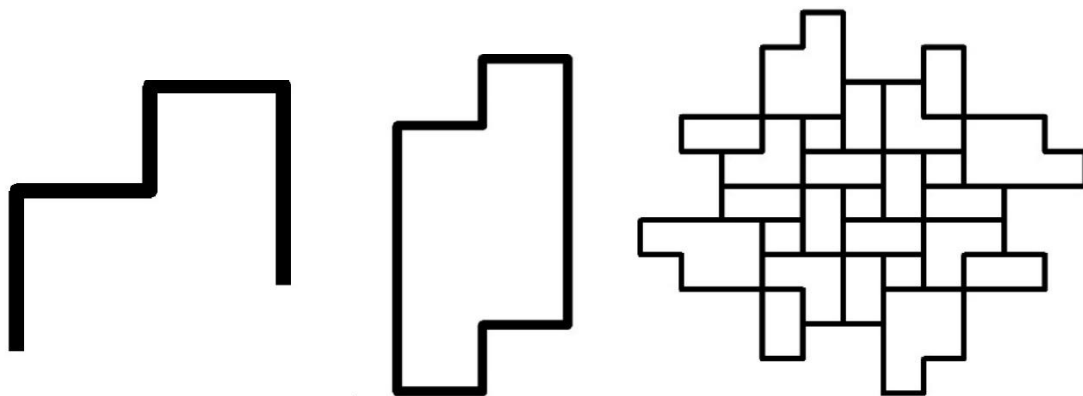
angle=90

axiom=F+F+F+F

F=FF+F-F+F+FF

Le carré central est parcouru par 4 séquences de 2 cotés





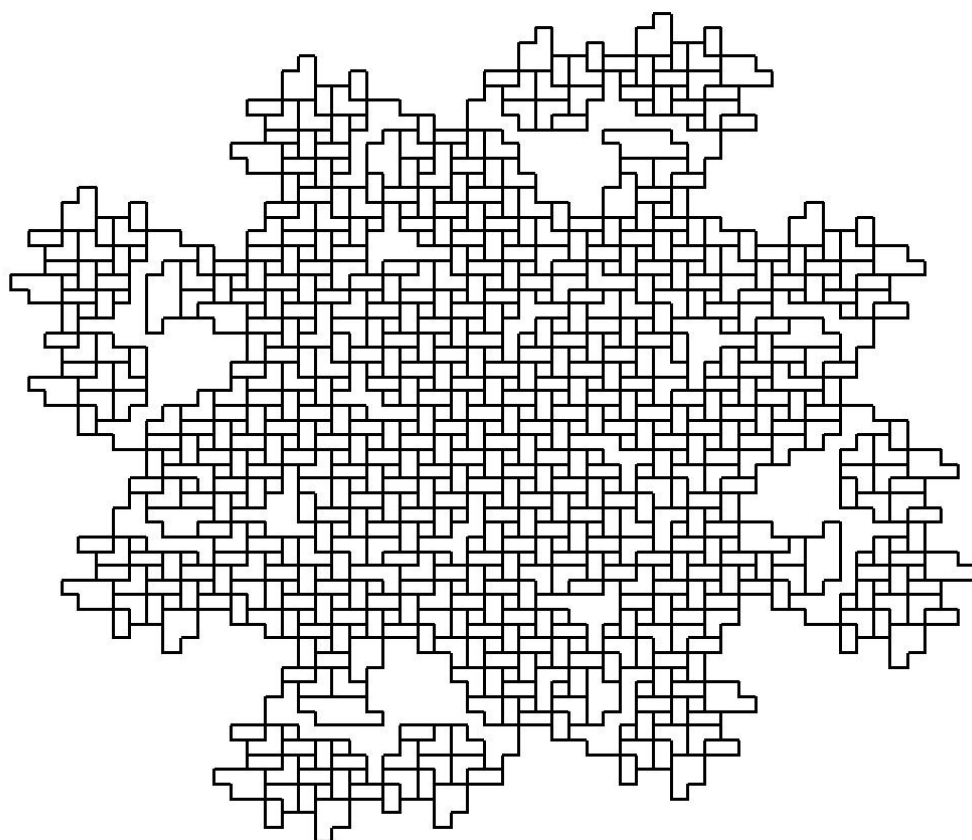
Koch carré 3

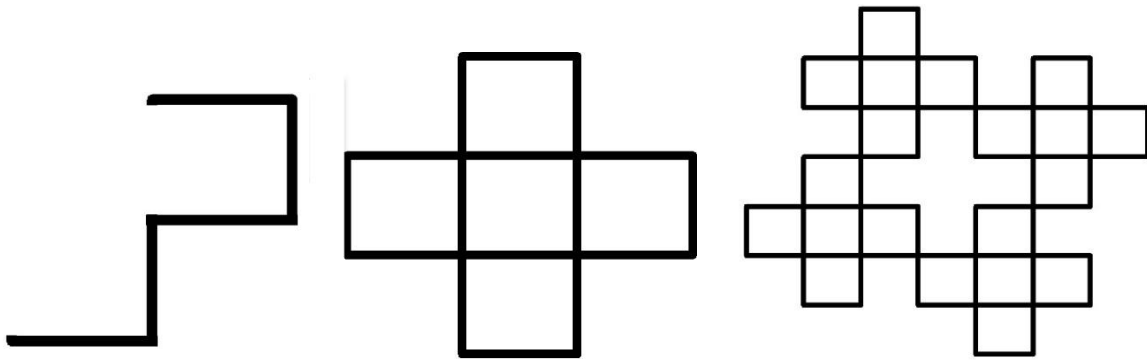
angle=90

axiom=F+F+F+F

F=-FF+F-F+F+FF

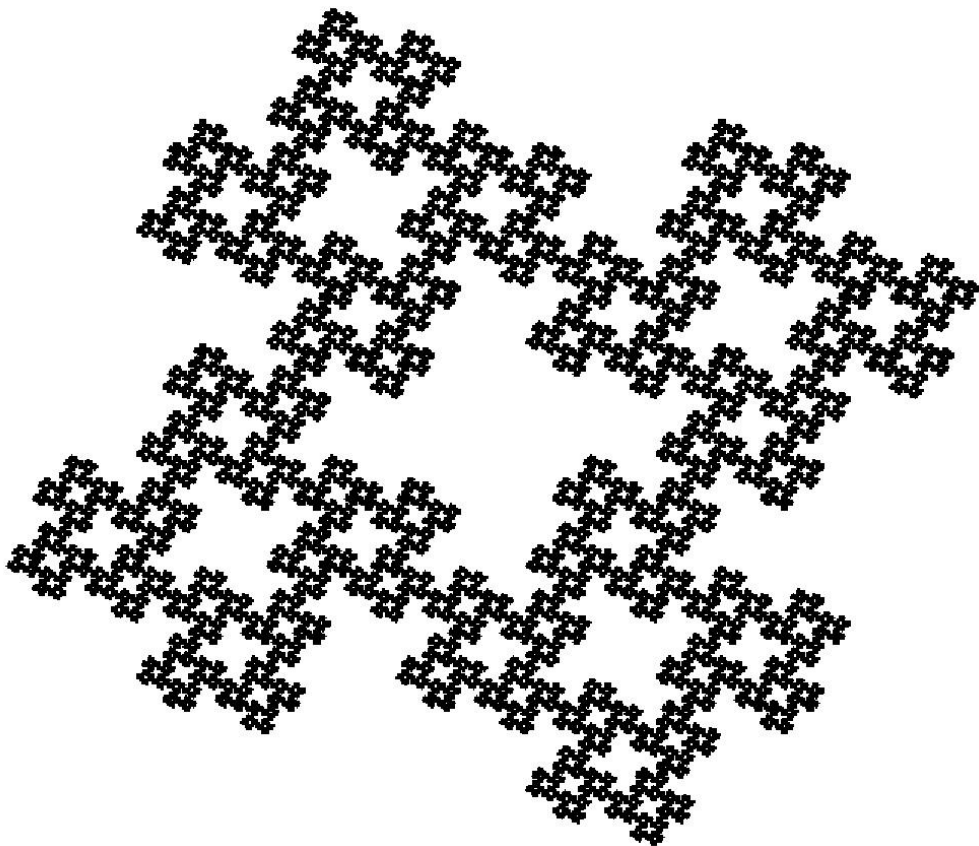
Je ne comprends pas pourquoi on ne voit pas sur le 2^{ème} dessin les transformations des 2 cotés verticaux du carré de départ, on devrait voir 4 fois 2F horizontaux, de même qu'on en voit 4 verticaux.

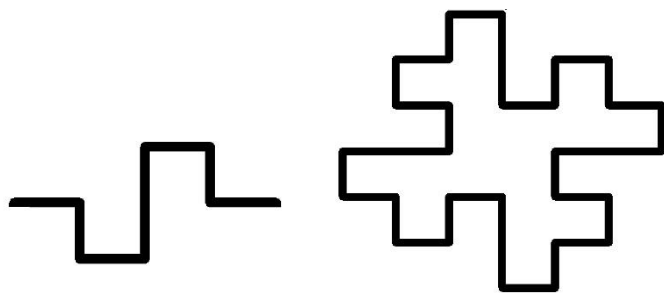




Koch carré 5

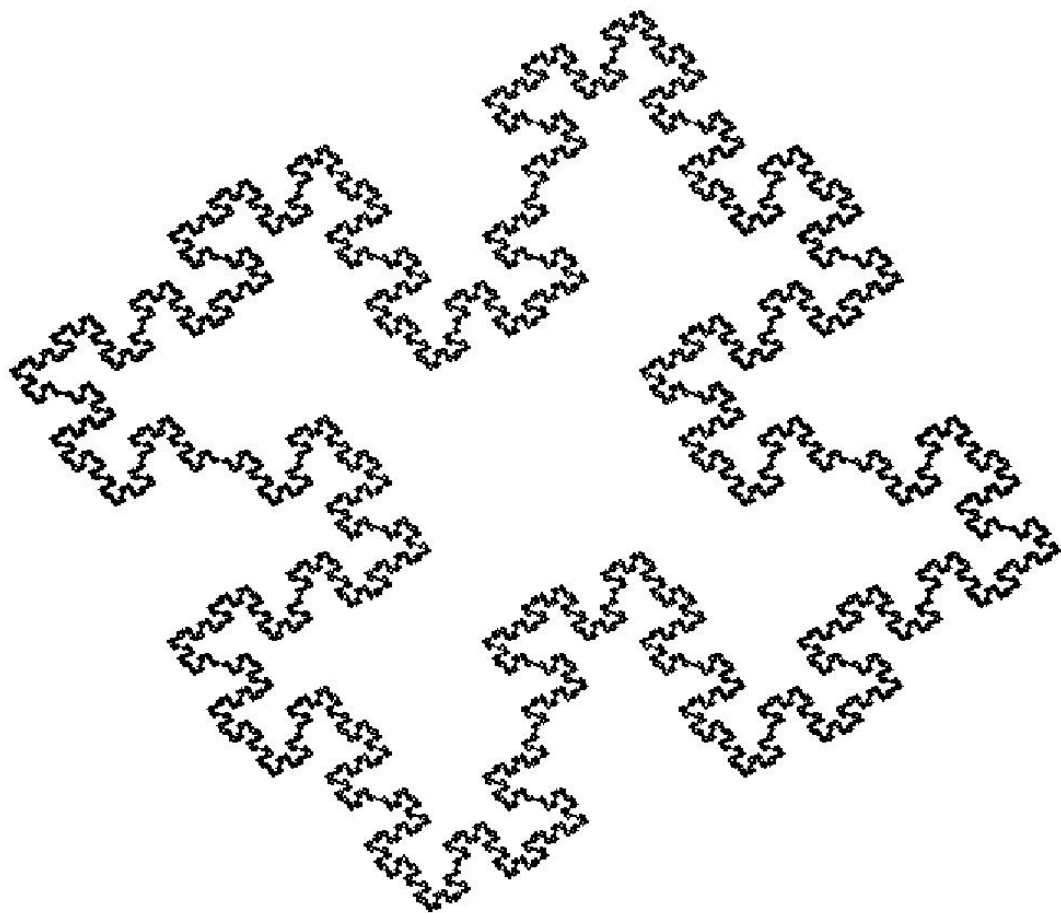
angle=90
 axiom=F+F+F+F
 F=F+F-F+F+F

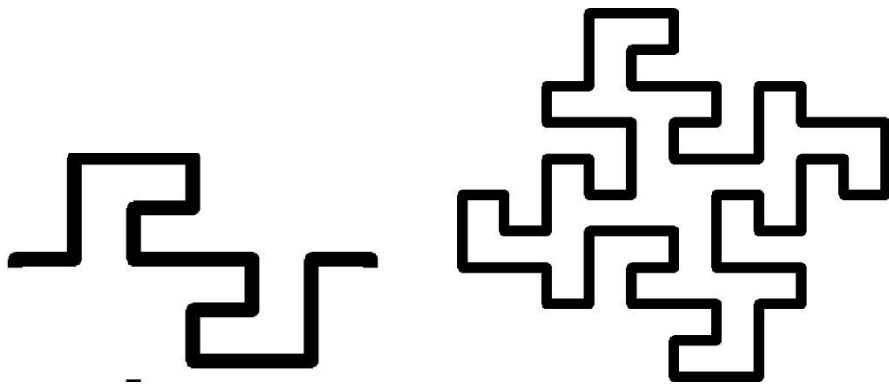




Koch island

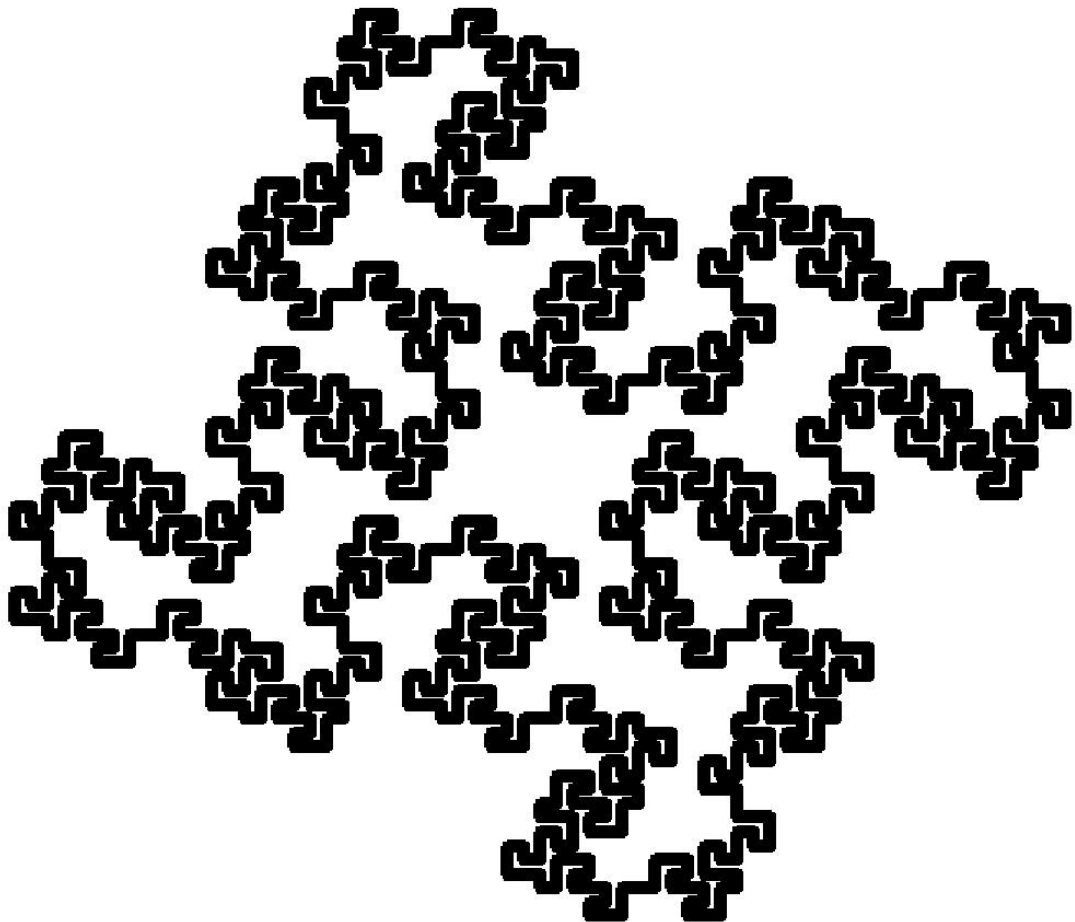
angle=90
 axiom=F+F+F+F
 F=F+F-F-FF+F+F-F



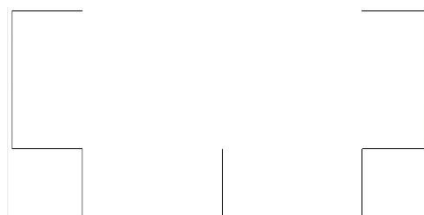
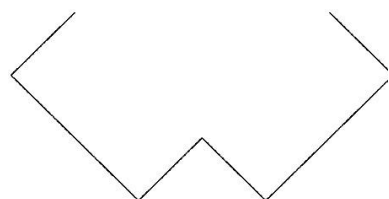
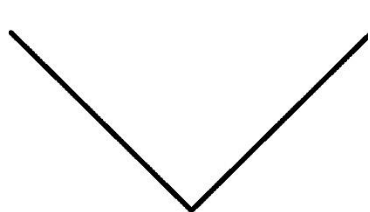


Quadratic koch island

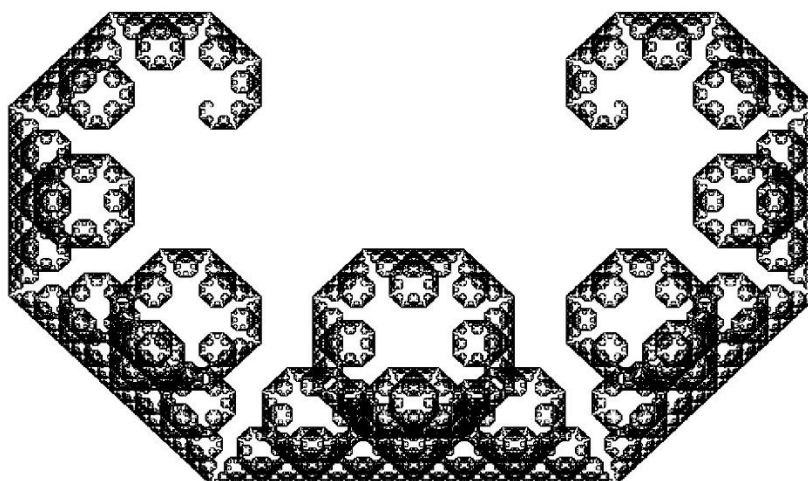
angle=90
 axiom=F+F+F+F
 F=F-FF+FF+F+F-F-FF+F+F-F-FF-FF+F



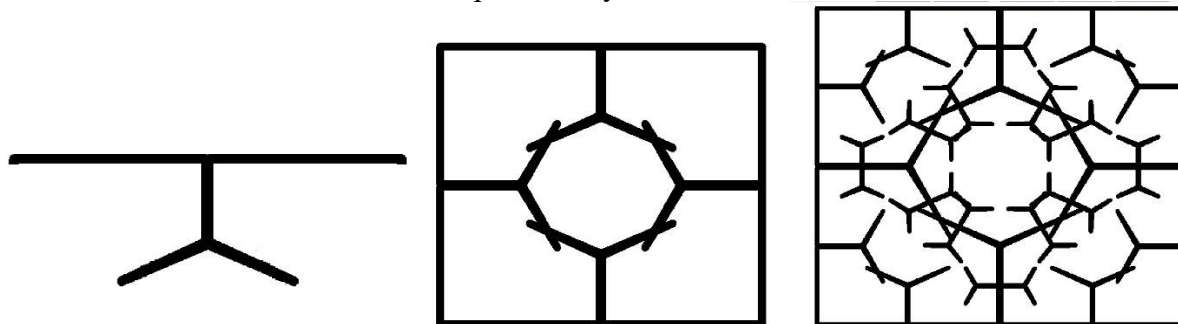
La courbe du C de Levy



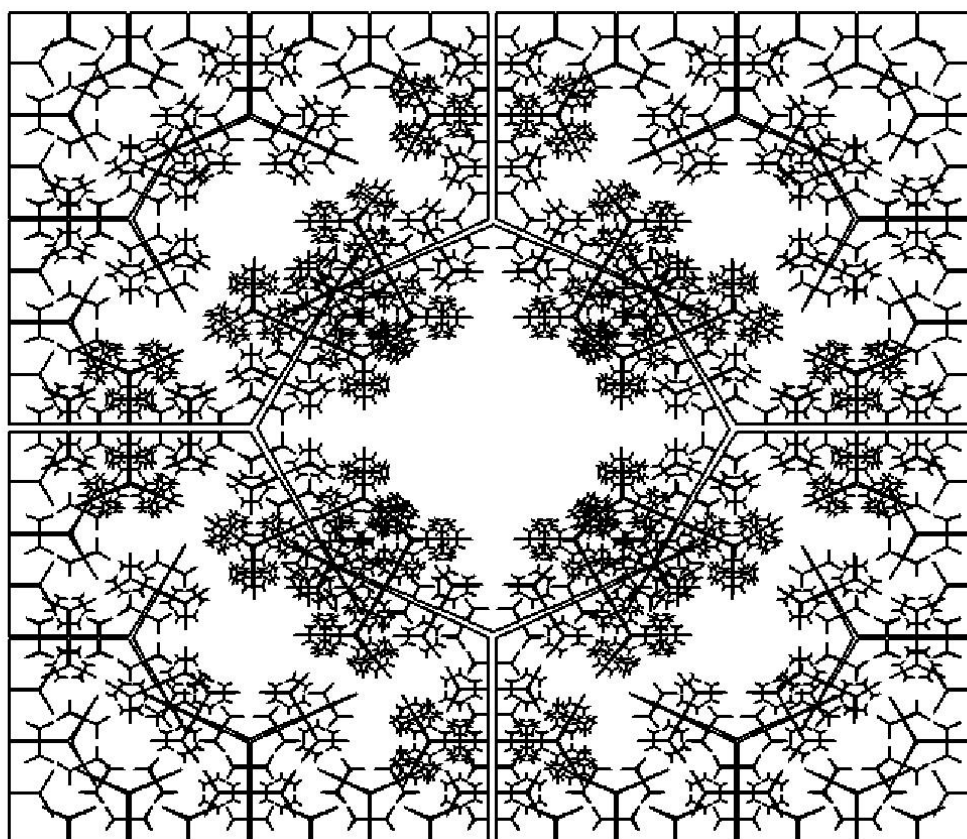
angle=45
axiom=F
F=+F--F+



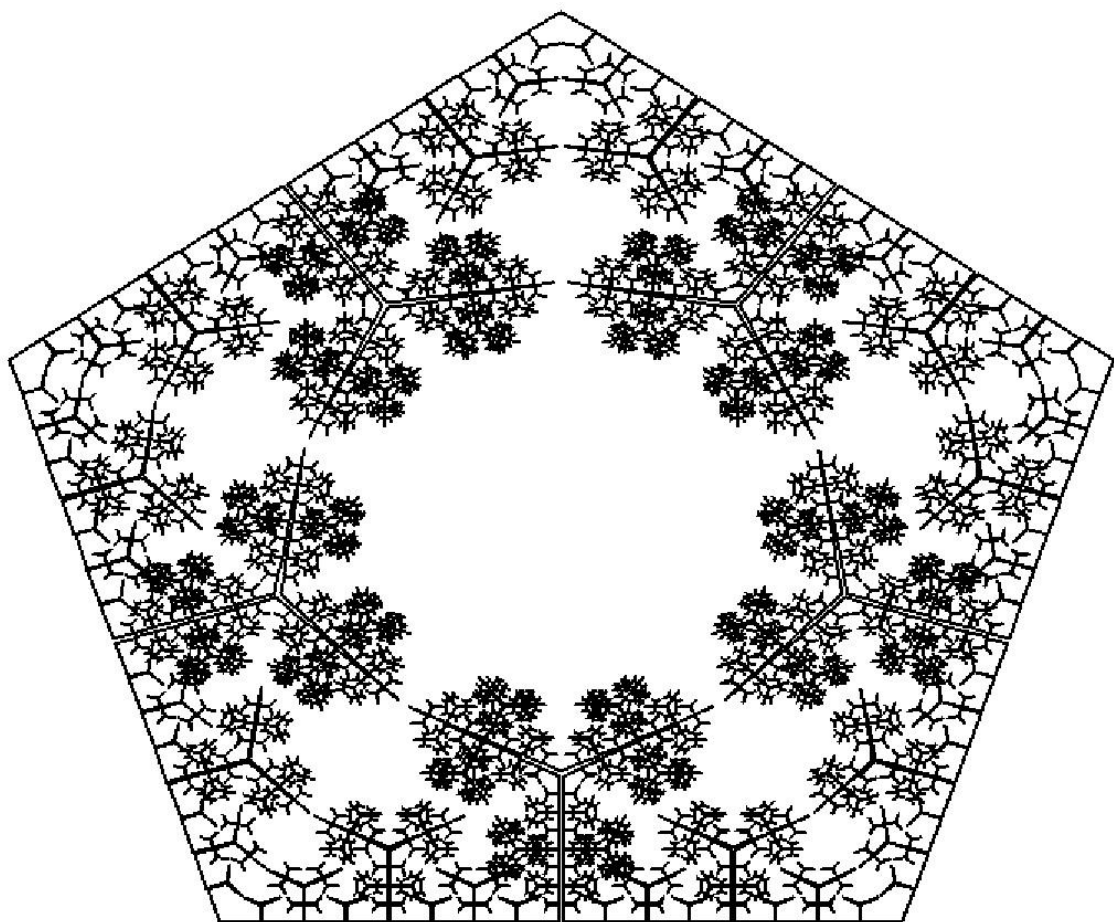
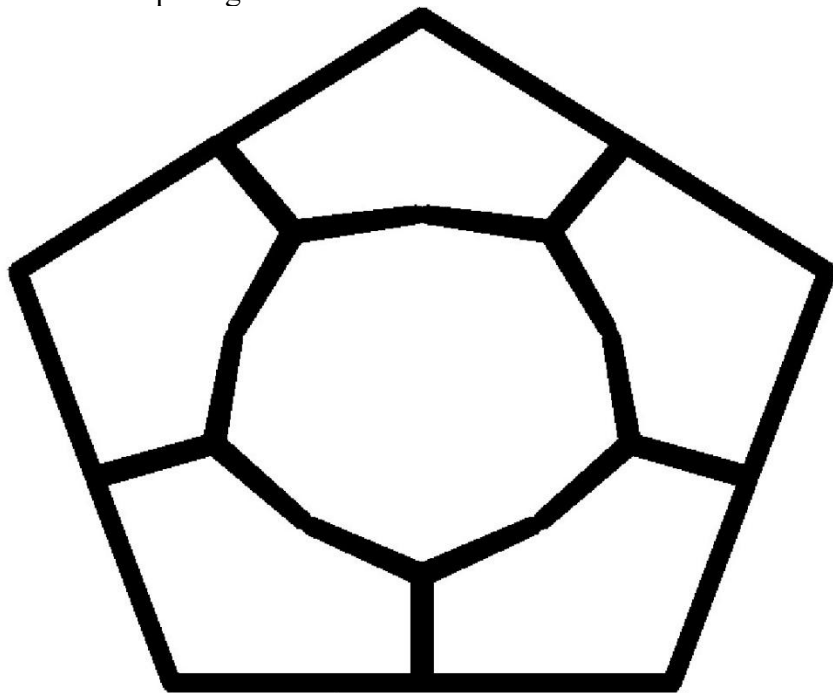
Une variante de la brisure de Cesaro par Antony Hanmer



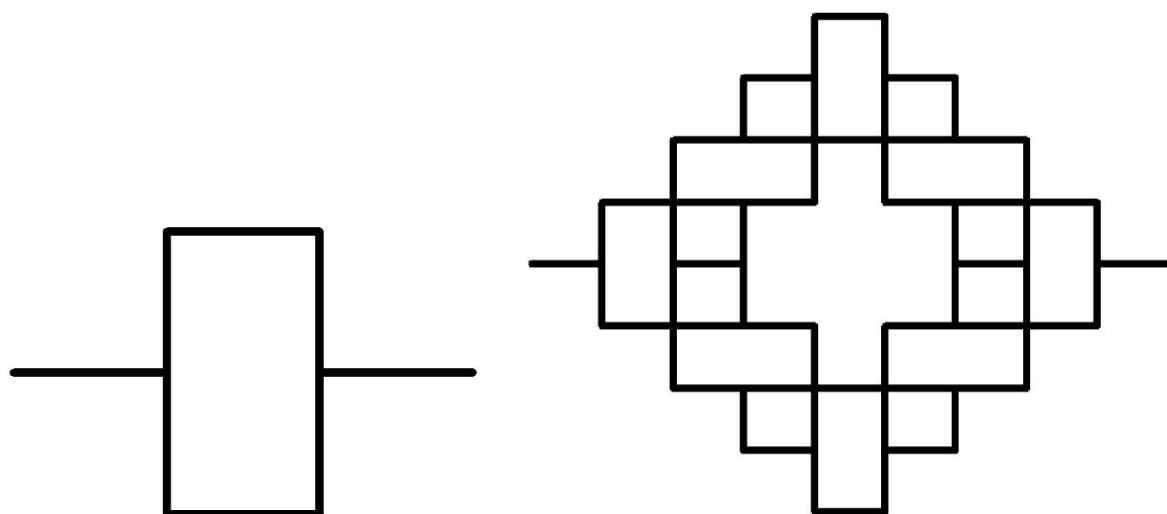
Hammer
angle=90
axiom=d+d+d+d
d=d+@.5d\62d/178d\52d/178d\62d+@2d



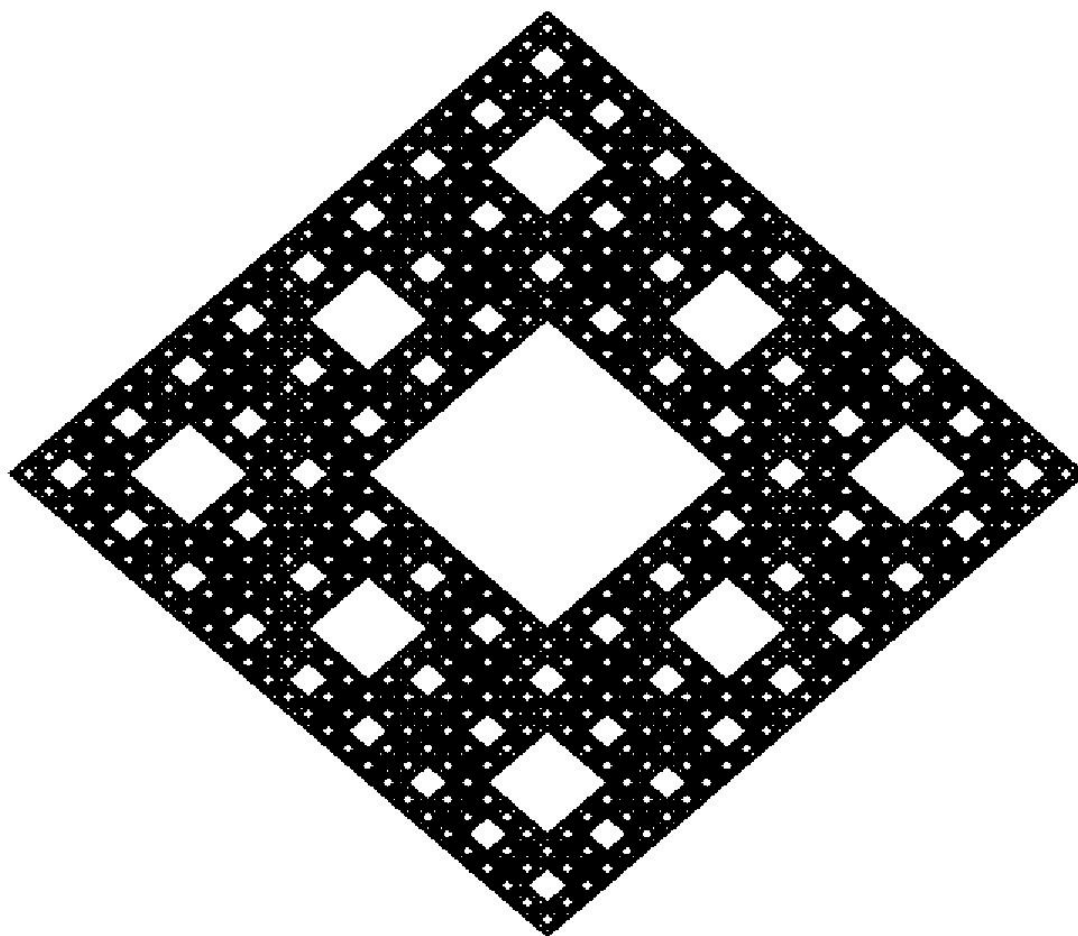
Hanmer sur un pentagone



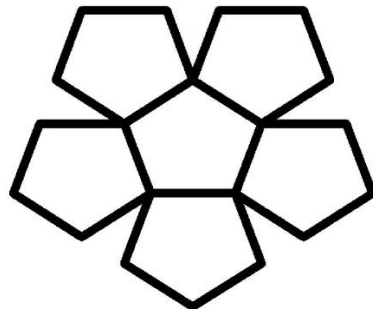
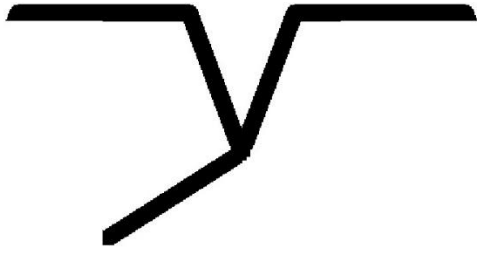
Le peano-tapis de Sierpinski



angle=90
 axiom=F
 F=F-F+F+F+G-F-F-F+F
 G=GGG



Les pavages



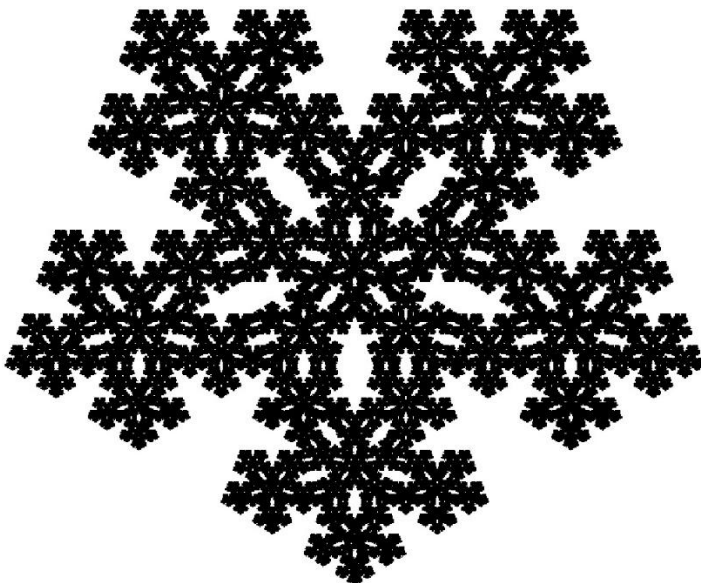
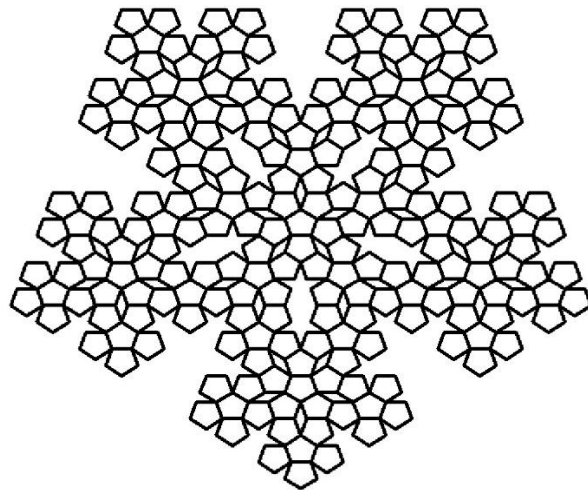
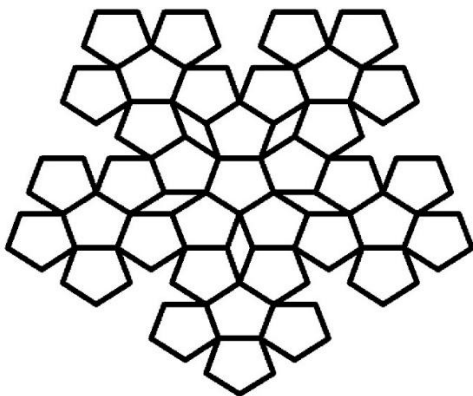
Pentaplexity

angle=36

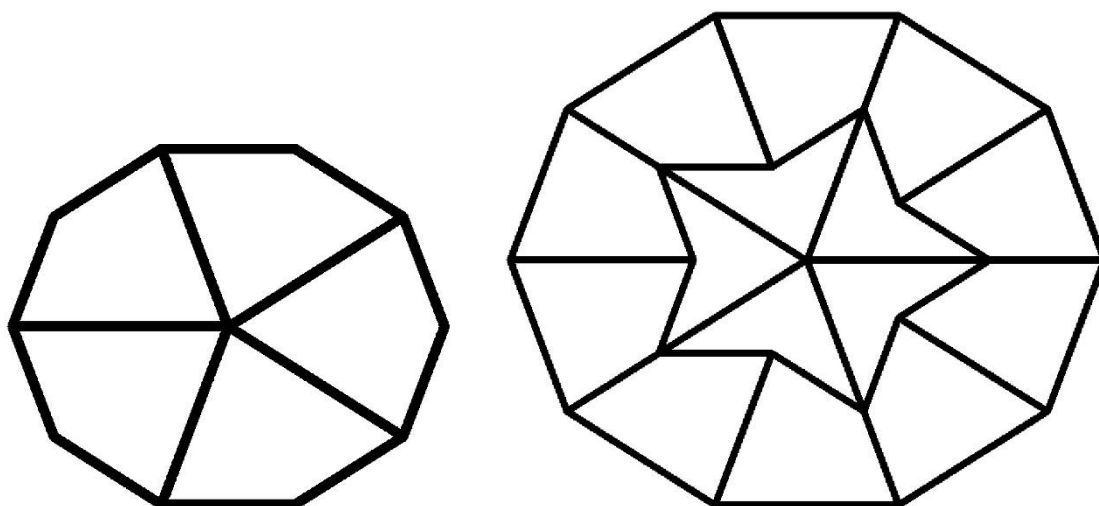
axiom=F++F++F++F++F

F=F++F++F|F-F++F

(le pentagone central est dessiné par F|F)



Kites and darts



angle=36

axiom=WG+XG+WG+XG+WG+XG+WG+XG+WG+X

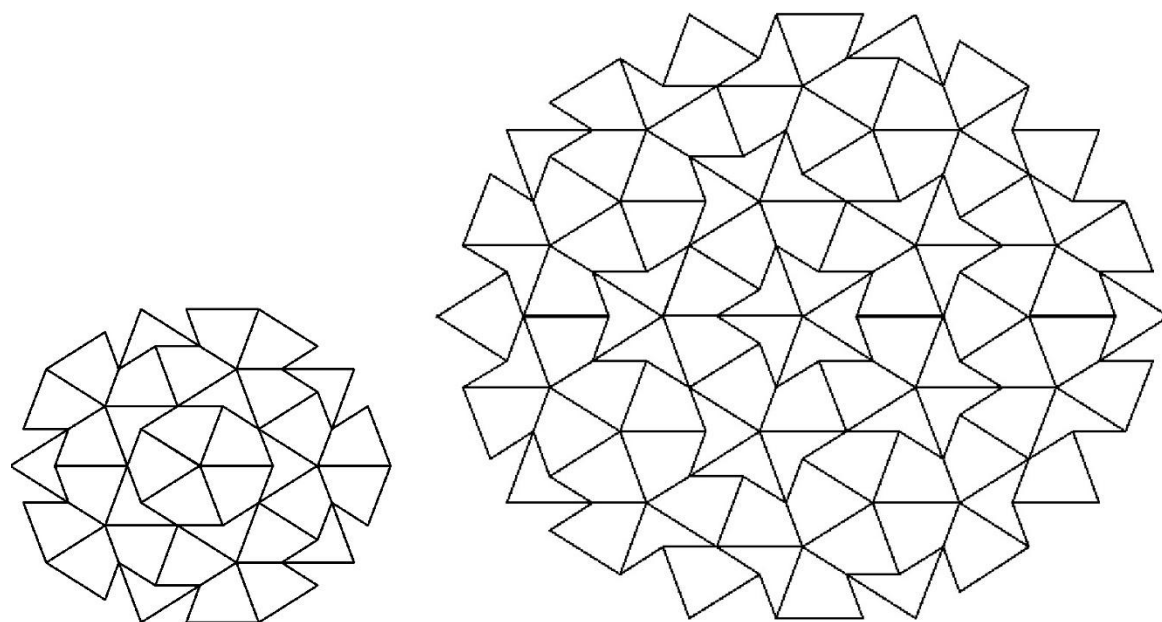
W=[F][++@1.618033989F][++G---@.618033989G|X-Y|G|W]

X=[F+++@1.618033989F][++@.618033989GZ|X|-G|W]

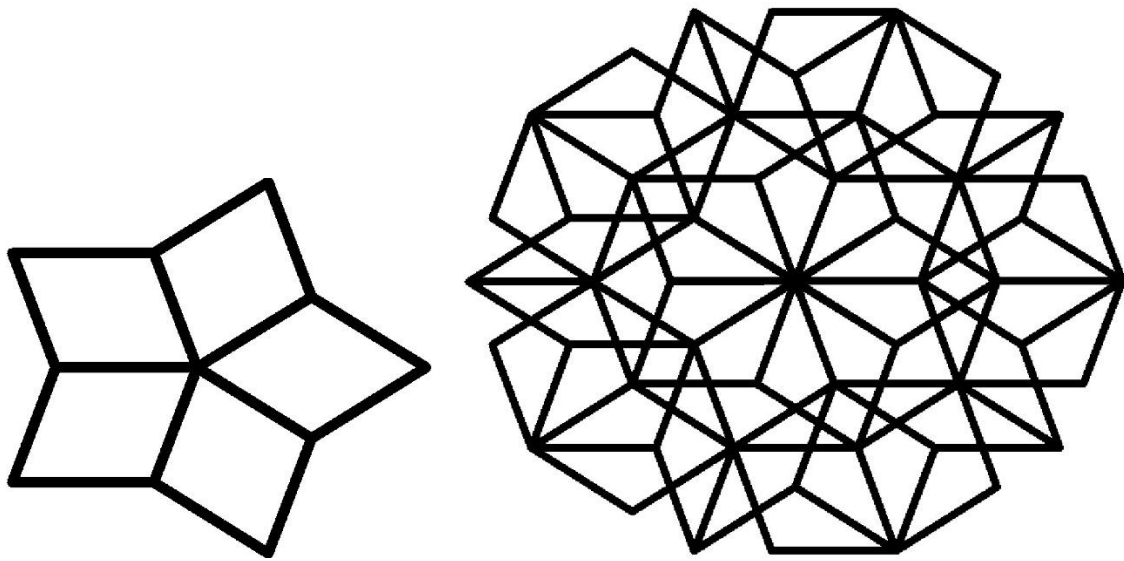
Y=[+F][@1.618033989F][+G@.618033989|Y+X]

Z=[-F][@1.618033989F][@.618033989G--WG|+Z]

F=



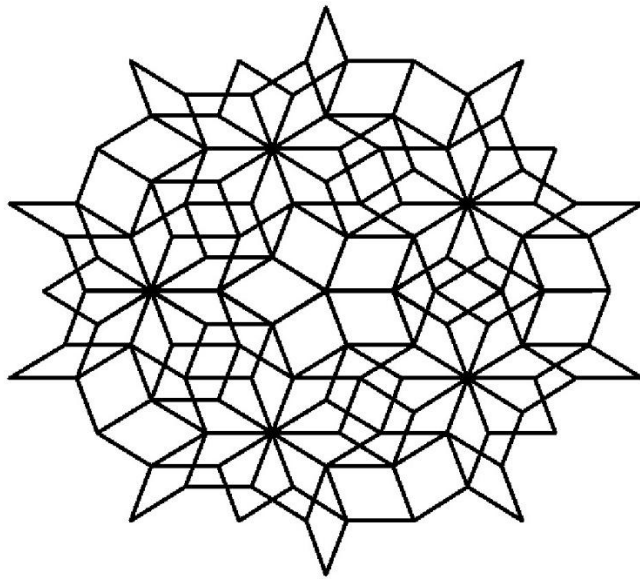
Penrose

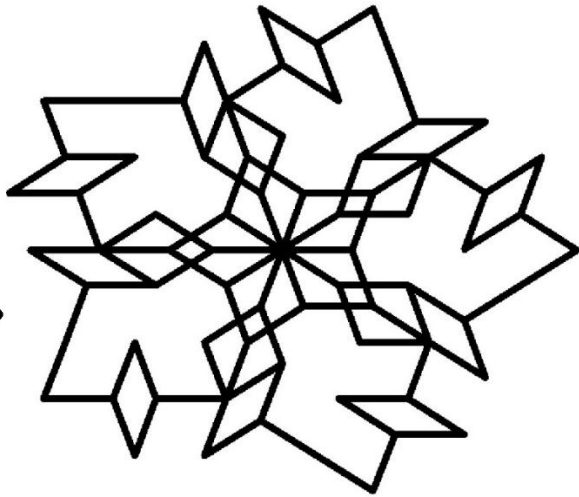
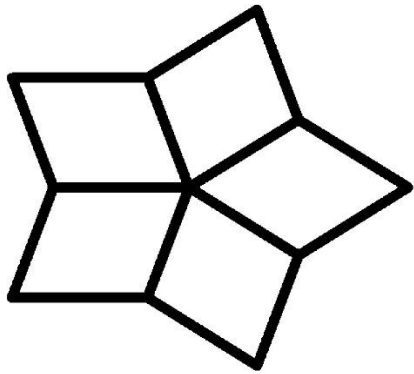


```

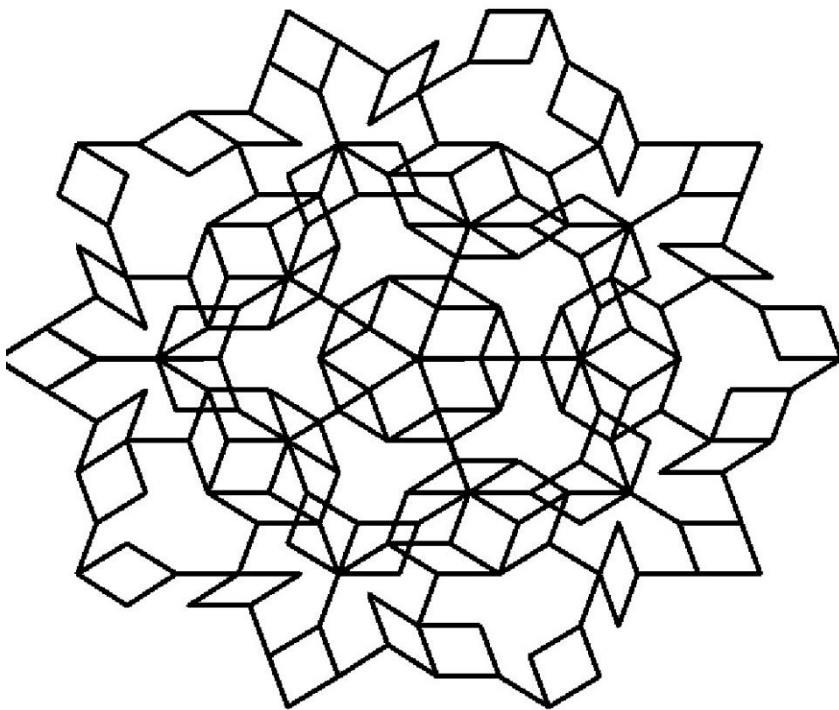
angle=36
axiom=[X][Y]++[X][Y]++[X][Y]++[X][Y]++[X][Y]", "W=YF++ZF----XF[-YF----WF]++
X=+YF- -ZF[---WF--XF]+
Y=-WF++XF[+++YF++ZF]-
Z=- -YF++++WF[+ZF++++XF]- -XF
F=

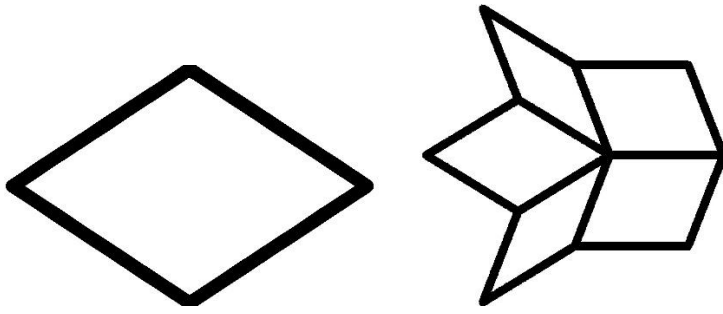
```



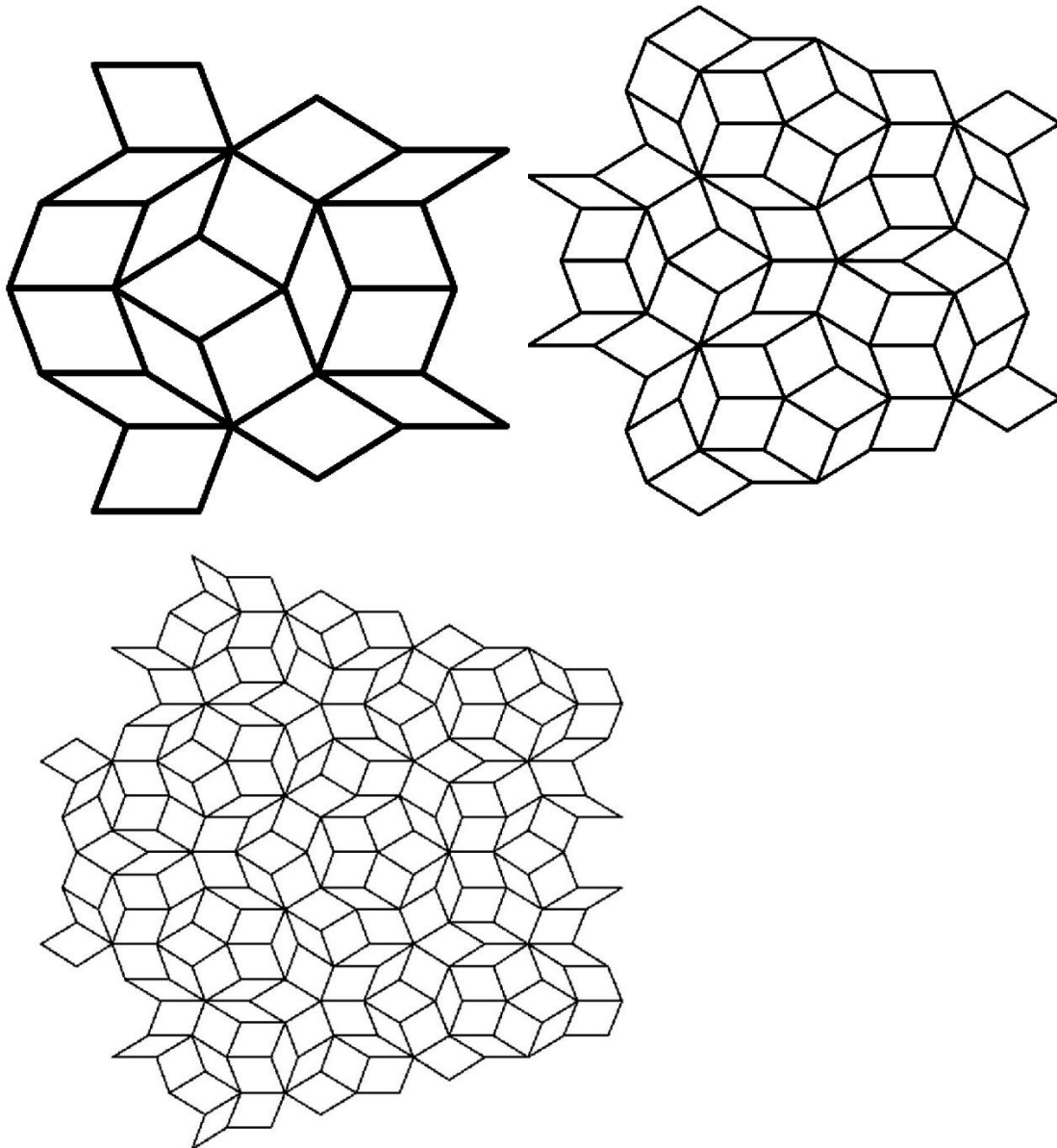


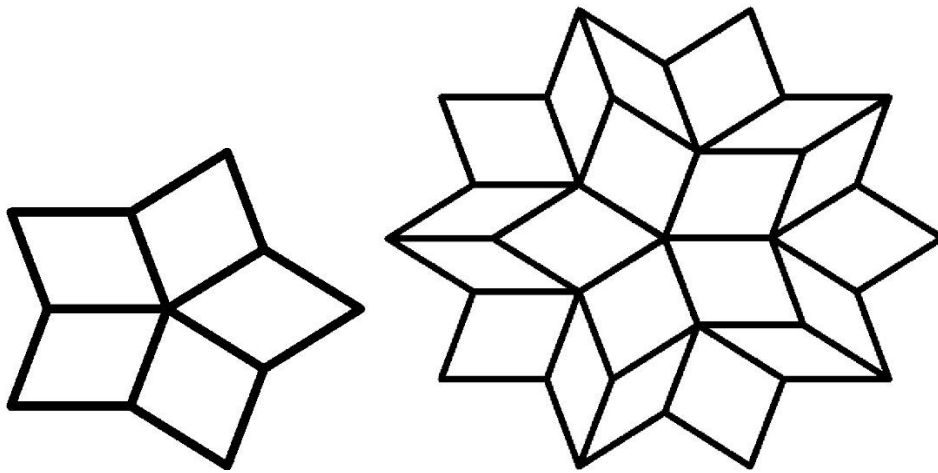
angle=36
 axiom=[7]++[7]++[7]++[7]++[7]
 6=81++91----71[-81----61]++
 7=+81--91[---61--71]+
 8=-61++71[+++81++91]-
 9=-81++++61[+91++++71]—71
 0=A","1=F
 F=



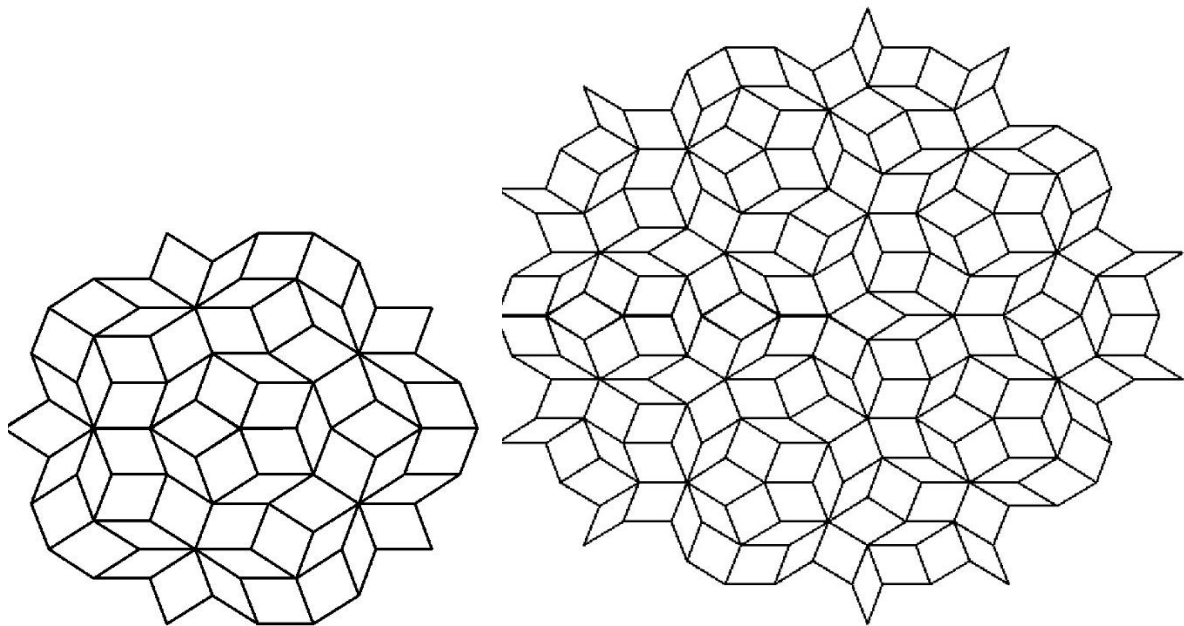


angle=36
 axiom=+WF--XF---YF—ZF
 W=YF++ZF----XF[-YF----WF]++
 X=+YF--ZF[---WF--XF]+
 Y=-WF++XF[+++YF++ZF]-
 Z=-YF++++WF[+ZF++++XF]—XF
 F=>





angle=36
 axiom=[7]++[7]++[7]++[7]++[7]
 6=81++91----71[-81----61]++
 7=+81--91[---61--71]+
 8=-61++71[+++81++91]-
 9=-81++++61[+91++++71]--71



Les dragons

La ligne brisée semble pivoter à chaque cycle et finit par une ile ou un archipel fractal plein

Dragon de Heighway

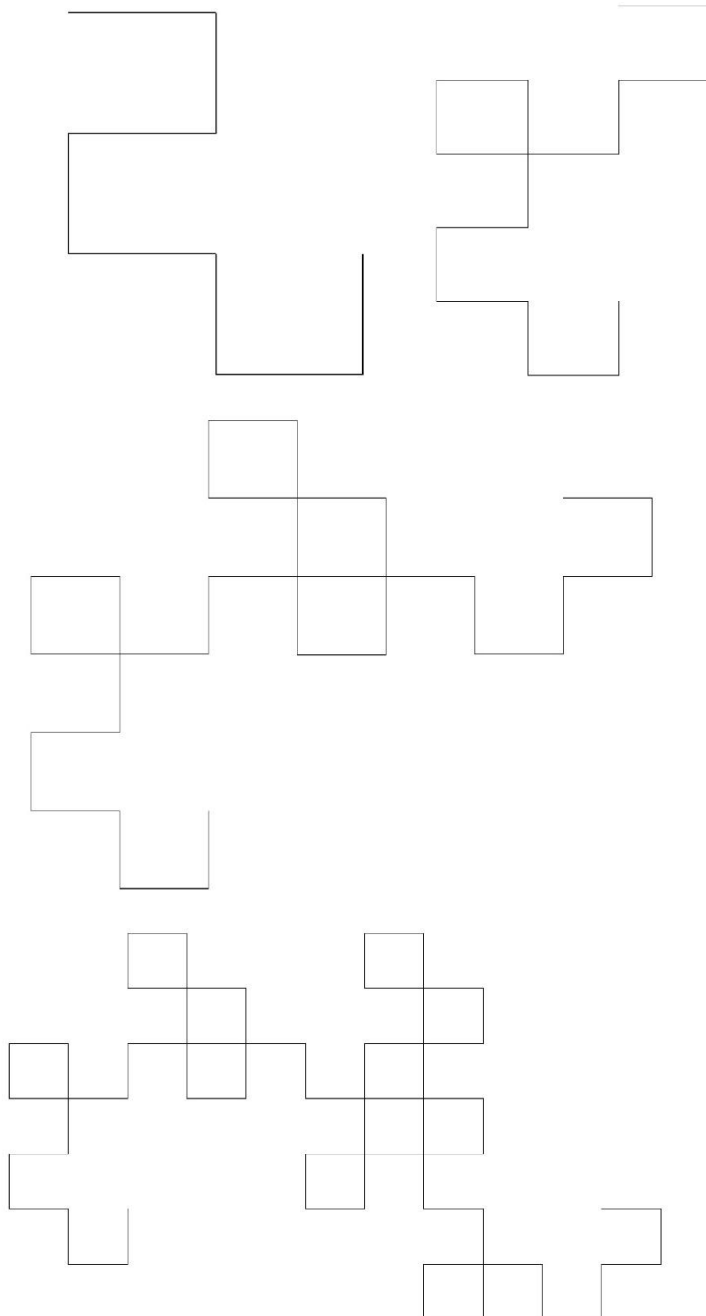
angle=90

axiom=FX

$X = X + YF +$

$Y = -FX - Y$

F=



Dragon au bord arrondi

angle=45

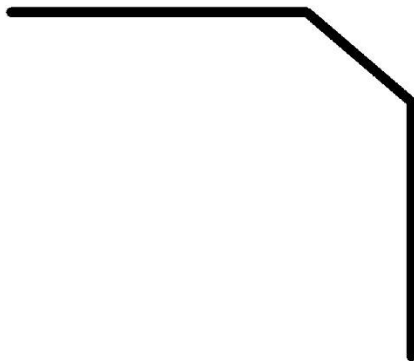
axiom=X

$X = FX + @.500FZ@2.00 + FY$

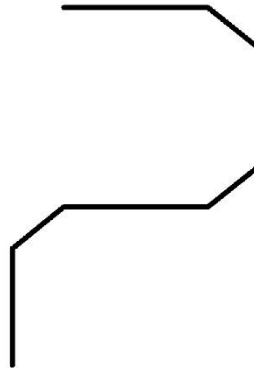
$Y = FX - @.500FZ@2.00 - FY$

$Z = FZ$

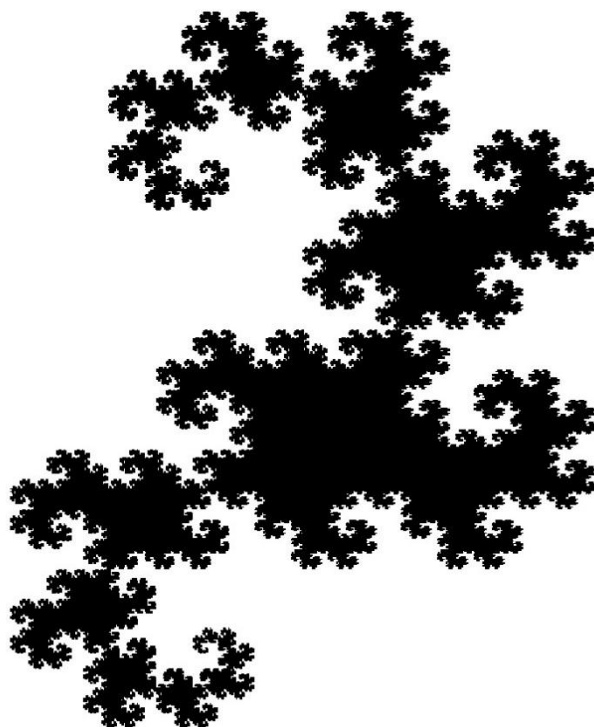
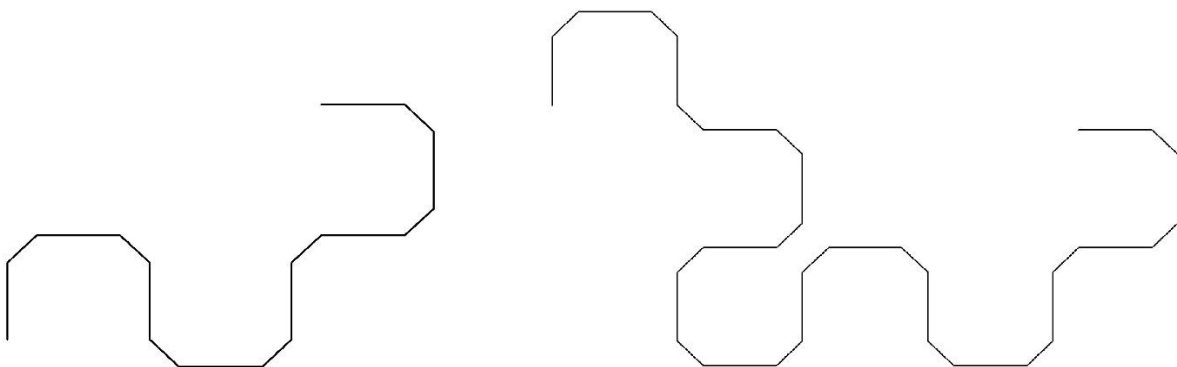
F=

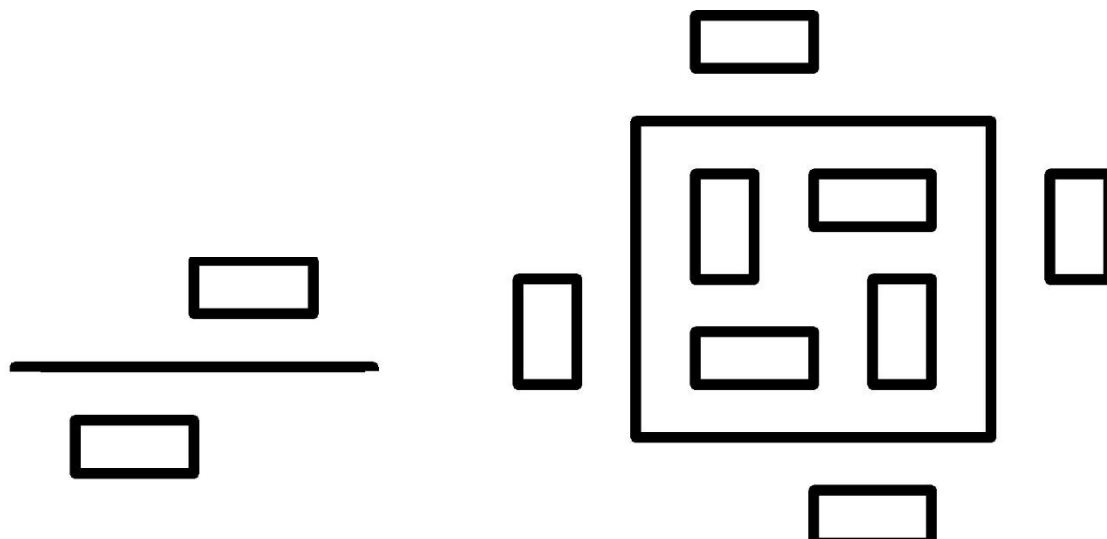


1^{er} cycle : X= F F/2 F



2^{ème} cycle : X=FF FF/2 FF FF/2 FF-FF/2 -FF





angle=90
 axiom=F-F-F-F
 F=F-G+FF-F-F-FG-FF+G-FF+F+FF+FG+FFF
 G=GGGGGG

Les Systèmes de fonctions itérateurs (IFS) de Michael Barnsley

On considère les transformations linéaires du plan complexe, $Z_n \in \mathbb{C} \rightarrow \mathbb{C}$:

C'est-à-dire telles que :

(a) $T(x + y) = T(x) + T(y)$, $\forall x, y \in \mathbb{C}$ (ou $\mathbb{R}^2$ ou $\mathbb{R}^3$)

(b) $T(kx) = k T(x)$, $\forall k \in \mathbb{R}$, $\forall x \in \mathbb{C}$

Et on définit le vecteur $AB = OB - OA = b - a$

Propriétés

$T(0) = 0$ (découle de a et b avec $x=y=0$)

Une transformation linéaire transforme une droite en une droite

En effet, soient les points distincts P, U, et X alignés, c'est-à-dire tels que

$$\forall X \in \text{droite}, \exists k \in \mathbb{R}, PX = k PU$$

Leurs transformés respectent cette affirmation

Des droites parallèles sont transformées en droites parallèles

La propriété : $Q \in \text{droite}_2, \forall X \in \text{droite}_2, \exists k \in \mathbb{R}, QX = k PU$ est conservée

D'ailleurs, les relations d'angles sont conservés par une transformation linéaire

Il existe une transformation inverse

On représente la transformation linéaire par une matrice de transformation des coordonnées

Chaque colonne de la matrice est l'image d'un des vecteurs unité du repère.

Attention : toutes les transformations matricielles ne sont pas linéaires, elles ne le sont que si on peut les inverser (si donc leur déterminant n'est pas nul)

On considère plus particulièrement les transformations suivantes :

- les homothéties $Z_{n+1} = r * Z_n$ dont la matrice de transformation des coordonnées est :

$$\begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix}$$

on se limite aux homothéties contractantes, dont $|r| < 1$, ce qui empêche la divergence vers l'infini, et explique la présence dans l'image résultat d'images plus petites, autosimilaires

- les rotations $Z_{n+1} = e^{i\theta} * Z_n$ - rappel : $e^{i\theta} = \cos(\theta) + i \sin(\theta)$

dont la matrice est :

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \text{ ou } \begin{pmatrix} \cos(\theta) & -\sin(\varphi) \\ \sin(\theta) & \cos(\varphi) \end{pmatrix}$$

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \quad \begin{pmatrix} \cos(\theta) & -\sin(\varphi) \\ \sin(\theta) & \cos(\varphi) \end{pmatrix} \quad \text{le déterminant vaut 1}$$

si $\theta \neq \varphi$ il y a déformation par cisaillement selon les axes

une homothétie ou une rotation par rapport à un autre centre que (0,0) se décompose en une homothétie ou une rotation autour de (0,0) suivie d'une translation, soit :

$$x' = (x-x_0) * \cos(\theta) - (y-y_0) * \sin(\theta) + x_0$$

$$y' = (x-x_0) * \sin(\theta) + (y-y_0) * \cos(\theta) + y_0$$

- les symétries $Z_{n+1} = Z_n' = R_n - i * I_n$ (axe des x)

et $Z_{n+1} = Z_n' = -R_n + i * I_n$ (axe des y)

ont pour matrices respectives :

axe des x axe des y bissectrice axe θ

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} \cos(2\theta) & \sin(2\theta) \\ \sin(2\theta) & -\cos(2\theta) \end{pmatrix} = -\sin(2\theta + \pi)$$

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} \sin(2\theta) & -\cos(2\theta) \\ \cos(2\theta) & \sin(2\theta) \end{pmatrix} = \cos(2\theta + \pi)$$

le déterminant vaut -1 pour les symétries d'axe et $\theta - \varphi = \pm\pi$

(la symétrie par rapport à un point est une simple rotation de 180° : $Z_{n+1} = -Z_n = -R_n - i * I_n$)

-et une petite dernière transformation linéaire : la déformation

$$\begin{pmatrix} r & 0 \\ 0 & s \end{pmatrix}$$

L'application dilate (si $r/s > 1$) ou contracte (si $r/s < 1$) la figure verticalement (si $r > s$) ou horizontalement (si $r < s$), mais conserve le parallélisme

Considérons aussi les projections (qui ne sont pas du tout linéaires, l'axiome (a) n'est pas respecté sur une perpendiculaire, il n'y a pas de transformation inverse car le déterminant est nul, on ne peut pas revenir en arrière, ...)

axe x	axe y	bissectrice		axe θ ($t = \tan(\theta)$)
$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$	$1/(1+t^2)$	$\begin{pmatrix} 1 & t \\ t & t^2 \end{pmatrix} = \begin{pmatrix} \cos^2\theta & \sin\theta \cos\theta \\ \sin\theta \cos\theta & \sin^2\theta \end{pmatrix}$

- on considère enfin la translation $Z_{n+1} = Z_n + Cst$ (qui n'est pas non plus linéaire car $T(0) \neq O$, $T(x+y) = x+y+Cst \neq T(x)+T(y) = x+y+2Cst$)

(e)

(f)

soit dans tous les cas les formules de transformation des coordonnées:

$$\begin{pmatrix} Z_{n+1} \\ Y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} X_n \\ Y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

et donc :

$$\begin{aligned} X_{n+1} &= a * X_n + b * Y_n + e = r * \cos(\theta) * X_n - s * \sin(\varphi) * Y_n + e \\ Y_{n+1} &= c * X_n + d * Y_n + f = r * \sin(\theta) * X_n + s * \cos(\varphi) * Y_n + f \end{aligned}$$

- on considère enfin une probabilité p pour choisir la transformation à appliquer par l'algorithme aléatoire

une transformation est donc complètement définie par le nuplet constitué des nombres réels a, b, c, d, e et f que l'on note : $(1, a, b, c, d, e, f, p)$ que l'on simplifie en

$(2, r, e, f, p)$ pour une homothétie ($a=r, b=0, c=0, d=r$)

ou $(3, \theta, \varphi, r, s, e, f, p)$ pour une rotation homothétie

ce que l'on représente en matrice

$$\begin{pmatrix} r \cos(\theta) & -s \sin(\varphi) \\ r \sin(\theta) & s \cos(\varphi) \end{pmatrix} \text{ soit } \begin{pmatrix} a & b \\ c & d \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

Si $\varphi = \theta$, c'est une rotation ;

si $\varphi = \theta + \pi$ ou $\theta - \pi$, c'est une symétrie par rapport à l'axe $\theta/2$

la première forme étant plus efficace pour les calculs répétés (pas de cosinus et sinus à calculer) les 2 autres formes sont transformées en la 1^{ère} par le programme.

Etant donné la transformation (a, b, c, d, e, f, p) , on peut calculer r, s, θ et φ

$$r = \sqrt{a^2 + c^2}$$

$$s = \sqrt{b^2 + d^2}$$

$$\theta = \arccos(a/r) * \text{sign}(c)$$

$$\varphi = \arccos(d/s) * \text{sign}(-b)$$

Les translations, rotations, réflexions conservent les distances (ce sont des isométries, de rapport 1)

- Une homothétie de rapport $k \neq 1$ multiplie les distances d'un facteur k
- Une similitude (rapport $\neq 1$) est contractante si et seulement si son rapport vérifie $0 < k < 1$.
- une similitude de rapport k composée avec une similitude de rapport k' est une similitude de rapport $k * k'$.
- La composition de similitudes contractantes est une similitude contractante.
- L'ensemble des similitudes (directes ou indirectes) forme un groupe pour la composition.

En composant ces transformations, on en trouve une nouvelle

Attention : comme on a ajouté des transformations non linéaires, ce n'est pas commutatif

Pour construire sans calcul le produit de 2 transformations, il faut savoir que :

- Une translation c'est la composition de deux symétries d'axes parallèles, perpendiculaires à la direction de la symétrie et distants de $d/2$. On peut placer le 1^{er} axe où on veut, perpendiculaire à la direction de la translation. Le 2^{ème} à $d/2$.
- Une rotation c'est la composition de deux symétries d'axe passant par le centre de rotation et faisant un angle $\theta/2$, et on peut choisir n'importe quel 1^{er} axe, passant par le centre de rotation. Le 2^{ème} passant par le centre, à $\theta/2$.
- C'est comme cela que l'on construit la composée de 2 rotations, en prenant pour 2^{ème} axe de la 1^{ère} rotation l'axe passant par les 2 centres de rotation, le 1^{er} à $-\theta_1/2$, et pour le 1^{er} axe de la seconde rotation le même que ce second axe de la 1^{ère} rotation. Les 2 symétries sur ce même axe s'annulent, et on voit bien que le résultat est une rotation d'angle $\theta_1 + \theta_2$
- Procédant de la même manière avec une translation et une rotation où l'inverse, on trouve la rotation résultat.
- Et avec deux translations, ça donne une rotation d'angle double si elles ne sont pas parallèles ou une translation $2d_1 + 2d_2$

On peut aussi faire les calculs, et par exemple pour trouver le point fixe éventuel de la transformation, il faut résoudre :

$$x = ax + by + e$$

$$y = cx + dy + f$$

soit :

si $b \neq 0$ et $(d-1)*(a-1) - b*c \neq 0$,

$$x = (b*f - (d-1)*e) / ((d-1)*(a-1) - b*c)$$

$$\text{et } y = ((1-a)*x - e) / b$$

si $b = 0$, $a \neq 1$, et $d \neq 1$,

$$x = e / (1-a)$$

$$\text{et } y = (c*x + f) / (1-d)$$

si $b = 0$ et $a \neq 1$ et $d = 1$,

$$x = -f/c = e / (1-a), \text{ perpendiculaire à } ox$$

les cas où on ne peut en trouver, c'est que les deux axes sont parallèles d'où le point fixe dans le cas standard :

$$x = e / (1-a)$$

$$\text{et } y = e*c / (1-a) + f$$

Une transformation linéaire est bijective si et seulement si $ad-bc \neq 0$

Elle est contractante si et seulement si

$$a^2 + c^2 < 1 \quad (f(1,0))$$

$$b^2 + d^2 < 1 \quad (f(0,1))$$

$$a^2 + b^2 + c^2 + d^2 - (ad - bc)^2 < 1$$

Un ensemble F de n transformations de ce type définit un système d'itérateurs (IFS)

Selon le théorème de Hutchinson, si les transformations sont contractantes, c'est-à-dire qu'elles rapprochent les points des figures sur lesquelles elles sont appliquées, alors l'IFS défini par $F^n(E)$ possède un ensemble de point attracteur, et il est unique. C'est-à-dire qu'en itérant un grand nombre de fois les transformations, de F , quel que soit le ou les points de départ, on tombe dans l'attracteur.

Un peu de théorie

Nous poserons d'abord quelques définitions et leurs conséquences les plus notables, puis les démonstrations les plus marquantes, à charge au lecteur de démontrer les intermédiaires passés sous silence.

Définitions

Dans un espace topologique E on définit la **distance** entre 2 points $x, y \in E$ telle que

$$d(x, x) = 0$$

$$d(x, y) = 0 \Leftrightarrow x=y$$

$$d(x, y) \leq d(x, z) + d(z, y)$$

et dans C, R^2 ou R^3 , la distance euclidienne remplit ce rôle.

- la **boule ouverte** (dans R^3 , ou le disque dans R^2) de centre x et de rayon r est telle que :

$$B(x, r) = \{ \forall y \in B ; d(y, x) < r \}$$

- la **boule fermée** de centre x et de rayon r est telle que :

$$B(x, r) = \{ \forall y \in B ; d(y, x) \leq r \}.$$

-Une partie non-vide U de E est un **ouvert** si,

$$\forall x \in U, \exists r > 0 \text{ tel que } B(x, r) \subset U.$$

– Un ensemble $F \in C$ est un **fermé** si toute suite $(P_n)_{n \in \mathbb{N}}$ telle que $P_n \in F$ (pour tout n) et $P_n \rightarrow Q$ alors $Q \in F$. (la limite est dans l'ensemble)

– Un ensemble $O \in C$ est un ouvert si $R^2 \setminus O$ est un fermé.

– Un ensemble est **borné** s'il est inclus dans une boule. Autrement dit $E \in C$ est borné s'il existe une boule B de centre o et de rayon $r > 0$ tel que $E \subset B(o, r)$.

- un **voisinage** d'un point est un sous-ensemble contenant un ouvert contenant ce point.

Soit E un espace topologique et a un point de E . Notons $V(a)$ l'ensemble des voisinages de a .

$V(a)$ est un voisinage d'un point a de E s'il existe un réel $r > 0$ tel que la boule ouverte de centre a et de rayon r soit contenue dans V . Autrement dit, V est un voisinage de a si tous les points très proches de a sont dans V .

Nous pouvons alors remarquer que :

1. $a \in E, E \in V(a)$ donc $V(a) \supset \emptyset$
2. $A, B \in V(a) \Rightarrow A \cap B \in V(a)$
3. $A \in V(a)$ et $A \subset B \subset E \Rightarrow B \in V(a)$
4. $\emptyset \notin V(a)$
5. $\forall A \in V(a), a \in A$
6. $\forall A \in V(a) \exists B \subset A \forall b \in B B \in V(b)$

Le voisinage d'un ensemble, c'est l'union du voisinage de tous ses points. En particulier

-Le **ϵ -voisinage** de $E \in H$ est l'ensemble $E(\epsilon)$ des points

$$E(\epsilon) = \{ P \in C, d(P, Q) \leq \epsilon \mid Q \in E \}$$

C'est l'ensemble de tous les points proches de E à ϵ près.

– Un ensemble est **compact** s'il est fermé et borné. il vérifie la propriété de Borel-Lebesgue et il est séparé (propriété de Hausdorff) ce qui signifie que pour deux points quelconques dans E disons x et y , on peut trouver un voisinage U de x et un voisinage V de y tels que $U \cap V$ soit vide. Autrement dit, « entre » deux points on peut toujours en trouver d'autres.

Distance entre deux fractales

Nous définissons H comme l'ensemble des sous-ensembles compacts de C , nous appellerons "fractale" n'importe quel élément de H .

Soit $x \in C$ et $B \in H$, nous définissons la **distance d'un point x à un ensemble B** , et nous la notons $d(x, B)$ comme étant :

$$d(x, B) = \min \{ d(x, y), \forall y \in B \}$$

Si cette distance est nulle, alors $x \in B$ car alors il existe $y \in B$, $d(x, y) = 0$ et donc $x=y \in B$

Soient A et $B \in H$. Nous définissons la **distance de la fractale A à la fractale B** et nous la notons $d(A, B)$ comme étant:

$$d(A, B) = \max \{ d(x, B), \forall x \in A \}$$

Attention, rien ne dit que $d(A, B) = d(B, A)$, et d'ailleurs c'est faux : pensez à un grand ensemble ayant des points lointains et un petit ensemble « proche » de la frontière du grand. Tous les points du petit sont proches du grand, mais les points lointains du grand ne sont pas proches du petit. Et donc le grand ensemble n'est pas proche du petit (on prend le max des distances)

C'est pourquoi nous définissons la "distance de Hausdorff" entre deux fractales A et $B \in H$, et nous la notons $h(A, B)$, comme étant:

$$h(A, B) = \max \{ d(A, B), d(B, A) \}$$

Cette distance a les propriétés suivantes, facilement démontrables ::

$$h(A, B) = h(B, A)$$

$$h(A, A) = d(A, A) = 0$$

$$h(A, B) = 0 \Leftrightarrow A = B$$

$$h(A, B) \leq h(A, C) + h(C, B)$$

transformation **contractante** :

il existe k , $0 \leq k < 1$, tel que pour tout $x, y \in C$, $d(F(x), F(y)) \leq k d(x, y)$

en particulier $d(F(x_1), F(x_0)) \leq k d(x_1, x_0)$

en terme clair, cela signifie que la transformation contracte les distances.

Une transformation contractante transforme un compact non vide en compact non vide

un **IFS** est déterminé par n contractions $f_1, f_2, \dots, f_n$ et on définit l'opérateur d'Hutchinson par $E \in H$, $F(E) = \bigcup_{i=1..n} f_i(E)$

La suite $E_{k+1} = F \{ E_k \}$ converge vers un ensemble unique, l'attracteur K , chaque similitude $f_i(K) \subset K$ donnant lieu à l'**autosimilarité** de l'attracteur K

On peut dire que l'attracteur présente une auto-similarité : un même motif se répète à des échelles différentes et ceci une infinité de fois.

$$K = F(K) = \bigcup f_i(K)$$

Existence de l'attracteur

Il s'agit de prouver qu'il existe un compact non vide $K \in R^2$ qui est invariant par la famille $F = \{ f_i \}$, f_i étant les transformations.

Comme les f_i sont des contractions il existe un disque fermé $D = D(o, r)$ (centré à l'origine o et de rayon r assez grand) tel que $f_i(D) \subset D$ pour tout i

En d'autres termes $F(D) \subset D$. Cela implique $F(F(D)) \subset F(D)$, soit $F^2(D) \subset F(D)$ et par récurrence $F^{k+1}(D) \subset F^k(D)$

L'ensemble de départ E est un fermé borné de $\mathbb{R}^2$ donc un compact. Les applications f_i sont contractantes donc continues, ainsi les $f_i(E)$ sont des compacts et $F(E)$ également, de même pour $F^k(E)$. Donc $F^k(E)$ est une suite de compact non vide, décroissante pour l'inclusion. Alors $K = \bigcap_{k=1, \rightarrow \infty} F^k(E)$ est un ensemble compact non vide.

Autre approche de l'existence de l'attracteur :

Du fait que la fonction F est k -contractante, on a immédiatement, en notant $u_n = F^n(x_0)$

$$\begin{aligned} d(u_{n+1}, u_n) &< k \quad d(u_n, u_{n-1}) < k^2 \quad d(u_{n-1}, u_{n-2}) < \dots \\ \text{donc } d(u_{n+1}, u_n) &< k^n d(x_1, x_0) \end{aligned}$$

Pour prouver la convergence il faut montrer qu'il existe un rang n au delà duquel deux éléments u_n et u_m , peuvent être rendus aussi proche que souhaité. Posons $m = n + p$ (c'est plus contraignant que de vérifier que 2 éléments consécutifs se rapprochent, ils pourraient se rapprocher mais parcourir tout le plan)

$$\begin{aligned} u_{n+p} - u_n &= (u_{n+p} - u_{n+p-1}) + (u_{n+p-1} - u_{n+p-2}) + \dots + (u_{n+1} - u_n) \\ &< d(u_1 - u_0) * (k^n + k^{n+1} + \dots + k^{n+p-1}) \\ &= d(u_1 - u_0) * k^n (1 + k + k^2 + \dots + k^{p-1}) \\ &= d(u_1 - u_0) * (k^n * (1 - k^p) / (1 - k)) \\ &< d(u_1 - u_0) * k^n / (1 - k) \quad \text{car } p \text{ est grand} \end{aligned}$$

La limite vers l'infini de cette expression est 0 car $k < 1$

C'est le critère de Cauchy, qui démontre que la suite converge, puisque on peut rendre la distance entre tous éléments de rang supérieur à n aussi petite qu'on le veut.

Unicité de l'attracteur

Il existe un unique compact non vide $C \in \mathbb{R}^2$ qui est invariant par la famille $F = \{ f_i \}$

Si f_i est contractante, de rapport $k_i < 1$, alors pour deux compacts $K, K' \in \mathbb{R}^2$ nous avons :

$$\begin{aligned} h(f_i(K), f_i(K')) &\leq k_i * h(K, K') \\ \text{donc } h(F(K), F(K')) &\leq \max(k_i) * h(K, K') \\ \max(k_i) &\text{ c'est même le max des } k_i \text{ contractantes, donc strictement } < 1 \end{aligned}$$

(certaines transformations k_i , les isométries, ne sont pas contractantes mais sont de rapport < 1)

Supposons que K et K' soient deux compacts invariants :

$$\begin{aligned} F(K) &= K \text{ et } F(K') = K' \\ \text{Alors } h(K, K') &\leq \max(k_i) * h(K, K') \\ h(K, K') (1 - \max(k_i)) &\leq 0 \end{aligned}$$

Comme $\max(k_i) < 1$ alors $h(K, K') = 0$ et donc $K = K'$

Théorème du collage, pour évaluer la distance à l'attracteur

$$\begin{aligned} d(E, K) &\leq d(E, F(E)) + d(F(E), K) \quad \text{par l'inégalité triangulaire.} \\ &= d(E, F(E)) + d(F(E), F(K)) \quad \text{car } F(K) = K \\ &\leq d(E, F(E)) + k * d(E, K) \quad \text{car } f \text{ est une famille de contractions.} \\ \text{D'où } d(E, K) &\leq d(E, F(E)) * 1/(1 - k) \end{aligned}$$

Donc on dispose d'une approximation de la distance à l'attracteur.

Toute figure est-elle l'attracteur d'un système d'itérateurs ?

Oui et il y a une infinité de solutions.

Il s'agit de prouver que pour n'importe quelle partie compacte E , on peut trouver une famille de contractions dont l'attracteur K approxime E .

Soit E une partie non vide et compacte de $\mathbb{R}^2$. Pour chaque $\varepsilon > 0$ il existe une famille $F = \{ f_i \}$ de contractions ayant pour attracteur un ensemble K tel que

$$d(E, K) < \varepsilon$$

Démonstration : Soit $\varepsilon > 0$. Nous recouvrons le compact E par un nombre fini de disques D_i , chacun de rayon $\varepsilon/2$ et dont les centres appartiennent à E .

L'union de ces disques est un ensemble des points situés dans E ou à une distance inférieure à $\varepsilon/2$ de E , c'est un $\varepsilon/2$ -voisinage de E .

Pour chaque disque D_i nous construisons une contraction f_i , de rapport $k_i < 1/2$, telle que $f_i(E) \subset D_i$. (par exemple une homothétie de rapport $\varepsilon/2L$ au centre du disque, L étant la plus grande dimension de E).

Nous allons considérer la distance entre E et $F(E)$.

on a vu que $d(E, K) \leq d(E, F(E)) * 1/(1-k)$ avec ici $k < 1/2$, le rapport de contraction

Comme $f_i(E) \subset D_i$ alors $F(E) = \bigcup f_i(E) \subset \bigcup D_i$

et $d(E, F(E)) \leq d(E, \bigcup D_i) \leq \varepsilon/2$, le rayon des disques

donc $d(E, K) \leq \varepsilon$

ce qui veut dire que l'on peut approcher l'image E aussi près que l'on veut, il suffit de faire des disques plus petits.

Certes, l'IFS obtenu ainsi n'est pas « le meilleur », en nombre d'itérateurs, mais il fonctionne.

Question : ne contenant que des homothéties de rapport positif, et des translations, je ne vois pas comment il rend compte des rotations et des inversions que l'on verrait à l'œil nu dans l'attracteur de départ.

Revenons à nos moutons

Il y a deux méthodes pour dessiner l'attracteur à partir de l'ensemble F des fonctions f_n :

- L'algorithme aléatoire (cliquer sur «aléatoire») consiste à prendre un point Z et à lui appliquer une des transformations, choisie au hasard, puis à recommencer avec le point résultat. il suffit de ne pas afficher les 100 premiers points, le temps que la convergence se fasse, puis d'afficher tous les autres.

C'est ce qu'on appelle « Le jeu du chaos » correspondant à l'IFS, il est donné par les mêmes contractions et leurs probabilités d'utilisation (strictement positives) suivantes :

$p_1, p_2, \dots, p_n$ où $\sum p_i = 1$.

Ce procédé est appelé Système Aléatoire de Fonctions Itérées (alors que l'IFS correspondant est déterministe).

Soit z_0 un point fixe de l'une des n contractions, alors tous les points de la suite $z_0, z_1 = f_{s_1}(z_0), z_2 = f_{s_2}(z_1), \dots$ sont dans l'attracteur A . De plus la suite $z_0, z_1, \dots, z_n$ représente l'attracteur A

Attention : La suite des points ne converge pas vers un point fixe, c'est l'ensemble des points qui converge vers une image fixe. Il faut remarquer que quel que soit l'algorithme de construction aléatoire ou l'algorithme déterministe, les points ne décrivent pas un cycle périodique, sinon l'ensemble des points de l'attracteur serait réduit à ces points du cycle et ne serait pas infini et ne serait pas compact. Dans l'algorithme de construction déterministe, un même point n'est pas atteint deux fois sinon il serait le début d'un cycle. En fait le point décrit un pseudo cycle puisqu'il se rapproche du point fixe d'une des transformations puis s'en éloigne en se rapprochant du point fixe de la transformation suivante, etc ...

On peut choisir un affichage progressif (un groupe de points par 500 ms) ou rapide (tous les points calculés d'un coup)

- L'algorithme déterministe (cliquer sur «déterminé») consiste à partir d'un ensemble de points (un segment, un triangle, un carré, les points fixes, un point fixe, une photo....) et à lui appliquer un cycle constitué d'une fois chacune des transformations, puis à recommencer avec l'ensemble des points images ainsi obtenus par l'application des n transformations.

On applique donc l'opérateur d'Hutchinson (c'est-à-dire toutes les transformations successivement)

le résultat de la première itération sera une image constituée de n points :

$$E_1 = \{f_1(z_0), \dots, f_n(z_0)\}$$

Après la seconde itération on obtient n^2 points et ainsi de suite.

la ligne joignant les points fixes est un bon ensemble de départ pour l'algorithme déterministe car il converge plus vite. Mais toutes figure fera l'affaire, y compris une photo de Ramses II, ou la photocopie d'une page de la chartreuse de Parme, puisque l'application est convergente quel que soit l'ensemble (ou le point) de départ.

On peut choisir un affichage «manuel», cycle par cycle, à chaque appui sur le bouton «cycle suivant» ou automatique (un cycle toutes les 500ms pendant une vingtaine de cycles).

Choisir une photo comme structure de départ implique le passage à l'algorithme déterminé, et montre que quelque soit l'image de départ, on parvient au même résultat

Dans le cas de l'affichage «manuel», pour bien comprendre le rôle de chaque transformation, on peut les appliquer une par une, mais dans ce cas, comme on n'applique qu'une seule transformation à l'image d'une seule transformation, contractante, on ne peut pas atteindre l'attracteur.

Il y a plusieurs cas de convergence :

- vers une surface remplie de points

En choisissant bien l'ensemble de départ, l'algorithme déterministe permet alors de distinguer la convergence vers une courbe qui, par ses circonvolutions de plus en plus serrées, remplit la surface. C'est le cas des dragons de Heighway, Sierpinski, Levy, du twindragon et de leurs dérivés, des courbes de Hilbert, Peano, Gosper, ...

L'algorithme déterministe montre alors à chaque cycle l'évolution de la courbe.

- vers une courbe, continue mais non dérivable en tous points, de longueur infinie, mais de surface enclose finie.

c'est le cas de la courbe et du flocon de von Koch et de leurs dérivés

- vers un compact dit «ouvert», contenant des trous

Les images de chaque transformations sont disjointes.

C'est le cas du triangle et du tapis de Sierpinski, pour lequel on place des homothéties à chaque sommet d'un polygone et on en choisit une différente de la précédente à chaque itération.

- vers une spirale autour d'un point fixe, par une rotation d'angle faible et de rapport voisin de 1 et une homothétie de fort rapport pour constituer une réduction de l'ensemble près d'un 2ème point fixe.

Le zoom (souris gauche) recentre sur le point choisit et effectue un agrandissement

on peut modifier les transformations de l'IFS, en modifiant ses transformations en en rajoutant ou retirant

on peut aussi définir un ensemble de départ et appliquer successivement l'une ou l'autre des transformations du système sur la structure de départ et ses images successives.

Question subsidiaire

Comment choisir les transformations d'un IFS devant converger vers un attracteur donné ?

L'idée est d'utiliser, en l'améliorant, la méthode de la démonstration de l'existence d'un IFS pour toute image.

On ne va pas dessiner des cercles mais des patatoïdes approchant au plus près l'image de départ et ses parties les plus remarquables. Il s'agit de repérer les auto-similarités et d'entourer pour chacune leur élément fondateur le plus gros, qui se reproduit à l'infini par une rotation ou une homothétie en un centre, ce centre induisant une translation.

L'idée est de faire une rotation, homothétie translation de l'ensemble en le faisant alors coïncider avec chacun des patatoïdes repérés, sans que ca se recouvre trop, et sans oublier une zone.

Pythagore

$$f_1(x) = \begin{bmatrix} \cos^2(\alpha) & -\cos(\alpha)\sin(\alpha) \\ \cos(\alpha)\sin(\alpha) & \cos^2(\alpha) \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$f_2(x) = \begin{bmatrix} \sin^2(\alpha) & \cos(\alpha)\sin(\alpha) \\ -\cos(\alpha)\sin(\alpha) & \sin^2(\alpha) \end{bmatrix} x + \begin{bmatrix} \cos^2(\alpha) \\ 1 + \cos(\alpha)\sin(\alpha) \end{bmatrix}$$

$$f_3(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x$$

La fougère de Barnsley



Aucun trait ne se génère tout seul, par exemple pour la fougère de Barnsley, il faut penser aux feuilles et aux branches et à ce qui va créer le tronc. D'ailleurs ce qui va créer le tronc servira aussi pour les branches elles-mêmes : une rotation homothétie de 72° et de rapport $\frac{1}{4}$ centrée en centre bas droite pour la 1^{ère} branche de droite, une de -72° et de rapport $\frac{1}{4}$ centrée en centre bas gauche pour la 1^{ère} de gauche, une similitude avec un petit angle de $0,05^\circ$ centrée en haut à droite pour avoir un tronc légèrement incurvé, et des branches incurvées également puisque chaque branche dérive du tronc. Chacune de ces 3 similitudes envoie l'ensemble de l'image l'une vers la branche de droite, l'autre vers la branche de gauche et la dernière en spirale vers celles du haut. Il en faut une dernière compressant tout fortement vers le 1^{er} segment du tronc lui-même, sans les branches ($x'=cst=0$, $y'=y/10$).

Supprimer une similitude supprime une partie de l'image :

Supprimer la 1^{ère} supprime la branche de droite, et donc toutes les branches de droite

Supprimer la 2^{ème} supprime la branche de gauche, et donc toutes les branches de gauche

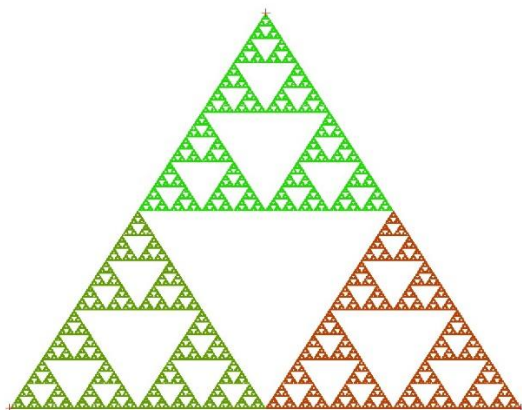
Supprimer la 3^{ème} ne laisse que les 2 branches du bas, mais réduites à la branche sans les feuilles

Supprimer la 4^{ème} supprime le tronc.

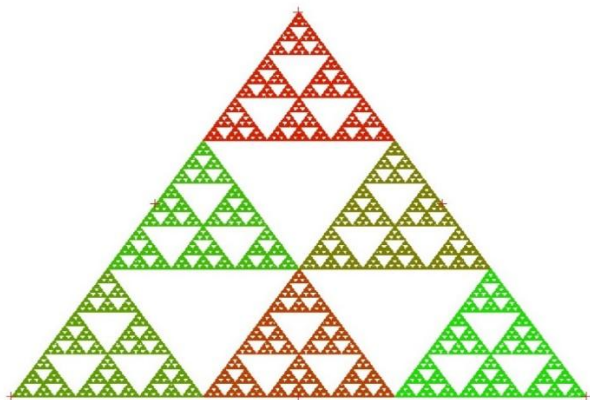
triangle de Sierpinski

on place 3 homothéties de rapport $\frac{1}{2}$ aux 3 sommets d'un triangle

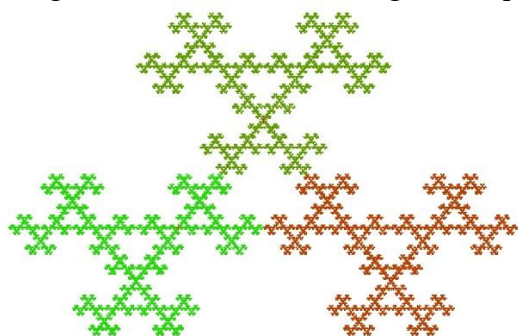
cela compresse l'image vers les 3 sommets et laisse un trou au centre



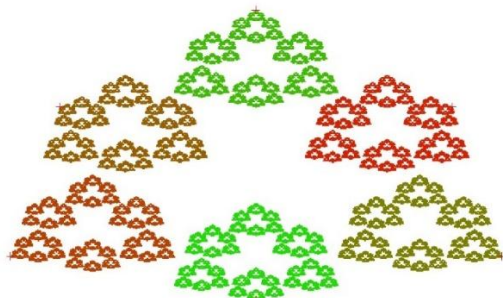
si on prend un rapport de $1/3$, le trou s'agrandit et si on ajoute 3 homothéties au milieu des 3 cotés, le trou se comble



on peut mettre une homothétie de rapport négatif (ou ajouter une rotation de 180°) ça envoie l'image vers l'extérieur du triangle de départ

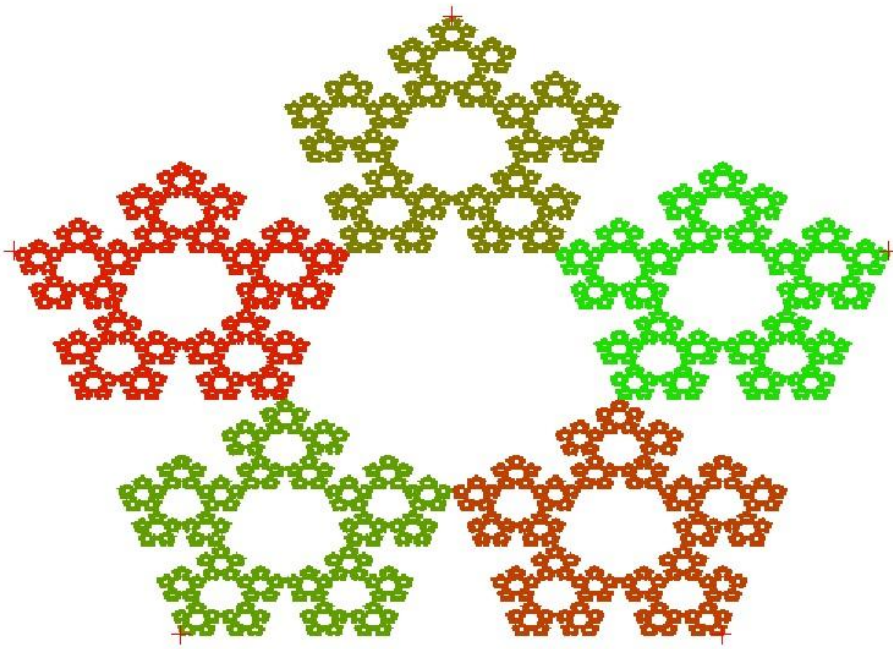


si on déplace légèrement les points fixes des cotés vers l'extérieur, les triangles se disjoignent



Au lieu de cela si on passe toutes les homothéties à $1/4$ au lieu de $1/3$

on peut faire pareil avec un carré ou un pentagone



cas des courbes couvrant le plan et les dragons

on part d'un segment et de rotations homothétiques de même rapport, organisées de sorte que les segments résultats du 1^{er} cycle soient contigus par un sommet. Etant contigus, ils le resteront à chaque cycle et formeront une courbe. Le principe consiste à construire le motif en faisant tourner et translater le segment (la translation définit le point fixe – s'il n'y a pas de translation, c'est (0,0)). Le segment de départ utilisé est (0,0) (1,0) et il faut procéder évidemment par l'algorithme déterministe et au pas à pas, et pas par l'algorithme aléatoire qui élimine les premiers points et qui ne montrerait que l'attracteur dans son état final.

Hilbert

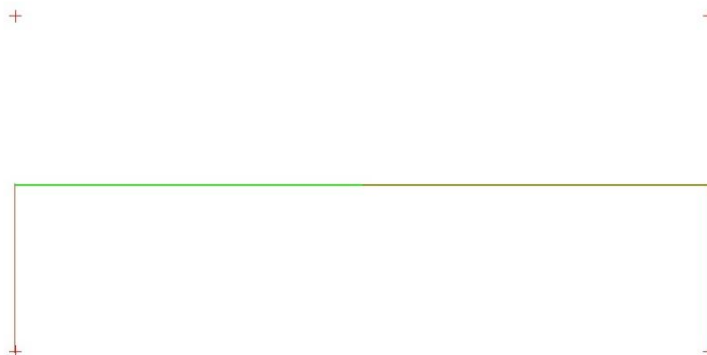
2 rotations 90 et 2 homothéties

S_0 la symétrie par rapport à la droite $y = x$

S_1 la translation de vecteur (0, 1)

S_2 la translation de vecteur (1,1)

S_3 la symétrie par rapport à la droite $y = -x$, composée avec la translation de vecteur (2,1).



Cas des quadra

Un segment 1 rotations de 90 au centre

Cas des spirales

Il faut une rotation d'angle faible, (20°) et de rapport voisin de 0,9 pour faire la spirale et une homothétie de rapport faible (0,2) pour faire le motif à partir de l'ensemble on place les centres n'importe où (à priori le centre de la spirale en O) et le 2^{ème} centre là ou va se construire le 1^{er} motif

Pour dessiner les traits il faut compresser l'ensemble en trait près du 2^{ème} centre avec donc un 3^{ème} point fixe où le trait se dessine pour le 1^{er} motif, si les traits sont mal positionné ou le petit motif mal cadré, on ajuste la translation correspondante.

P(P_x, P_y) étant le 3^{ème} point fixe

Pour faire un trait horizontal

$f(x) = P_x + k(x - P_x)$ avec k faible (0,2)

$f(y) = P_y$

$$\begin{pmatrix} k & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} P_x - kP_x + e \\ P_y \end{pmatrix}$$

xmin dépend de l'échelle choisie pour le dessin, ce n'est pas la taille de l'écran mais un nombre du genre -2, limite des valeurs possibles qu'on se donne pour spécifier les points fixes.

$x - x_{\min} + k(x - P_x)$

$x - x_{\min} + x_p$

$x_p + k(x_{\min} - x_p) + k x_p$

Pour faire un trait vertical

$f(x) = P_x$

$f(y) = P_y + k(y - P_y)$

soit une similitude contractante T de $C \rightarrow C$ définie pour tout $z \in C$

il est souvent utile de la représenter par

$T(z) := p + f(z - p)$ où $p \in C$ est le point fixe de T, et $f = k(z - p) e^{i\theta} \in C$ avec $k \in (0, 1)$ le facteur de contraction de T, et $\theta \in (-\pi, \pi)$ l'angle de rotation de T.

Ainsi on retrouve de manière claire une Translation au point fixe, une rotation et une homothétie contractante de centre le point fixe

Attracteurs étranges

A partir des coordonnées d'un point initial, un attracteur étrange se construit en itérant un grand nombre de fois une formule sur les coordonnées successives obtenues.

soit, en 4 dimensions :

$$(X_{n+1}, Y_{n+1}, Z_{n+1}, W_{n+1}) = f(X_n, Y_n, Z_n, W_n)$$

L'ensemble des point images, s'il ne diverge pas vers l'infini mais couvre seulement une région limitée, est appelé l'attracteur du système de formules.

l'attracteur peut être :

- un point fixe
- un cycle de période plus ou moins grande
- un ensemble de points, le plus souvent chaotique, et parfois avec des symétries et rotations, mais pas de translations

Poincaré a montré qu'un ensemble limite d'un système d'équations différentielles à 2 dimensions, qui ne contient pas de point d'équilibre est une orbite périodique, et donc que tout système macroscopique repasse une infinité de fois aussi près que l'on veut de son état initial, après un cycle qui peut être plus ou moins long.

Les premiers 100 points sont ignorés, ils servent à converger vers le bassin d'attraction de l'attracteur.

Les premiers 1000 points ne sont pas affichés non plus, ils servent à calculer les bornes de l'attracteur et à définir les facteurs d'échelle pour l'affichage, en supposant que ces 1000 premiers points parcourent tout l'attracteur.

Parfois, l'attracteur continue à grossir après les 1000 points, (Hopalong, Martin) il est alors nécessaire d'utiliser le bouton «zoom out» pour recadrer l'attracteur

L'attracteur peut s'afficher en mode progressif (1000 points par seconde) ou en mode rapide en mode progressif, on peut choisir un passage de cycle à cycle manuel ou automatique, c'est à dire afficher 1000 points par 1000 points

Lorsque l'attracteur est affiché, on peut faire calculer encore autant de cycles de 1000 points que nécessaire

On peut aussi visualiser les itérations successives du point et en suivre le mouvement.

Les attracteurs se classent par leur nombre de variable (1 à 4), par le degré des polynômes (2 à 5), par les fonctions utilisées et par le nom de leur découvreur.

On distingue les attracteurs standards des attracteurs différentiels, pour lesquels les fonctions portent sur les coordonnées et calculent les dérivées.

On calcule alors d'abord les dérivées puis les nouvelles coordonnées par :

$$X_{n+1} = X_n + \text{epsilon} * X'$$

Certains attracteurs sont non polynomiaux, et font intervenir les fonctions abs, sinus, cosinus, sqrt, ...

Voir le livre : «Strange Attractors: Creating Patterns in Chaos» par Julien C. Sprott, dont ce programme est issu en ce qui concerne les attracteurs polynomiaux.

En choisissant judicieusement les fonctions parmi les transformations du plan, on peut faire en sorte que l'attracteur ait des axes de symétries et des rotations

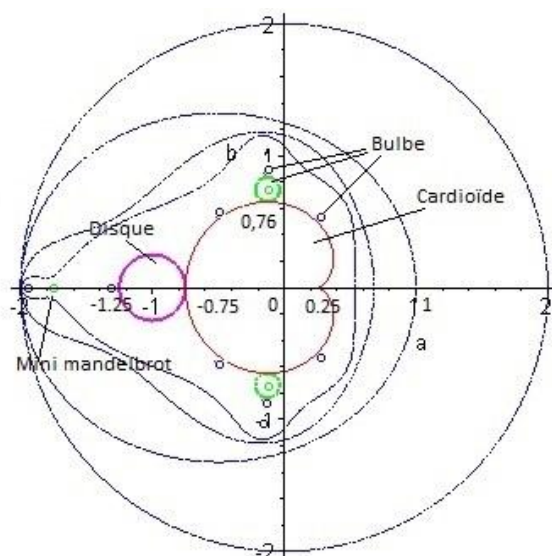
voir le livre : «Symmetry in chaos» de Michael Field et Martin Golubitsky dont ce programme utilise les algorithmes.

Mandelbrot

L'ensemble de Mandelbrot est l'ensemble des points C du plan complexe tels que la suite

$$Z_{n+1} = Z_n^2 + C \quad \text{avec } Z_0 = (0,0)$$

ne diverge pas vers l'infini, il est fermé : $f(Z)$ et sa limite appartient à l'ensemble, il est connexe, c'est à dire que tous ses points sont reliés, il est symétrique par rapport à l'axe des x , il est inclus dans le disque fermé centré en 0 de rayon 2.



pourquoi $Z^2 + C$?

Si on étudie $az^2 + bz + c$, par un changement d'axe on peut se limiter à étudier $a(z^2 + b/a z + c/a)$ soit $(z + b/2a)^2 - (b^2 - 4ac)/4a^2$ soit $Z^2 + C$ avec le changement de variable $Z = z + b/2a$

L'ensemble de Julia est, pour un C donné, l'ensemble des points Z_0 du plan complexe tels que la suite $Z_{n+1} = Z_n^2 + C$ ne diverge pas vers l'infini.

il est fermé, $f(z)$ et sa limite appartient à l'ensemble.

il est symétrique par rapport au point $(0,0)$ car deux points opposés z et $-z$, ont la même trajectoire par $z^2 + C$

il est connexe et contient $(0,0)$ si et seulement si C est dans l'ensemble de Mandelbrot.

il est totalement non connexe et ne contient pas $(0,0)$ si et seulement si C n'est pas dans l'ensemble de Mandelbrot.

Quand il n'est pas connexe, c'est toujours un nuage de points, une «poussière de Cantor» (quand C est près de la frontière il semble y avoir des composants connexes, mais cela est dû à l'imprécision des calculs).

$Julia(C)$ est le symétrique de l'ensemble $Julia(\text{conj}(C))$ par rapport à l'axe des abscisses.

si C est sur l'axe des x le Julia est symétrique par rapport à l'axe des x et par rapport à l'axe des y , il est inclus dans le disque fermé centré en 0 de rayon $\max(2, |C|)$

Pour $C=0$, $Julia(0)$ est le disque fermé centré en 0 de rayon 1.

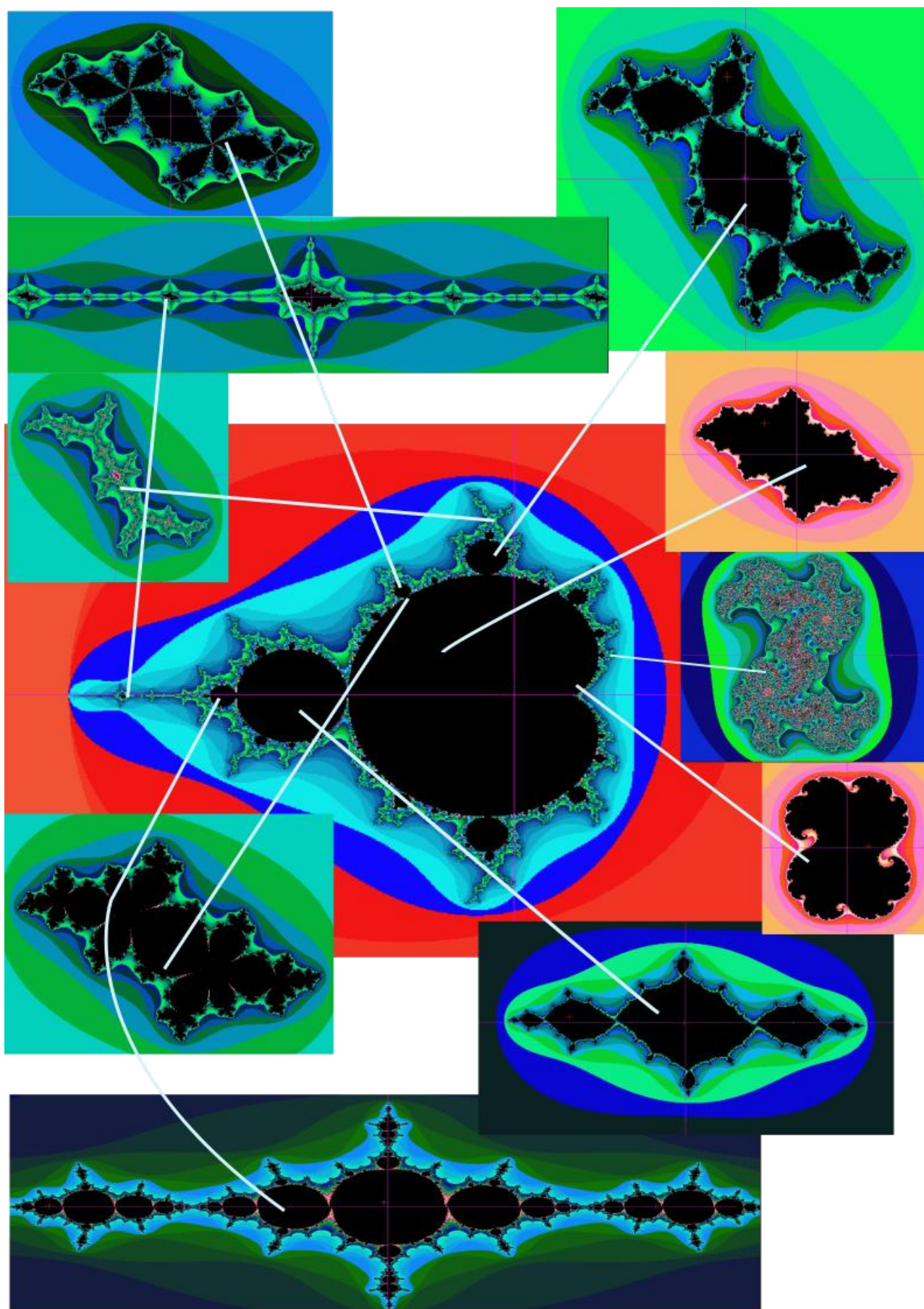
Pour $C=-2$, $Julia(-2)$ est le segment $[-2,0]$

Pour C dans la cardioïde c'est une patatoïde centrée en $(0,0)$

Pour C dans les disques tangents c'est une forme ressemblant à un lapin.

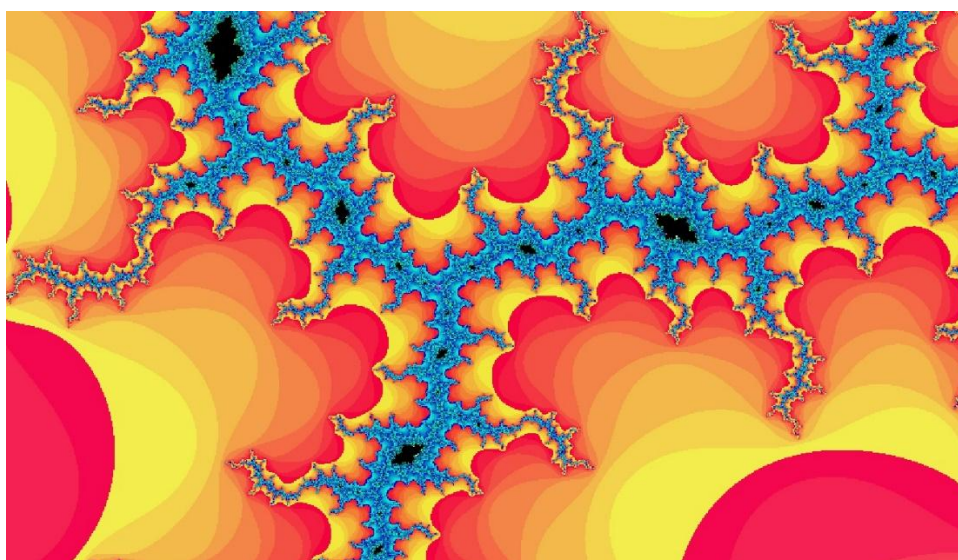
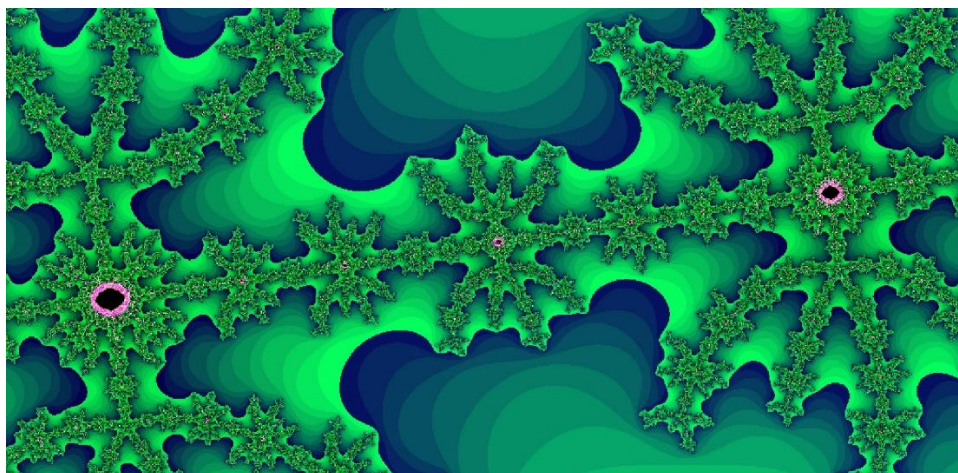
Pour C dans les disques voisins de l'axe des x à gauche de la cardioïde, le lapin s'aplatit et c'est une forme ressemblant à un avion vu de face.

Pour C traversant le bord du Mandelbrot, ça s'amincit en filament, c'est une «dentrite» qui ne contient pas de points intérieurs et qui n'a ni point fixe ni cycle, tout converge à l'infini

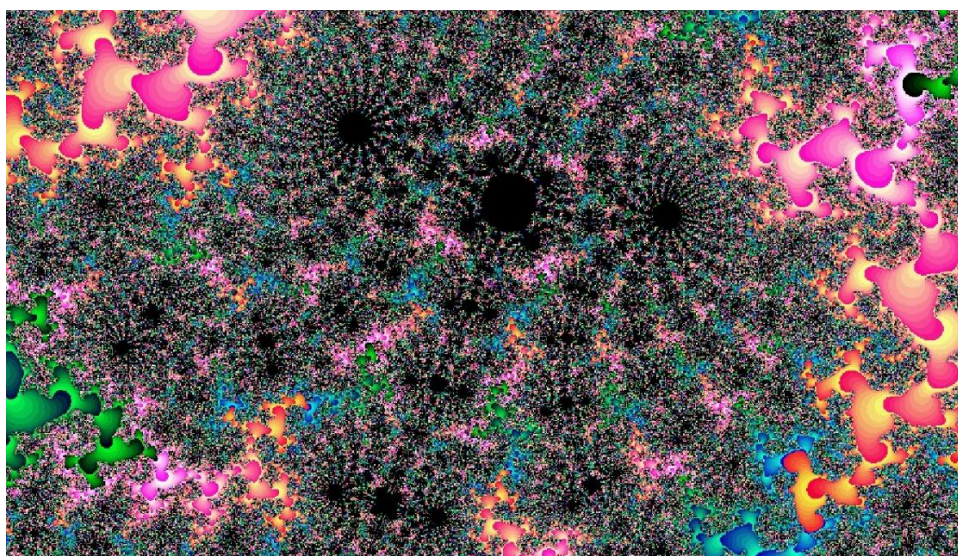


Forme des julia selon la position du point C

En zoomant sur une dentrite d'un julia, on trouve des points entre les points qui semblaient isolés :



Et là on voit le julia près d'éclater en un nuage de points :



Dans le programme joint, le zoom (souris gauche) recentre sur le point choisit et effectue un agrandissement.

Le zoom dynamique (souris du milieu) recentre et effectue un agrandissement toutes les secondes.

zoomer dans l'ensemble de Mandelbrot définit le point C

Le traveling consiste à faire déplacer le point C verticalement ou horizontalement et à redessiner l'ensemble de Julia correspondant. la vitesse du déplacement est réglable, il faut la choisir très petite lorsque le point C est au voisinage de la frontière du Mandelbrot car c'est là que l'ensemble de Julia change rapidement de forme pour passer de connexe à non connexe. Pendant le traveling, on peut recentrer et zoomer à l'aide de souris gauche et de unZoom

L'ensemble de Mandelbrot contient une multitude de composants connectés, ayant chacun un julia où le cycle des points est périodique et l'étude des points fixes et des cycles va permettre de préciser la géométrie des différents bulbes constituant le mandelbrot.

Dans chacun des composants, le point O est pré périodique en général, mais il y a une valeur de C pour laquelle il est dans le cycle périodique.

Points fixes

Pour trouver les points fixes, il faut résoudre

$$f(Z)=Z$$

$$\text{donc } Z^2+C=Z \text{ soit } Z^2-Z+C=0$$

il y a deux solutions pour chaque Julia:

$$(1 \pm \sqrt{1-4C})/2$$

la dérivée en ces points fixes donne leur nature : attractif (si $f' < 1$), répulsif (si $f' > 1$) ou neutre (si $f' = 1$)

Au point neutre, $f'(z)=2z$, comme $|f'(z)|=1$ on pose $f'(z)=e^{it}=2z$

soit, au point fixe neutre, $e^{it}=1 \pm \sqrt{1-4C}$

$$\text{et } 1-4C=(e^{it}-1)^2$$

$$1-4C=e^{it2}-2e^{it}+1$$

$$\begin{aligned} C &= e^{it}(-e^{it}+2)/4 = e^{it}/2(1-e^{it}/2) = e^{it}/2(1-e^{it}/2-e^{-it}/2+e^{-it}/2) = e^{it}/2(1-\cos(t)+e^{-it}/2) \\ &= e^{it}/2(1-\cos(t))+e^{it}/2 e^{-it}/2 \\ &= e^{it}/2(1-\cos(t))+1/4 \end{aligned}$$

d'où l'ensemble des C donnant un point fixe neutre est : $C = e^{it}/2 (1 - \cos t) + 1/4$

C'est la cardioïde principale de l'ensemble de Mandelbrot.

Si et seulement si C est dans la cardioïde les suites du Julia convergent vers un point fixe en suivant un pseudo cycle influencé par le cycle du ou des disques tangents les plus proches.

Si au point fixe t est un irrationnel de π , il apparait un disque de Siegel autour du point fixe, et les points de ce disque orbitent dessus.

Si t est un rationnel de π , de la forme $2\pi p/q$ ($p < q$), il apparait tangent au mandelbrot un bulbe ayant q antennes, la p^{ème} dans le sens trigonométrique est la plus petite et dans l'autre sens c'est la plus grosse.

Les bulbes de la cardioïde

Il y a q-1 bulbes de période q, on a vu qu'ils sont tangents à la cardioïde en

$$\exp(2i\pi p/q)/2 * (1-\exp(2i\pi p/q)/2)$$

et le rayon du bulbe est $\sin(\pi p/q)/q^2$, q étant la période.

Le bulbe $(q-p)/q$ est le conjugué du bulbe p/q par rapport à l'axe ox
 Entre deux bulbes p/q et p'/q' de la cardioïde, le plus gros bulbe est $(p+p')/(q+q')$ (addition de Farey)

Le Julia des points C d'un bulbe p/q contiennent p lobes autour du point fixe et le cycle y passe d'un lobe à l'autre en sautant p lobes à chaque pas, c'est-à-dire qu'il effectue une rotation de $2\pi p/q$ autour du point fixe.

Sur les bulbes qui ne sont pas tangents à la cardioïde, la période p suit la même règle mais le saut d'un lobe à l'autre est plus complexe, car le cycle utilise d'autres lobes.

Cycles

Quand il n'y a pas de point fixe attractif, il y a des cycles

Pour trouver les n -cycles il faut résoudre

$$f^n(z)=z \text{ pour les trouver}$$

et $f^n(C)=0$ pour trouver les C centre de ces bulbes (je n'ai pas compris pourquoi le fait que 0 est pré-périodique est important, certes $f(0)=C$)

C'est un polynôme de degré 2^n , il y a donc au plus (en excluant les points fixes) 2^n composants qui ont un cycle de période n

En ces points des cycles $f(Z_i)=Z_{i+1}$, $f^n(Z_i)=Z_i$, et les dérivées nièmes de $f^n(z)$ sont toutes égales à

$$2^n * z_1 * z_2 * \dots * z_n.$$

2-cycles - il faut donc résoudre :

$$f(f(z))=z$$

$$\text{donc } (z^2+C)^2+C=z$$

or les points fixes précédents sont forcément solution, donc cela se transforme en

$$(z^2-z+C)*(z^2+z+C+1)=0 \text{ et } z^2+z+C+1=0$$

d'où les deux points 2-cycle solutions de $z^2+z+C+1=0$

$$(-1 \pm \sqrt{-3 - 4C}) / 2$$

de plus la dérivée est de module $4z(z^2+C)=-4z(z+1)=4(z^2+z)=4(C+1)=e^{it}$

$$\text{donc } C = -1 + e^{it}/4$$

C'est le cercle de centre -1, de rayon 1/4 et tangent à la cardioïde en -3/4.

Si et seulement si le point C est dans le disque principal tangent à la cardioïde les suites du Julia convergent vers un cycle à deux points.

Et effectivement en résolvant $c^2+c=0$, on trouve $c=-1$, le centre du disque

dans toutes les autres zones que l'on trouve en zoomant il apparaît aussi des cycles. Pour visualiser les cycles, cliquer droit dans les Julia pour suivre l'itération de la fonction à partir de ce point. Les cycles passent d'une antenne à l'autre, dans un sens ou dans l'autre en sautant p antennes à chaque pas.

Les composants n -cycles sont, en particulier, tous les disques tangents à la cardioïde : le disque tangent à gauche à la cardioïde est un 2-cycle, le disque supérieur est un 3-cycle. à partir du disque supérieur, en tournant vers la droite on trouve dans les disques des cycles 3,4,5,6,...

entre deux bulbes n et $n+1$, le plus gros est $2n+1$

entre deux disques consécutifs de périodes p et q , le plus gros est de période $p+q$. on trouve donc vers la gauche des cycles 3,5,7,9...les bulbes tangents à un n -bulbe ont une période q qui double et suivent ces règles, p et q étant premiers entre eux.

Cette règle s'applique à tout disque ayant des disques tangents, en multipliant q par la période du disque père. les julia correspondants sont des «lapins de Douady ayant chacun des pattes à n branches

3-cycles

$$((z^2+c)^2+c)^2+c=z$$

Sachant que les solutions déjà étudiées de $z^2-z+C=0$ et $z^2+z+C+1=0$ sont solutions de cette équation, il reste :

$$C^3+2C^2+C+1=0$$

$$C=-1.9408 \text{ (au bout de l'antenne à gauche, proche de -2)}$$

$$C=-0.1226+0.7449i$$

$$(c^2+c)^2+c=0 \text{ donne}$$

$$C=-1.7549, \text{ le centre du mini-mandelbrot visible à l'avant du mandelbrot, } 1/3$$

$$\text{et } C=-0.1226\pm 0.7449 i, \text{ le centre des bulbes } 1/3 \text{ et } 2/3$$

4-cycles

$$((c^2+c)^2+c)^2+c=0$$

$$C=-1.3107 \text{ (centre du 2}^{\text{ème}} \text{ disque à gauche)}$$

$$C=-1.9408, \text{ le satellite à l'extrême gauche}$$

$$C=0.282\pm 0.53 i \text{ (centre du 2}^{\text{ème}} \text{ bulbe en haut à droite de la cardioïde)}$$

$$C=-0.1565\pm 1.0323 i \text{ (mini mandelbrot au sud du bulbe } 2/3)$$

5-cycles

Il y a 15 centres, de plus en plus proches de la frontière du mandelbrot

Revenons en à 0 pré-périodique (je ne sais toujours pas pourquoi, mais je le constate)

$$0 \rightarrow C \rightarrow C^2+C \rightarrow (C^2+C)^2+C \rightarrow ((C^2+C)^2+C)^2+C$$

Ce chemin peut reboucler sur 0, sur C, sur C^2+C , ...

Pour un cycle 3 ; $(C^2+C)^2+C=C^2+C$ d'où $c=-2$, l'extrémité de la grosse antenne

Pour un cycle 4 : $((C^2+C)^2+C)^2+C = (C^2+C)^2+C$

$$C=-1.54369 \quad \text{centre du mini mandelbrot à l'avant}$$

$$C=-0.22816\pm 1.11514 i$$

$$\text{Ou : } ((C^2+C)^2+C)^2+C = C^2+C$$

$$C=\pm i \quad \text{mini mandelbrot au bout de l'antenne des bulbes } 1/3 \text{ et } 2/3$$

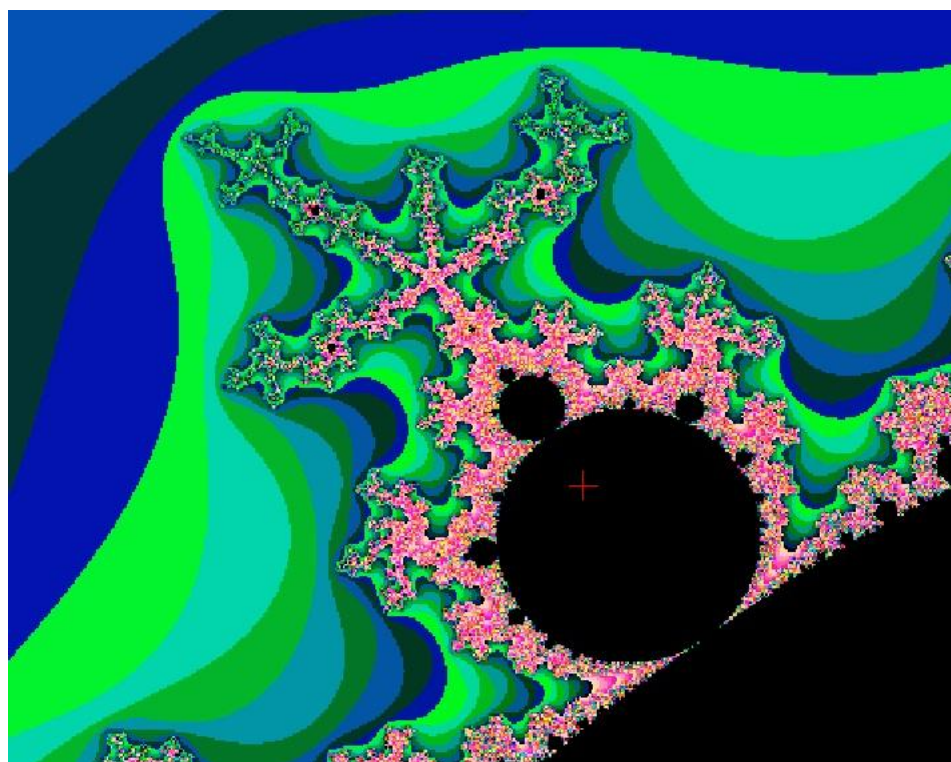
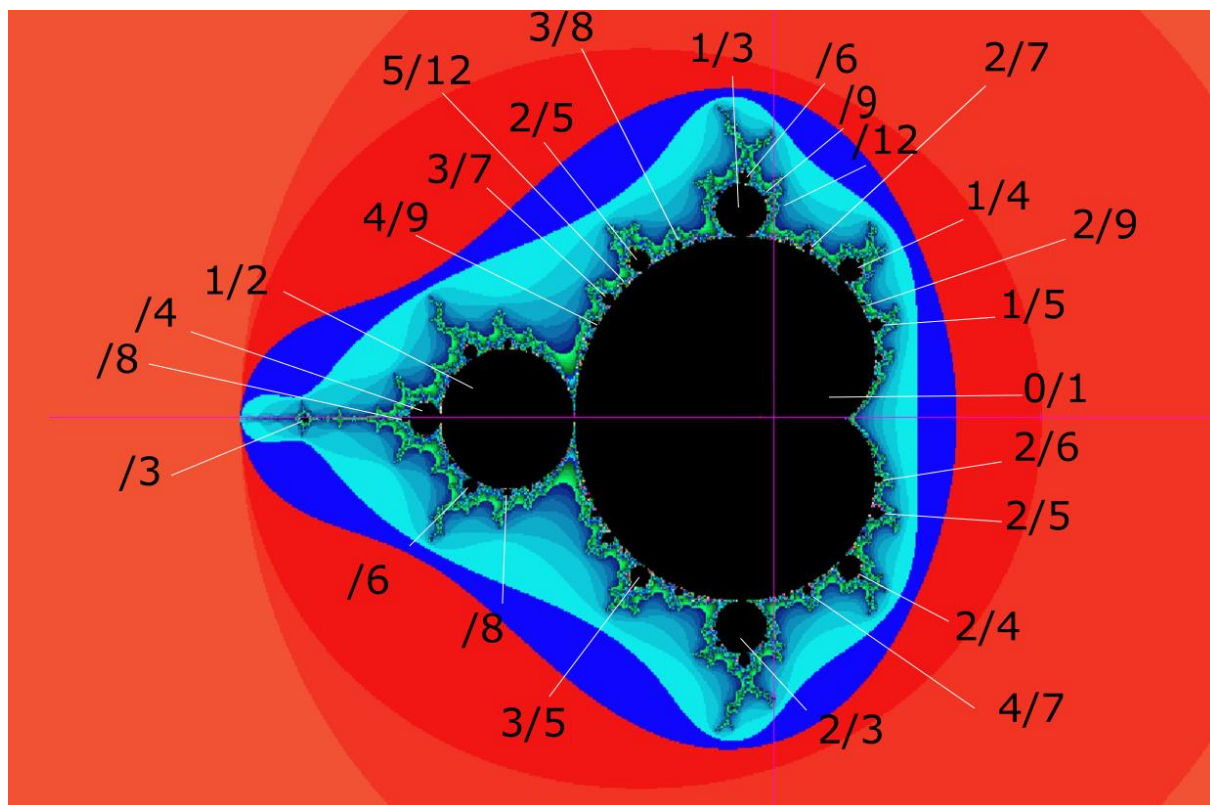
Si le polynôme est de la forme $ax+bx^2+\dots$, (donc pas dans le cas de Z^2+C) avec $a=\exp(2 i \pi t)$, et sous certaines conditions d'irrationalité du coefficient t, les «disques de Siegel» apparaissent autour des points fixes neutres, et le cycle tourne autour de ces cercles. si t est rationnel, $t=p/q$, $t^n=1$ et on a un n cycle autour de 0, sinon un cycle infini sur le cercle de centre O. Lorsque le polynôme est de degré >3 apparaissent des «anneaux de Herman»

à la gauche du Mandelbrot les disques successifs ont un cycle de 2,4,8,... points, le rapport des diamètres de ces disques tend vers 4.669, le nombre de Feigenbaum
sur la gauche les centres des miniMandelbrot sont repartis de sorte que $(C_n-C_{n-1})/(C_{n+1}-C_n)$ tend vers 4

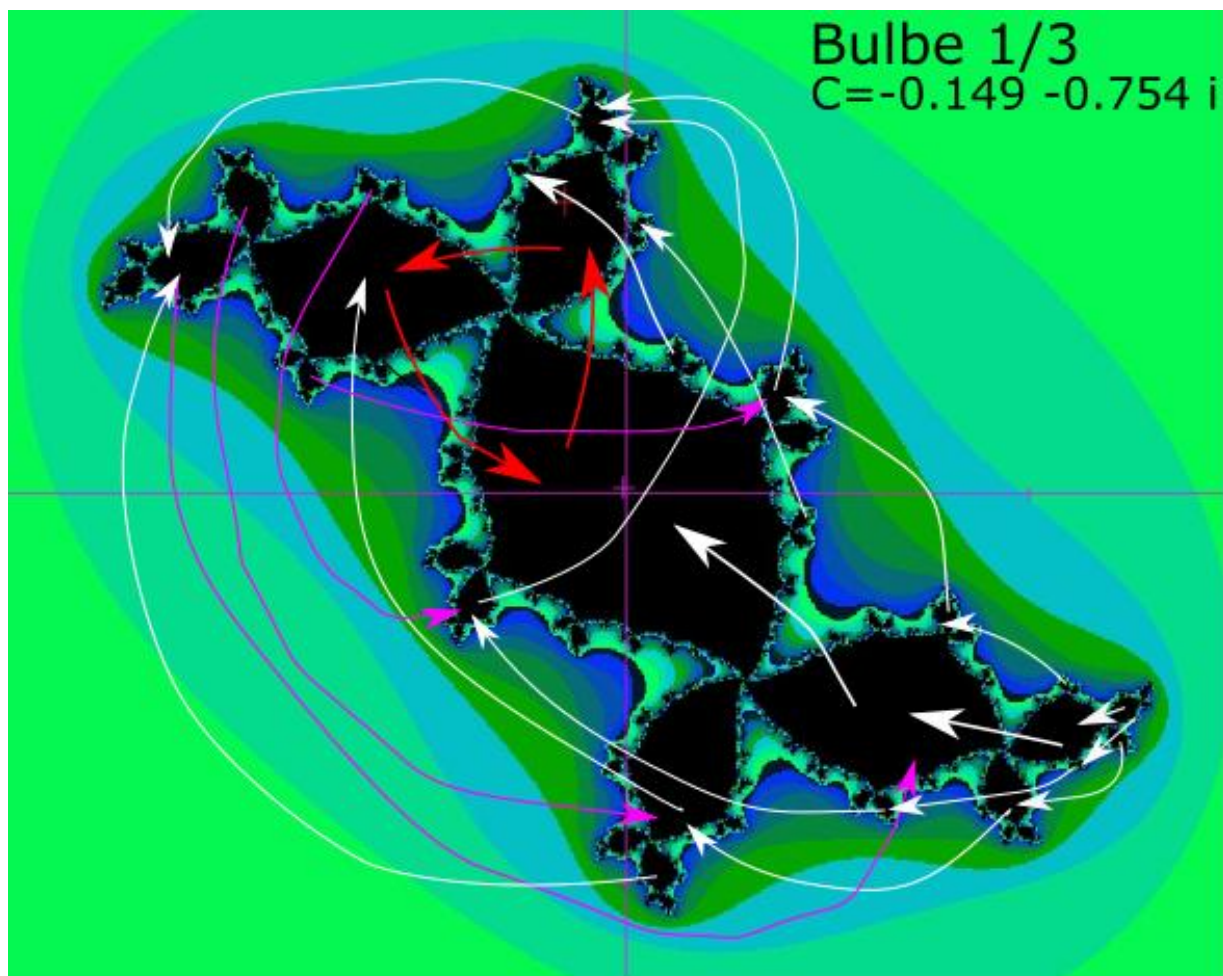
à la frontière, tout cercle contient des points du Mandelbrot et des points qui n'y sont pas
tout cercle sur la frontière contient un Mandelbrot réduit.

certain points de la frontière, les points de Misiurewicz, itèrent sur un point d'un cycle, ce sont les centres des spirales, les embranchements, le julia correspondant ressemble à un agrandissement du Mandelbrot en ce point.

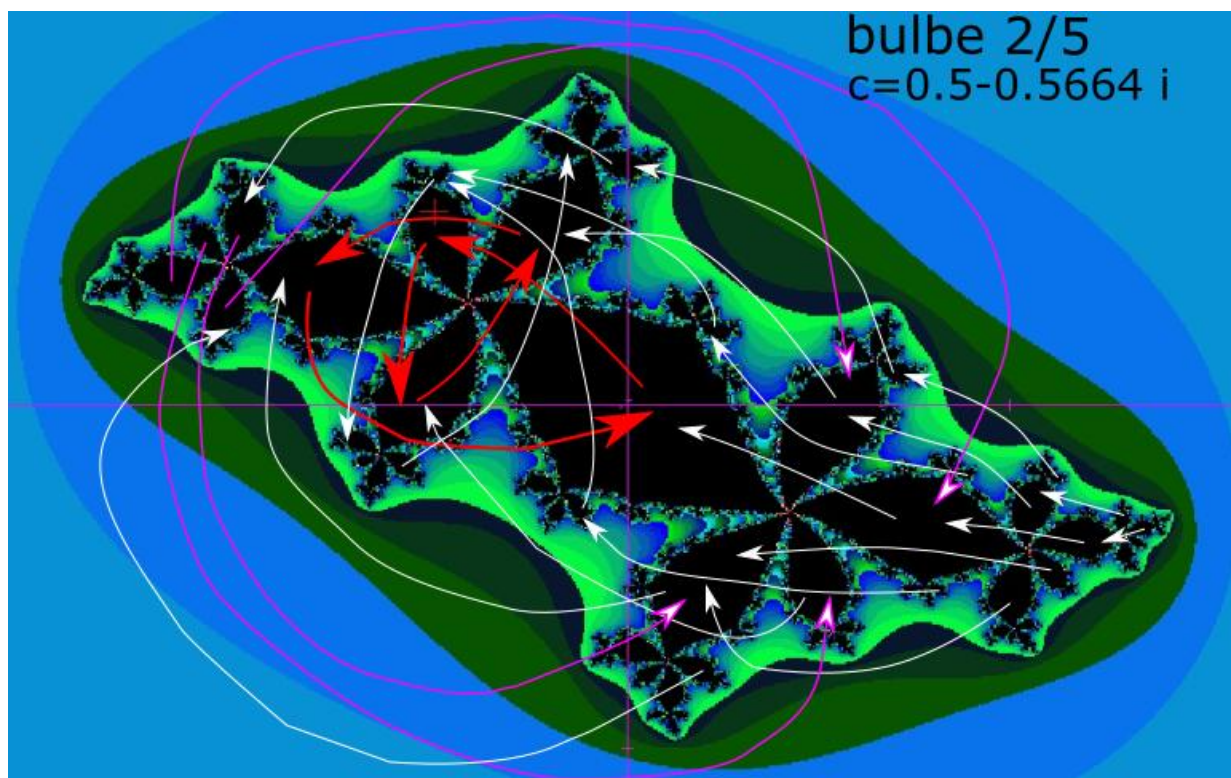
en ces points un léger déplacement du point C déforme rapidement le Julia, qui passe d'un type de cycle à un autre (période différente, ...) on les caractérise par : $f^n(0)=f^{n-k}(0)$

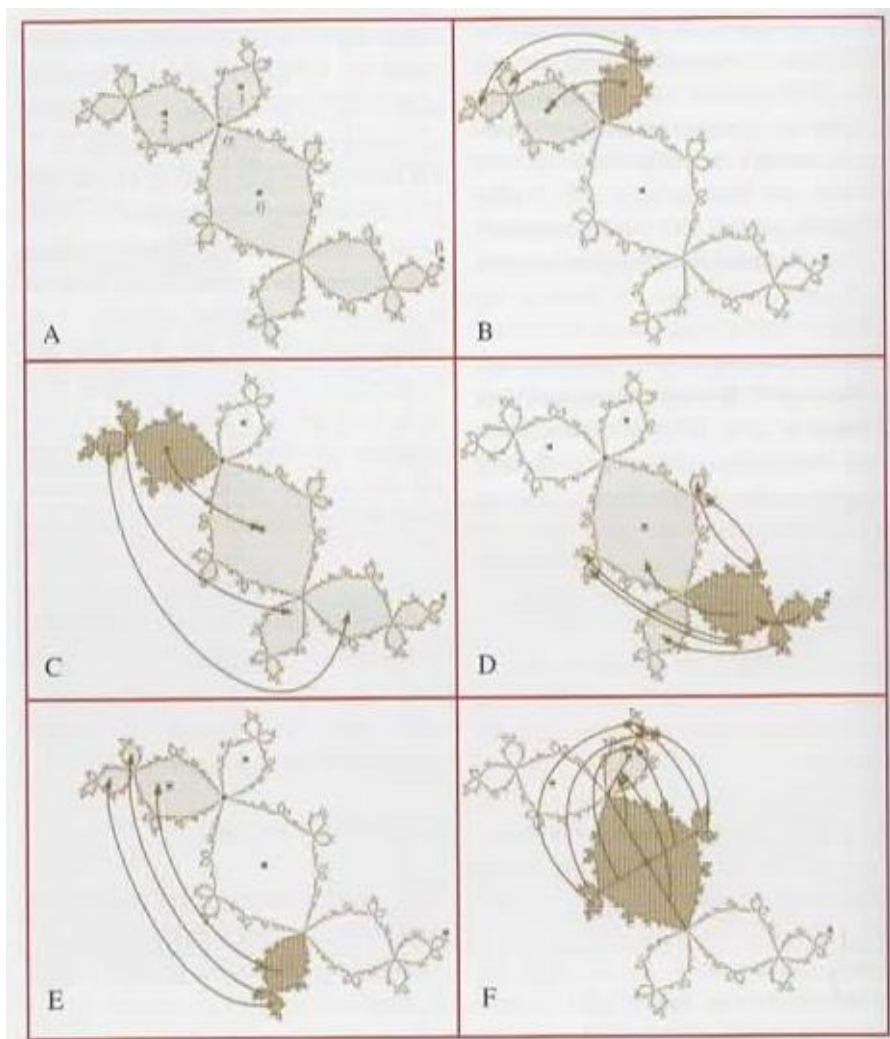


Bulbe 2/5 – la petite antenne est la 2^{ème} de 5 dans le sens trigonométrique

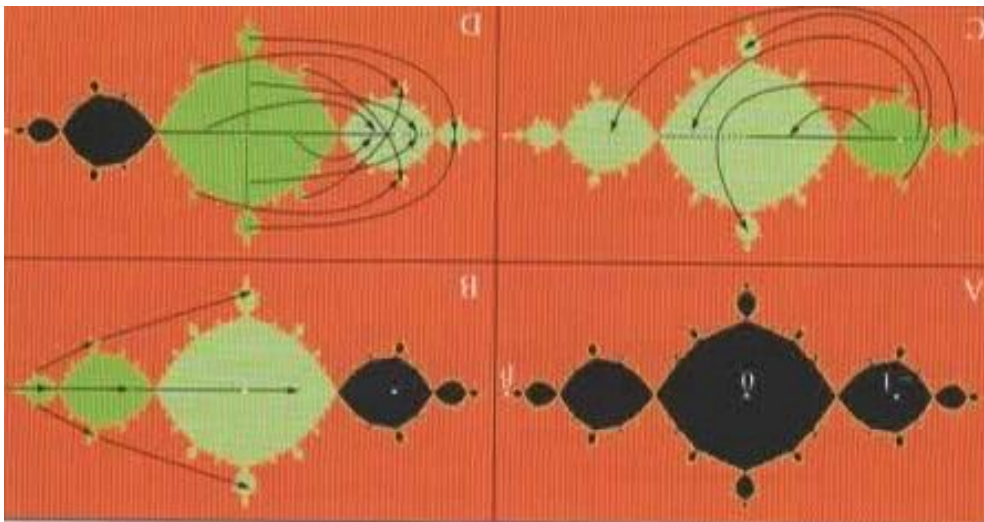
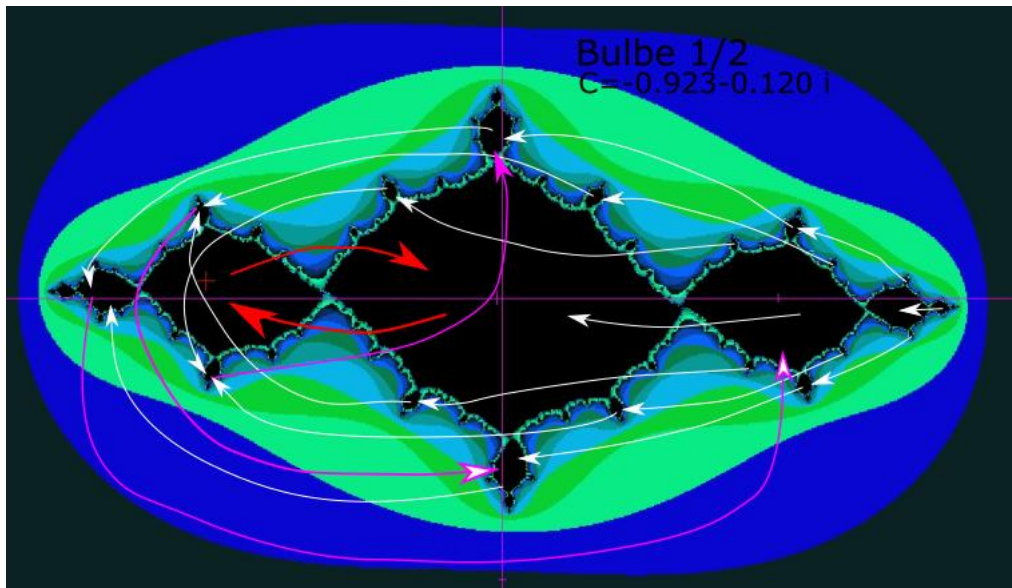


Les mouvements pré-cycliques des points du julia $1/3$





Transformée d'une branche : les décorations d'un lobe, y compris le lobe central, vont décorer la transformée du lobe en respectant leur position parmi les décorations voisines. Sauf pour les 2 lobes qui se projettent sur le lobe central – pour l'un certaines décorations restent attachées à la branche en se rapprochant du lobe et certaines vont sur la branche du lobe voisin Pour l'autre, faisant partie du cycle, certaines décorations vont sur les lobes opposés



Plusieurs types de coloration sont possibles :

par le nombre d'itération avant la divergence vers l'infini

par le module de divergence

par l'angle de divergence

par le module minimum de $f^n(0)$

Par le nombre d'itération pour atteindre le minimum de $f^n(0)$, nombre qui, résultat surprenant, suit la suite de Fibonacci ($F_n = F_{n-1} + F_{n-2}$)

En visualisant les points et bassins d'attraction, la colorisation se fait par le nombre d'itération avant la convergence vers l'infini un point fixe ou un cycle.

La détection du cycle est plus longue car il faut détecter l'égalité à epsilon près avec un des n points précédents.

souris gauche dans Mandelbrot : choix du point C, recentrage et zoom.

souris gauche dans Julia : recentrage et zoom.

souris droite dans Mandelbrot : choix du point C et affichage du Julia correspondant.

souris droite dans Julia : choix d'un point Z et suivi de son cycle.

souris du milieu : recentrage et zoom dynamique

Rappel :

le nombre complexe $x + i y$ se représente par le point de coordonnées cartésiennes (x,y) , toutes les opérations et fonctions classiques sont définies pour des complexes :

$$Z = x + i y = r \exp(i t) = r * (\cos(t) + i \sin(t)) \text{ avec } r = \sqrt{x^2 + y^2} \text{ et } t = \arctan(y/x)$$

$$x + i y + m + i n = x + m + i (y + n)$$

$$(x + i y) * (m + i n) = x * m - y * n + i (x * n + y * m)$$

$$\sin(x + i y) = \sin(x) * \operatorname{ch}(y) + i \cos(x) * \operatorname{sh}(y)$$

$$\cos(x + i y) = \cos(x) * \operatorname{ch}(y) - i \sin(x) * \operatorname{sh}(y)$$

$$\exp(x + i y) = \exp(x) * (\cos(y) + i \sin(y))$$

$$z * z' = (r * \exp(it)) * z' = \exp(z' * (\log(r) + i t))$$

Par le programme, on peut aussi étudier les fonctions suivantes, qui sont de deux sortes :

$$Z_{n+1} = f(Z_n) + C$$

$$Z_{n+1} = C f(Z_n)$$

Certaines ont des temps de calcul longs.

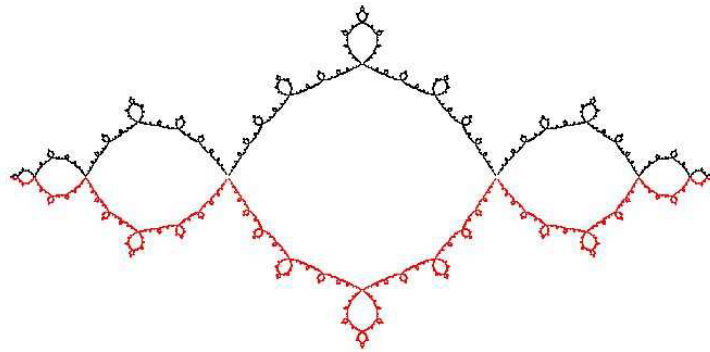
Julia comme attracteur d'un IFS

On considère les 2 fonctions inverses de Z^2+C , qui donnent les 2 antécédents possibles d'un point z de la suite $z_{n+1} = z_n^2 + c$

Ce sont $z_{n+1} = \pm \sqrt{-C + z_n}$ (sqrt est la racine dont la partie réelle est positive)

Rappel : En notation polaire z c'est le point (r, θ) et $\sqrt{z}$ c'est le point $(\sqrt{r}, \theta/2)$.

Cette suite des antécédents ne diverge pas, car ces points ont par définition des successeurs par z^2+C dans le Julia et donc qui ne divergent pas et donc ces antécédents sont dans le Julia, ils sont même de plus en plus proches de la frontière, et ces deux fonctions constituent donc un ifs dont l'attracteur est la frontière du Julia. Voilà un bon moyen pour le dessiner !



Le Julia est l'invariant dans cet IFS, or le Julia est constitué de plusieurs bulbes tangents 2 à 2, et il est curieux de comprendre comment ces différents « ovales » composants du Julia sont transformés par ces 2 fonctions.

Pierre Audibert en a fait une étude très intéressante ([JuliaFatou \(pierreaudibert.fr\)](http://JuliaFatou.pierreaudibert.fr)) où il montre comment les différents bulbes d'un Julia s'agrègent. En voici un résumé :

En noir l'ovale de départ (ressemblant de loin à un Julia) et sa translation selon C (ovale vert)

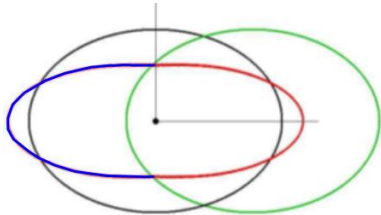
On prend l'exemple $C=-1$ (centre du disque $\frac{1}{2}$) Ce Julia a une forme d'avion vu de face.

Il coupe ox et oy en $(a,0)$ et $(0,b)$

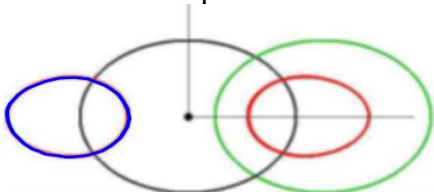
Il faut alors distinguer 2 cas selon la position de O par rapport à l'ovale vert

- O est dans l'ovale vert

En rouge et bleu l'ovale d'arrivée, parcouru en rouge pour les abscisses positives par la 1^{ère} fonction et en bleu par la seconde pour les abscisses négatives



- O n'est pas dans l'ovale vert



Cela montre que pour que cela reste un Julia il faut être dans le 1^{er} cas, avec $a > 1$

La transformée de l'intersection avec l'axe ox doit rester invariante donc $\sqrt{a+1}=a$

D'où $a=(1+\sqrt{5})/2$ c'est le nombre d'or, φ

et l'autre extrémité à 180° donne l'intersection à 90° sur oy , soit $b=\sqrt{\varphi-1}=\sqrt{\varphi'}$

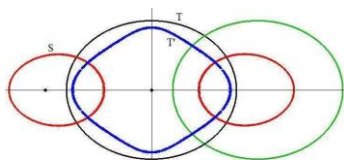
avec $\varphi'=\varphi-1=(\sqrt{5}-1)/2$

Considérons maintenant un ovale plus petit, centré en 0, représentant le bulbe central du julia. Il a un $a < 1$ et donc se transforme en 2 bulbes sur l'axe des x, qui doivent rester invariants à l'itération suivante, ce qui amène à étudier l'itération double $\pm \sqrt{1 - \sqrt{1 + z}}$ et les transformés de l'intersection aux axes comme ci-dessus :

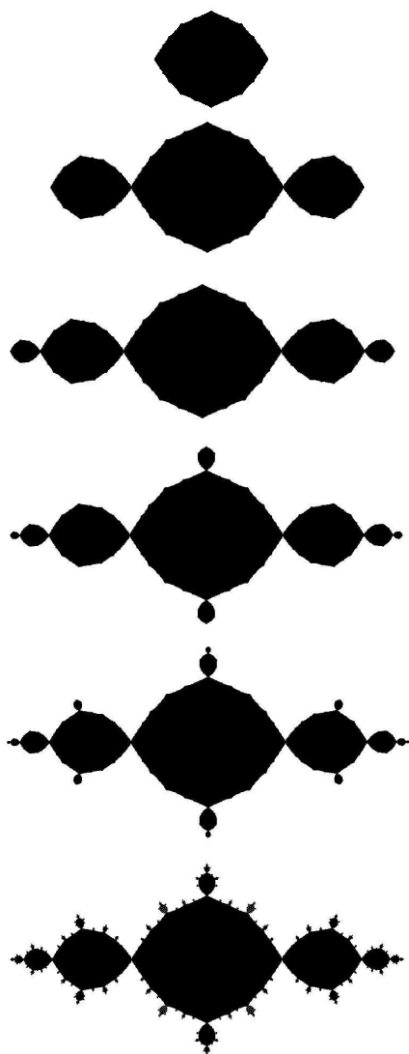
$$\sqrt{1 - \sqrt{1 - a}} = a, \text{ ce qui donne } a = \varphi' \approx 0,618$$

$$\sqrt{\sqrt{1 + a} - 1} = b, \text{ d'où } b \approx 0,53$$

Ainsi on voit apparaître à chaque itération des couples de bulbes qui sont tangents aux précédents. (un bulbe dont le centre est à l'angle θ donne un bulbe à $\theta/2$ et son symétrique par rapport à O).



Le tronc en bleu, les 2 premiers bulbes en rouge



A partir du tronc, fabrication étape par étape des nouveaux morceaux à partir des précédents sous l'effet de la double fonction répétée $z' = \pm \sqrt{z - C}$ dans le cas $C = -1$

Newton

L'approximation de Newton pour rechercher les zéros d'une fonction consiste à approximer la courbe par sa tangente :

$$F(Z) - F(r) = (Z-r) * F'(Z)$$

$$\text{or } F(r)=0.$$

$$\text{donc } r = Z - F(Z)/F'(Z)$$

On considère la suite

$$Z_{n+1} = Z_n - F(Z_n) / F'(Z_n)$$

Selon le point de départ, cette suite converge ou non vers une des n racines du polynôme. l'ensemble des points ne convergeant pas à la forme des ensembles de Julia associés à l'ensemble de Mandelbrot.

La 1ère fonction présentée, de degré 3, est un cas où tous les points ne convergent pas. pour les suivantes ils convergent, à moins de déplacer les racines (souris du milieu)

Les points sont colorés selon la racine vers laquelle ils convergent ou selon la rapidité de convergence - noir indique une divergence vers l'infini ou un cycle au voisinage de 2 racines.

Ri étant les n racines, on sait que $F(Z)=(Z-R_1)(Z-R_2)...(Z-R_n)$

On en déduit facilement que la dérivée $F'(Z)$ est la somme des n produits $Z-R_i$ correspondants à $n-1$ racines.

Pour un zoom, cliquer gauche sur les points colorés voisins des frontières, ce qui place le point au centre de l'écran, ou utiliser le bouton «zoom dynamique», qui zoom sur le centre de l'écran L'option d'affichage de la page html doit être à 100%

Augmenter le degré du polynôme revient à ajouter une racine

les racines sont initialement réparties sur le cercle unité

On peut sélectionner une des n racines (elle apparaît alors avec une croix rouge) puis effectuer un traveling ou un changement de position en cliquant avec la souris du milieu ou avec le bouton «déplacer la racine sélectionnée»

cliquer droit pour définit le point de départ d'un cycle de convergence

si on choisit un point dans les ensembles noirs (Julia) le cycle ne converge pas vers une racine

souris gauche : zoom une étape

souris droite : début de cycle de convergence vers une racine

souris du milieu : déplacer la racine sélectionnée

Acte III :

MULTO SERIOSO

if you = pas sérieux goto 1ère partie

Dans l'antre de Bakus, une file d'arbres et un arbre de files
Syntaxikos et Semantica discutent en se cachant

Semantica :

- basics et son frère fortranos, y commencent à me les gonfler, ils
arrêtent pas d'aller d'avant en arrière et de droite à gauche, j'arrive pas à les
suivre.

Syntaxikos

- pour sur, on devrais voir à leur faire leur compte avec Pascalina

Semantica

- bien vu, ou alors avec ada, avec eux au moins on peut se syntaxiquer en
rond

Semantica

- et le nouveau, Prologistos, il pourrait nous aider,
il est vachement balaise à ce qu'on dit.

Prolog

Un langage informatique classique manipule principalement des nombres et des caractères, et sert à définir une suite d'opérations et de manipulations à leur faire subir pour arriver à un résultat, un langage classique décrit donc un algorithme que le programme applique.

Le langage Prolog est né à Marseille vers 1972, ses pères sont Alain Colmerauer et Philippe Roussel. C'est un langage déclaratif, non procédural, c'est à dire que l'on se contente de déclarer des faits et des règles de déduction, on indique donc le "quoi" et le moteur d'inférence intégré à l'interpréteur prolog se charge du "comment" et trouve les solutions en parcourant ces déclarations sans que l'on ait à préciser l'algorithme à utiliser.

Prolog manipule le nom des objets plutôt que leur contenu et permet donc des manipulations symboliques, comme on le verra dans l'exemple de dérivation et intégration mathématique. Il fournit aussi une gestion de liste grâce à laquelle on peut représenter des phrases que le moteur d'inférence compare à plusieurs modèles qui servent de filtre d'où on déduit une analyse de la phrase. Prolog est ainsi l'outil rêvé pour l'analyse de grammaires, ainsi qu'à quatre pages d'ici je vous le fais savoir.

1.1 Un peu de logique

1.1.1 calcul des prédicats

On se propose d'étudier des affirmations logiques, chacune pouvant donc être soit vraie soit fausse. Pour cela, on doit d'abord considérer un ensemble d'objets et un ensemble de relations possibles ou prédicats entre ces objets. Les affirmations peuvent contenir des objets dont on ne spécifie pas la valeur, les variables, et on dit alors que la logique est d'ordre 1 par opposition à la logique d'ordre 0, sans variables. Enfin il faut aussi considérer les quantificateurs "pour tout" et "il existe" qui sont toujours associés aux affirmations contenant des variables. Une affirmation avec variable est soit vraie pour toutes les valeurs que l'on donne à ces variables, soit toujours fausse, soit vraie pour certaines valeurs et fausses pour d'autres, selon le cas on dit alors que c'est une tautologie, une contradiction ou une affirmation satisfaisable.

Voici quelques exemples d'affirmations :

gros minet est un chat
il existe X, X est vivant
pour tout X, il existe Y, Y est père de X

On combine enfin ces affirmations par les opérateurs :

et, ou, non, implique, est équivalent à

Et on obtient d'autres affirmations, dont on peut calculer la vérité selon les tables de vérité de chacun de ses opérateurs.

X est parent de Y et Y est un homme implique Y est fils de X

Pour faciliter la suite des opérations, prolog adopte une notation parenthésée où le nom de la relation tient une place de fonction, soit pour l'affirmation précédente :

est_parent(X , Y) et est_homme(Y) implique est_fils(Y , X)

Certaines de ces affirmations, bien que de formes différentes, ont le même sens, selon des règles très voisines de l'algèbre, l'addition devenant "ou" et la multiplication "et".

non (A et B)	non A ou non B
non (A ou B)	non A et non B
non (non A)	A
A et (B ou C)	(A et B) ou (A et C)
A et (B et C)	A et B et C
A ou (B ou C)	A ou B ou C
A et B	B et A
A ou B	B ou A
A et A	A
A ou A	A
A et non A	faux
A ou non A	vrai
A et vrai	A
A et faux	faux
A ou vrai	vrai
A ou faux	A
A implique B	non A ou B
si A alors B	A implique B
	soit : non A ou B
si A alors B sinon C	A implique B et non A implique C
	soit : (non A ou B) et (A ou C)
A est équivalent à B	soit : (A et B) ou (B et C) ou (non A et C)
	A implique B et B implique A
	soit : (non A ou B) et (non B ou A)
	soit : (A et B) ou (non A et non B)
non(tout X, relation(X))	il existe X, non relation(X)
non(il existe X, relation(X))	tout X, non relation(X)

Munis de ces quelques règles, on peut simplifier toutes les affirmations, exactement comme on le fait avec des formules arithmétiques. On commence par supprimer les "implique" et les "est équivalent à", puis on applique les "non" qui portent sur des affirmations complexes. Et en procédant ainsi, toute affirmation finit par s'écrire sous une forme ne contenant que des "ou", la forme clausale :

P ou Q ou R

ce qui est équivalent à

R ou P ou Q

chaque élément P, Q ou R pouvant être nié et contenir des "et" (comme un polynôme arithmétique) :

A ou B ou (C et D) ou non M ou non N ou non P

1.1.2 clause de horn

C'est une forme clausale dont un seul terme n'est pas nié :

$A \text{ ou non } M \text{ ou non } N \text{ ou non } P$

ce qui a, d'après nos règles, le même sens que

$M \text{ et } N \text{ et } P \text{ implique } A$

et cela peut aussi se comprendre :

si M, N et P sont vrais alors A est vrai.

C'est la seule forme que l'on représente directement en Prolog :

$A \rightarrow M, N, P.$

Maintenant que nous disposons de cet ensemble de formes clausales correspondants aux faits et aux règles, le problème consiste à en déduire quelque chose ou à répondre à une question. Le mécanisme de la déduction consiste à déduire une nouvelle affirmation d'un ensemble d'affirmations ou axiomes de départ. Les mécanismes de base, répertoriés par les anciens grecs sont :

le modus ponens

si P est vrai et si P implique Q

alors on peut déduire que Q est vrai

le modus tollens

si Q est faux et si P implique Q

alors on peut déduire que P est faux

le chainage

si P implique Q et si Q implique R

alors on peut déduire que P implique R

la spécialisation

si pour tout X , $P(X)$ est vrai

alors $P(a)$ est vrai

La question n'est rien d'autre qu'une affirmation dont il faut vérifier l'exactitude. On la met donc elle aussi sous forme clausale. Cette forme clausale est particulière, elle doit n'avoir qu'un élément (pas de "ou").

$I \text{ et } J \text{ et } K$

ce que l'on note en prolog :

$I, J, K ?$

et qui signifie : est ce que I et J et K sont vrais tous ensemble ?

Le problème est alors analogue à celui de la résolution d'un système de N équations à P inconnues. il peut y avoir plusieurs, une seule ou pas de valeurs possibles pour les variables, qui jouent le rôle des inconnues. La première idée qui vient à l'esprit consiste à appliquer toutes les règles applicables, d'où l'on déduit de nouveaux faits que l'on ajoute à la base de faits. Cette méthode s'appelle le chainage avant, et l'algorithme s'arrête lorsqu'on ne trouve plus de nouvelles règles applicables ou lorsque l'on a trouvé un fait qui correspond justement à la question. La méthode employée par Prolog s'appelle le chainage arrière, elle consiste pour démontrer un fait à appliquer uniquement les règles qui ont ce fait en conclusion, et à démontrer successivement les conditions de cette règle.

Les objets prolog

Prolog manipule des termes et des listes de termes. un terme étant soit une constante, soit une variable, soit un terme composé.

2.1 Constante

Les constantes sont les nombres, les caractères, les chaînes de caractères et les identificateurs.

Les nombres réels sont représentés selon la notation exponentielle :

+0.11 e+10

pour les entiers, pas de surprise.

Une chaîne de caractères est entourée de guillemets. C'est par exemple un texte à afficher ou une phrase à analyser.

"c'en est une"

Un identificateur est une suite de caractères symbolisant un concept quelconque. Si ce nom contient des caractères spéciaux, il faut l'entourer d'apostrophes.

nom

autre_exemple

'c'en est un autre'

2.2 Variable

Une variable désigne un objet de type quelconque dont on recherche les valeurs possibles c'est à dire qui soient solutions du système des équations logiques formées par la question et par les faits et règles. Cette variable sera "unifiée", c'est à dire substituée, par cet objet au cours de l'exécution du programme. La variable est dite libre avant cette unification et liée après. On note les variables par un identificateur dont le premier caractère est "_". Une variable anonyme est constituée du seul "_" ou commence par "__". disons qu'elle désigne un objet quelconque dont on ne s'intéresse pas à la valeur mais seulement à l'existence.

2.3 Terme composé, prédicat

Un terme composé comprend un identificateur de fonction et des termes de types quelconques (pouvant être distincts) séparés par des virgules :

fonction(paramètre1 , paramètre2 , _var , 45 , sousfon([1,a]))

2.4 liste

Une liste regroupe un nombre quelconques d'objets de tous types (pouvant être distincts). Une liste est ordonnée, ce n'est pas un ensemble. On sépare les objets par des virgules et on entoure le tout de "[" et de "]". les éléments d'une liste pouvant être eux-mêmes des listes, on peut ainsi représenter toutes arborescences.

```
[ premier,deuxieme,_var,12,fon(a),[1,2]]
```

La liste vide se note :

```
[]
```

Le caractère "!" sert à définir le reste de la liste

(un peu comme le "cdr" du langage Lisp):

```
[a, b, !_x]
```

cette liste a visiblement deux éléments constants, connus : a et b et une fin inconnue, représentée par la variable : _x , qui peut être vide.

Une chaîne de caractères est une liste de caractères :

```
[a, h, ' ', b, o, n] est identique à "ah bon"
```

Programme

Un programme est un objet prolog composé de clauses, elles-mêmes regroupées en sous programmes, et de questions.

3.1 Clause

Une clause est constituée d'une tête de clause, d'une queue de clause et d'un point. La tête de clause est un terme composé. La queue de clause est un ensemble de termes composés séparés par des virgules. Tête et queue de clause sont séparés par "->" .

3.1.1 Fait

C'est une clause dont la queue est vide :

```
aime(souris, gruyère).
```

3.1.2 Règle

C'est une clause dont la queue n'est pas vide :

```
canidé(_animal) ->  
    chien(_animal) .
```

Cette règle signifie sans aucun doute que si l'animal est un chien, alors c'est un canidé.

```
frères(_x, _y) ->  
    père(_x, _),  
    père(_y, _),  
    _x \= _y,  
    homme(_x),  
    homme(_y) .
```

Celle-ci signifie que si `_x` et `_y` sont des hommes différents et si ils ont le même père (`"_"`), alors ils sont frères.

Les règles servent à déduire d'un ensemble de conditions constitué par la queue de clause, une conclusion : la tête de clause. La virgule entre les conditions de la queue de règle a donc le sens de "et". Les variables de la tête de règle servent à communiquer avec la règle appelante mais à la différence des langages traditionnels, une même variable sera tantôt variable d'entrée et tantôt variable de sortie, selon la forme de l'appel. Les variables d'une règle sont locales à cette règle, comme dans les procédures classiques. L'ensemble des faits et des règles constitue la base de données que le programme manipule.

3.2 Sous-programme

C'est un ensemble de clauses commençant par le même prédicat.

```
sympa(_qqn) -> sourire(_qqn).
```

```
sympa(_qqn) -> raconte(_qqn, _qqchose), histoire_drôle(_qqchose).
```

où il apparaît que quelqu'un est sympa si il a le sourire ou si il raconte une histoire drôle.

Le passage d'une règle à une autre de même prédicat ayant le sens de "ou".

3.3 Question

Une question a la structure d'une queue de clause et est précédée ou suivie du caractère "?". L'appel d'un prédicat prédéfini est automatiquement considéré comme une question et non comme un fait. (ceci pour des raisons de commodité, car on oublie 3 fois sur 4 le "?"). La réponse à une question consiste à chercher les faits qui s'unifient à la question posée.

```
sympa(Cezigue)?
```

qui signifie : est-ce que Cezigue est sympa?

```
père(_fils , toto) , homme(_fils) ?
```

qui signifie : quels sont les fils de toto ?

soit encore la règle :

```
père(_fils , _père) ->
```

```
    fils(_père , _fils) .
```

et un ensemble de faits :

```
fils( .. , .. ).
```

Contrairement aux habitudes des langages classiques, lors des appels on peut utiliser tout paramètre comme constante ou comme variable

```
père(_fils , bernard)?
```

et

```
père(pierre , _père)?
```

sont tous deux valides et rendent respectivement :

```
_fils = les fils de bernard
```

et

```
_père = le père de pierre
```

Ceci n'est pas toujours vrai pour les prédicats prédéfinis, qui imposent que leurs paramètres ne soient pas tous des variables et même parfois qui imposent la position des paramètres fournis et des variables: les prédicats prédéfinis sont tous plus ou moins des violations de Prolog. Lorsque tous les paramètres sont fournis, cela signifie que l'on veut vérifier un résultat.

La documentation de l'interpréteur prolog est dans doc_prolog.pdf.

En voici un extrait :

Pour lancer l'interpréteur prolog, faire ;

```
    myprol fichier.pro  
ou    myprol fichier  
ou    myprol
```

Pour lire un fichier faire alors

```
    echo.  
    load(fichier).
```

clear_all	efface toutes les règles et tous les faits
list_all	fait un listing complet
list(prédicat)	liste le sous-programme en question
quit	fin de myprol

Moteur d'inférence

Fonctionnement

L'interpréteur va étudier les conditions de la question posée les unes après les autres pour déterminer si elles sont réalisées ou non. Dès qu'une condition n'est pas remplie, le sous-programme s'arrête et rend la valeur "faux". L'ordre d'analyse des conditions est, on l'a vu, théoriquement indifférent, mais Prolog effectue toujours son analyse de gauche à droite et en ce sens, le langage est procédural car dans la mesure où certaines conditions sont en fait des directives cela permet justement de préciser l'ordre dans lequel on veut les exécuter, autrement dit cela permet de préciser un algorithme.

La vérification d'une condition consiste à chercher des faits (ou des têtes de règles) qui s'unifient aux éléments de la condition. (si c'est une règle il faut alors vérifier toutes les conditions de la queue de la règle). Certaines conditions contenant des variables, Prolog s'efforce de donner à ces variables des valeurs telles que la condition soit réalisée. Les résultats obtenus pour démontrer les premières conditions servent pour vérifier les suivantes, c'est à dire que si la même variable apparaît dans plusieurs conditions, et que la variable a reçu une valeur dans les premières conditions, c'est cette valeur qui est utilisée pour la variable pour vérifier les conditions suivantes, et cette variable ne pourra plus changer de valeur. Et bien voilà autre chose ! on ne peut pas, dans le même sous-programme faire :

```
    _x := 4  
puis, un peu plus loin :  
    _x := 5  ?
```

Non, car les conditions précédemment vérifiées l'ont été avec la première valeur (4) et ne le seraient peut être pas avec la nouvelle (5), et cela conduirait à déduire tout et le contraire de tout.

Lorsque l'interpréteur a vérifié la dernière condition, c'est qu'il a trouvé une solution qui satisfait toutes les conditions. Il affiche alors la valeur qu'il a donné aux variables présentes dans la question ou simplement "oui" s'il n'y en avait pas. il lui reste alors à chercher d'autres valeurs pour les variables, pour avoir toutes les solutions. Si il ne peut trouver de solution, il affiche "non", qu'il faut parfois comprendre comme :

"je ne sais pas, ce fait n'est pas dans la base"

Unification

C'est le mécanisme de base de la résolution. il consiste à chercher à mettre en correspondance deux objets (pattern matching). C'est à cette occasion que les variables sont substituées par ce qu'il faut pour que l'unification réussisse et donc pour que la condition soit

réalisée. On montre que si la substitution est possible, il existe une substitution minimal unique des variables qui rendent les deux termes égaux et toute autre substitution se décompose en un produit dont l'un des termes est la substitution minimale.

Soit à unifier :

homme(_qqn) et homme(Nabuchodonosor) _qqn s'unifie à Nabuchodonosor

$1 + \sin(_x) - _y(a,b)$ et $1 + \sin(_p) - \text{fonc}(_p,b)$
_x s'unifie à _p puis à 'a', soit donc à 'a' et _y à 'fonc'

Le cas de la liste est intéressant, il faut bien comprendre le signe "!"

[1,2,_reste] et [1,2,3] _reste s'unifie à 3

[1,2,_reste] et [1,2,[3,4]] _reste s'unifie à [3,4]

[1,2,_reste] et [1,2,3,4] échoue
car si _reste s'unifiait à [3,4] on serait dans le cas précédent

[1,2,!_reste] et [1,2,3,4] _reste s'unifie à [3,4]
car "!" indique que _reste est la liste égale à la fin de la liste

si _phrase vaut [Pierre , donne , chocolat , à , Marc]
alors, unifier _phrase à [_sujet , donne , _qqchose , à , qqun]
donne : _sujet = Pierre, _qqchose = chocolat et _qqun = Marc

On voit donc comment l'on organiserait un petit programme chargé d'analyser les commandes d'un petit système informatique intégré :
intégré -> repeat , readwords(_phrase) , analyse(_phrase) .

analyse([imprime , _fichier , sur , l , imprimante])->print(_fichier).
analyse([imprime , _fichier , sur , l , écran]) -> type(_fichier) .
analyse([calcule , !_reste]) -> calcul(_reste) .
analyse(_x) -> writeln("vous dites ?") .

calcul([le , bénéfice , de , _mois]) -> execute('benef.exe' , _mois) .
calcul([la , paye]) -> paye.
calcul(_x) -> writeln("je ne sais pas calculer " , _x) .

Le sous-programme "analyse" réalise, en beaucoup plus fort, ce que l'on fait par un "case" en pascal. L'élément du "case" n'est plus un simple entier mais un terme composé qui peut être très complexe, c'est ici la liste des mots de la phrase à analyser.

Retour arrière

En cas d'échec à l'unification, et même en cas de succès, si l'on veut toutes les solutions, l'interpréteur est amené à revenir sur les choix qu'il a fait tant au niveau des variables unifiées que au niveau des règles des sous-programmes. il y a donc parcours de l'arbre de

résolution en arrière jusqu'au nœud correspondant au choix à refaire. Les variables qui avaient été substituées sont alors remises à l'état de variables.

algorithme d'unification

On peut exprimer en Prolog ce que unifier deux choses veut dire :

```
unifier(_x , _y) ->
    var(_x) , _x = _y .
unifier(_x , _Y) ->
    var(_y) , _y = _x .

unifier(_x , _y) ->
    constant(_x) , constante(_y) , _x = _y .
unifier(_x , _y) ->
    structure(_x) , structure(_y) ,
    unifier_structure(_x , _y) .

unifier_structure(_x , _y) ->
    functor(_x , _fx , _nbarg) ,
    functor(_y , _fy , _nbarg) ,
    unifier_arguments(_x , _y , _nbarg).

unifier_arguments(_x , _y , 0).
unifier_arguments(_x , _y , _n) ->
    _n > 0 ,
    arg(_n , _x , _ax) ,
    arg(_n , _y , _ay) ,
    unifier(_ax , _ay) ,
    _n1 := _n - ! ,
    unifier_arguments(_x , _y , _n1) .
```

Pour empêcher une variable de s'unifier avec un terme qui la contient, certains préfèrent remplacer le début par :

```
unifier(_x , _y) ->
    var(_x) , var(_y) ,
    _x = _y.
unifier(_x , _y) ->
    var(_x) , nonvar(_y) ,
    not_occurs_in(_x , _y) ,
    _x = _y.
unifier(_x , _y) ->
    var(_y) , nonvar(_x) ,
    not_occurs_in(_y , _x) ,
    _x = _y.
```

et le reste comme dans la première version.

```

not_occurs_in(_x , _y) ->
    var(_y) , _x /= _y.
not_occurs_in(_x , _y) ->
    nonvar(_y) , constant(_y).
not_occurs_in(_var , _structure) ->
    nonvar(_structure) ,
    structure(_structure) ,
    functor(_y , _fy , _nbarg) ,
    not_occurs_in_args(_nbarg , _x , _y).

not_occurs_in_args(0 , _x , _y).
not_occurs_in_args(_n , _x , _y) ->
    _n > 0 ,
    arg(_n , _y , _arg) ,
    not_occurs_in(_x , _arg) ,
    _n1 := _n - ! ,
    not_occurs_in_args(_n1 , _x , _y) .

```

Exemple de résolution

Soient les faits et les règles constituant les trois sous_programmes suivants:

```
aime(Claude , nager).  
aime(Catherine , nager).
```

```
parler_tout_le_temps(Claude).
```

```
boire(Philippe , petit_bordeaux_1948).  
boire(c_est_pas_moi , la_tasse).  
boire(_gus , la_tasse) ->  
    aime(_gus , nager) ,  
    parler_tout_le_temps(_gus).
```

Lors de la question :

```
boire(_qui , la_tasse)?
```

qui signifie visiblement :

```
qui c'est-t-y qui a bu la tasse ?
```

L'interpréteur choisit la première clause du sous-programme "boire" et la compare mot pour mot (l'unifie) à la question posée. on obtient donc d'abord _qui = philippe. puis il y a échec, car faut pas confondre le pinard et le château la pompe. Il y a alors abandon de la clause et l'interpréteur choisit la deuxième clause du sous-programme boire. L'unification réussit et donne la première solution:

```
_qui = c'est pas moi, c'est ma soeur qu'a cassé la machine à vapeur.
```

il y a tout de même abandon de la clause et choix de la troisième, qui est une règle. L'unification réussit en liant _gus à _qui et l'interpréteur cherche à démontrer les deux conditions de la règle. La première entraîne le déroulement du sous-programme "aime" et _gus s'unifie avec "Claude". l'appel du sous programme "parler_tout_le_temps" permet de satisfaire la deuxième condition et cela donne la deuxième solution :

```
_qui = Claude
```

Il y a alors retour à la deuxième clause du sous-programme "aime" et _gus s'unifie à "Catherine" mais la condition parler(Catherine) ne s'unifiant avec aucune clause du sous-programme "parler", il y a échec et retour arrière. et cette fois il n'y a plus de clause possible , ni pour aime, ni pour boire. Lors de la résolution, l'interpréteur affiche au fur et à mesure de chaque démonstration la valeur de substitution des variables non anonymes de la question posée.

soit dans notre cas :

_qui = c'est pas moi
_qui = claude

Lors de la question :

boire(_ , la_tasse)?

qui signifie :

y a t-y quelqu'un qui a bu la tasse ?

la variable est anonyme, on ne s'intéresse pas à sa valeur et il n'y a pas d'autre affichage que "oui".

rôle des caractères spéciaux

(	début des arguments d'un prédicat
)	fin des arguments d'un prédicat
,	séparateurs des arguments d'un prédicat ou des éléments d'une liste
, ou &	séparateur des prédicats d'une queue de clause (règle) ou des prédicats d'une question
[	début de liste
]	fin de liste
"	début et fin d'une chaîne de caractère
'	début et fin d'un identificateur contenant des caractères spéciaux
/	coupe des choix
!	séparateur entre tête et queue de liste
_	indicateur de variable
+	opérateurs arithmétiques
-	
*	
/	
^	
:=	affectation du nom (sans évaluation)
::=	affectation de la valeur (avec évaluation)
:=.	affectation du contenu (sans évaluation)
/*	début de commentaire
*/	fin de commentaire
->	séparateur entre tête et queue de règle
.	fin de fait, de règle ou de question
?	indicateur de question
<	comparateurs arithmétiques avec évaluation des deux membres
<= ou =<	
>	
>= ou =>	
=	égalité du nom sans évaluation
::=	égalité de la valeur avec évaluation
\=	différence sans évaluation
\:= ou <>	différence avec évaluation
<<	comparateurs symboliques sans évaluation
<<=	
::	transformation de liste en structure

les majuscules et les minuscules sont supposées distinctes. en particulier, les prédicats prédéfinis le sont en minuscule. l'espace ou la tabulation sont sans signification dans la mesure où ils ne coupent pas un identificateur ou une chaîne de caractères enclose de " ou '. une ligne peut ne pas être complète, la suite se trouvant sur la ligne suivante. une ligne peut contenir plusieurs commandes.

Acte 3 :

Prologos exemplaris

if not (you vouloir mourir idiot)
then sienta te
else passe ton chemin

Ad initio, Syntaxikos erat,

Et syllogismus hypotheticos fuit.

Et relationes et associationes

De ensemblis et listas, nodos et arcus labelos, et graphos

Et in omnia parte bakus notationem profitabile infiltravit .

Et inferencas motores rugisserunt in semanticos resos

Et in finem Semantica illuminavit .

Scène 1

Exemples en Prolog

Bon, assez bavardé, du concret.

Les quelques courts exemples qui suivent vont permettre de bien sentir le mécanisme de déduction interne à prolog, avant de se lancer dans des entreprises plus conséquentes.

Règles des relations familiales.

En Prolog, les règles de parenté peuvent s'écrire :

gosse(_x,_y) -> père(_y,_x).

dont voici la traduction simultanée :

si _x est le père de _y, alors _y est l'enfant de _x.

```
gosse(_x,_y)    -> mère(_y,_x).
vieux(_x,_y)    -> gosse(_y,_x).
mec(_x,_y)       -> nana(_y,_x).
nana(_x,_y)      -> père(_x,_y)      , mère(_x,_y) , / .
conjoint(_x,_y)  -> mec(_x,_y).
conjoint(_x,_y)  -> nana(_x,_y).
gamin(_x,_y)     -> gosse(_x,_y)      , mec(_y).
gamine(_x,_y)    -> gosse(_x,_y)      , nana(_y).
gendre(_x,_y)    -> gamin(_x,_y)      , nana(_y,_).
brue(_x,_y)      -> gamin(_x,_y)      , nana(_x,_y).
beauf(_x,_y)     -> conjoint(_x,_z)    , freresoeur(_y,_z) , mec(_x).
bellesoeur(_x,_y) -> conjoint(_x,_z)    , freresoeur(_y,_z) , nana(_x).
belledoche(_x,_y) -> conjoint(_x,_y)      , mère(_x,_y).
beaudab(_x,_y)   -> conjoint(_x,_y)    , père(_x,_y).
freresoeur(_x,_y) -> père(_x,_y)        , père(_y,_y) , _x \= _y.
freresoeur(_x,_y) -> mère(_x,_y)       , mère(_y,_y) , _x \= _y.
frangin(_x,_y)   -> freresoeur(_x,_y) , mec(_y).
frangine(_x,_y)  -> freresoeur(_x,_y) , nana(_y).
oncletante(_x,_y) -> gosse(_z,_x)        , freresoeur(_z,_y).
tantine(_x,_y)   -> oncleante(_x,_y)   , nana(_y).
tonton(_x,_y)    -> oncleante(_x,_y)   , mec(_y).
neveu(_x,_y)     -> oncleante(_y,_x)   , mec(_y).
nièce(_x,_y)     -> oncleante(_y,_x)   , nana(_y).
grand_parent(_x,_y) -> gosse(_y,_z)    , gosse(_z,_x).
```

```

petit_enfant(_x,_y)      -> grand_parent(_y,_x) .
pépé(_x,_y)             -> grand_parent(_x,_y) , mec(_y).
mémé(_x,_y)             -> grand_parent(_x,_y) , nana(_y).
petit_fils(_x,_y)       -> grand_parent(_y,_x) , mec(_y).
petite_fille(_x,_y)      -> grand_parent(_y,_x) , nana(_y).
cousin(_x,_y)           -> gosse(_z,_x) , gosse(_w,_y) , _z\=_w , freresoeur(_z,_w).
descendant(_x,_y)       -> gosse(_x,_y).
descendant(_x,_y)       -> gosse(_x,_z) , descendant(_z,_y).
ascendant(_x,_y)        -> descendant(_y,_x).

```

Remarques, gloses et notules diverses à bien se mettre dans la tête :

- Les deux règles "gosse" correspondent à un "ou" : on peut être de deux façons l'enfant de quelqu'un (c'est notre père ou c'est notre mère).

- La virgule de la règle "gamin" correspond à un "et" : on est le gamin de quelqu'un si on est son enfant et si on est un garçon.

- Prolog distingue les procédures de même nom et ayant un nombre de paramètres différents :

```

mec(_x)    _x est un homme
et mec(_x,_y)  _y est le mari de _x.

```

- La règle "freresoeur" donne un exemple de variable anonyme. _x et _y sont frère ou soeur si le père de _x est "_" et si le père de _y est aussi "_" ,mais le nom de ce père n'apporte rien à la chose, il est anonyme.

- La restriction $_x \neq _y$ dans la règle "freresoeur" est indispensable sinon on serait frère ou soeur de soi même car

```

père(_x , toto) ,
père(_y , toto) ,

```

donne pour _x tous les enfants de toto, puis pour chaque _x , donne pour _y tous les enfants de toto et rends donc tous les couples possibles .

Sans la restriction, la question

```
freresoeur(_x , _y) ?
```

donnerai donc :

```

_x , _y =      cain , cain
                cain , abel
                abel , cain
                abel , abel

```

avec la restriction il reste :

```

                cain , abel
                abel , cain

```

- Dans toutes ces règles, on est prié de penser :

père(_l_enfant , _le_père)

mais il serait tout aussi correct de tout réécrire en pensant :

père(_le_père , _l_enfant)

car ce n'est évidemment pas Prolog qui donne leur sens aux paramètres, le tout est de choisir une convention et de s'y tenir.

- La règle "descendant" s'appelle elle même : elle est récursive . Cela finit par s'arrêter grâce à la première clause, non récursive, du sous programme descendant. On verra plus loin d'autres cas de ce genre, car la récursivité est une arme puissante très utilisée en Prolog.

Les faits

Il suffit alors de rentrer des faits du genre :

père(quelqu'un , son_père).

mère(quelqu'un , sa_mère).

mec(quelqu'un).

ou nana(quelqu'une).

père(Marie,Joachim).

mère(Marie,Anne).

père(Jésus,Joseph).

mère(Jésus,Marie).

et, puisque nous sommes tous frères :

mère(bibi,Marie).

père(Oreste , Agamemnon).

mère(Oreste , Clytemnestre).

mec(Oreste).

père(Hermione , Ménélas).

mère(Hermione , Hélène).

nana(Hermione).

père(Tisaménos , Oreste).

mère(Tisaménos , Hermione).

On peut alors interroger cette base de données. Attention, dans tous ces exemples bien voir si les prédicats et les terminaux sont ou non avec des majuscules et avec des accents.

Ici il y a des accents et pas de majuscules dans famille.pro

>myprol famille

Vous voudriez bien savoir quels sont les grands parents de Jésus :

grand_parent(Jésus , _grand_parent)?

d'où _grand_parent = Joachim

_grand_parent = Anne

Et puis qui sont les petits enfants de Joachim :

```
grand_parent(_petit_fils, Joachim)?  
d'où      _petit_fils = Jésus  
          _petit_fils = bibi
```

```
grand_parent(_x,_y)-> gosse(_y,_z)      , gosse(_z,_x).
```

Cette règle, on vient de le voir, calcule les grand parents de jésus (_x), mais elle permet aussi de calculer les petits enfants de joachim (_z).

par soucis d'efficacité, surtout si on utilise l'indexation de règle avec hsh_coded(gosse(_,_)), on pourrait l'écrire sous la forme :

```
grand_parent(_x,_y)-> var(_z) , gosse(_y,_z)  , gosse(_z,_x).  
grand_parent(_x,_y)-> var(_x) , gosse(_z,_x)  , gosse(_y,_z)
```

Pardon, oui, vous n'en avez que faire, mais vous voudriez savoir qui est le beau-père d'Hermione ?

voilà une question qu'elle est intéressante :

```
beaudab(hermione , _x)?
```

d'où _x = c'est ce roi barbu qui s'avance, bu qui s'avance, Aga Agamemnon (air connu)

Mes biens chers frères, nous allons maintenant observer une minute de silence en pensant à tous ce que l'humanité a dû réaliser pour que nous puissions arriver à ce résultat, à savoir que Agamemnon est le beau-père d'Hermione : la théorie des nombres, l'algèbre, les tables de logarithme, la règle à calcul, l'additionneuse de Pascal, les cartons perforés des métiers à tisser, la machine de Babbage, l'algèbre de Boole, l'électricité, la chimie, les tubes à vide, les balances, le code machine binaire, les transistors, les circuits intégrés, les supports magnétiques, les assembleurs, les systèmes d'exploitations, les compilateurs, les imprimantes, le papier, les forêts, les crayons et les gommes.

C'était notre rubrique : "ben dis donc, ça grouille là d'dans"

à vous les studios.

Le repas

Cet exemple est un standard connu de tout prologophile .

```
repas(_entrée,_plat,_dessert,_cal) -> /* un repas, c'est : */
    entrée(_entrée,_cale), /* une entrée un plat et un dessert */
    plat(_plat,_calp), /* chacun ayant un nombre de calories */
    dessert(_dessert,_cald),
    _cal := _cale + _calp + _cald .

équilibré(_x,_y,_z,_c) -> /* un repas est équilibré si il ne */
    repas(_x,_y,_z,_c) , /* dépasse pas un certain nombre de calories*/
    _c <= 6 .

plat(_plat,_cal)-> viande(_plat,_cal). /* un plat, c'est de la viande */
plat(_plat,_cal)-> poisson(_plat,_cal). /* ou du poisson */

entrée(tomate,1). /* carte des entrées */
entrée(pate,2).
entrée(salade,1).

viande(steak,3). /* carte des viandes */
viande(canard_farci,5).
viande(gigot,4).

poisson(crabe,3). /* carte des poissons */
poisson(truite,2).

dessert(gateau,2). /* carte des desserts */
dessert(fruit,1).
dessert('crème au chocolat',3).
```

pour connaitre tous les repas possibles, il suffit de poser la question :

```
>myprol repas
repas(_entrée , _plat , _dessert , _calorie) ?
```

pour avoir ceux ayant exactement un nombre de calories donné :

```
repas(_entrée , _plat , _dessert , 6) ?
```

enfin, les repas équilibrés s'obtiennent par :

```
équilibré(_entrée , _plat , _dessert , _calorie) ?
```

et parmi ceux-là, ceux avec de la crème au chocolat et de la viande :

```
équilibré(_entrée,_plat,'crème au chocolat',_c) , viande(_plat,_calv)?
```

Enfer et damnation, il n'y en a pas !

L'éléphant rose

Il était une fois, Babar, Totor et Titine :

```
éléphant(Babar).
éléphant(Totor).
rose(Babar).
rose(Titine).
aime(_qqn, porcelaine) -> rose(_qqn).
magasin(porcelaine, existe).
entre(_qqn, magasin(_objet))->
    aime(_qqn, _objet),
    magasin(_objet, existe).
catastrophe(_qqn)->
    entre(_qqn, magasin(porcelaine)),
    éléphant(_qqn).
```

Posons alors la question fatidique : quel est, ou quels sont, les occurrences du tissu pragmatique ou social qui vont être stimulus, sur l'axe de la finalité du non voulu, d'une transformation irréversible de l'ordre établi dans un sens dévalorisant ?

>myprol babar
catastrophe(_qui) ?

D'où une seule solution, Titine n'étant pas une éléphante et Totor n'étant pas rose et donc n'aimant pas la porcelaine, et donc n'étant pas tenté d'entrer dans le fameux magasin :

_qui = Babar

Musique

Pour transformer le clavier azerty en piano, il suffit de faire :

```
hsh_coded(joue(,_)).
joue(q,131) .   joue(s,147) .   joue(d,165) .   joue(f,175) .
joue(g,196) .   joue(h,220) .   joue(j,247) .   joue(k,262) .
joue(z,139) .   joue(e,156) .   joue(t,185) .   joue(y,208) .
joue(u,233) .
```

```
joue(' ',0) -> nosound , exit.
```

```
joue( _autre,0) .
```

```
équivalent( _note , _fréquence) -> joue( _note , _fréquence) , / .
```

```
music ->
```

```
    repeat ,
```

```
        readkbd( _note) ,
```

```
        équivalent( _note , _fréquence) ,
```

```
        sound( _fréquence) ,
```

```
        fail .
```

On pourrait aussi baser la boucle sur la récursivité infinie (ou plutôt terminale) . Il faut noter que les langages traditionnels n'aiment évidemment pas cette méthode, et que tous les prologs ne la permettent pas. (il y a là un problème de pile de paramètre, qui augmente indéfiniment)

```
music ->
```

```
    readkbd( _note) ,
```

```
    équivalent( _note , _fréquence) ,
```

```
    sound( _fréquence) ,
```

```
    music .
```

sound fonctionnait avec turbo pascal, mais je n'ai pas cherché à le remplacer en delphi

Send more money

Or donc, par maintes foyes advinct-il que vos compaings s'en viennent vous priant de resovdre des problèmes fort subtils où les opérations de la mathématique se font sur les noms des choses vulgaires et sur les lettres d'yceux en lieux et place des nombres et des chiffres.

Ainsi disent-ils :

$$\begin{array}{r} \text{POMMES} \\ + \text{POIRES} \\ \hline = \text{FRUIT S} \end{array} \qquad \begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline = \text{MONEY} \end{array}$$

Et bien ne vous fatiguer plus inutilement, grâce à ce programme qui est déjà utilisé par des millions de gens, et que pas un seul ne voudrait échanger contre deux tableurs intégrés avec éditeur graphique déroulant et justificateur à fenêtre relationnelle au menu selon le protocole communicationnel à la norme des serveurs modulaires et compatibles avec l'interface des multifonctions du réseau partagé des souris autoconfigurables, vous garderez l'esprit libre pendant que la puissance bestiale de la machine soumise et déterministe vous fournira les solutions, oui, les vrais solutions (applaudissements nourris), et jusqu'à la dernière car il n'est pas question ici de demi-mesure (applaudissements frénétiques) oui, nous parcourrons l'arbre de résolution de Robinson jusqu'à ses dernières feuilles, pas de cut , pas de / ! (la salle est debout).

Le programme sait ajouter des mots de taille quelconques, contenant éventuellement des chiffres déjà placés. Pour cela il effectue l'opération comme vous et moi, en commençant par les lettres de droite et en propageant la retenue. Dès qu'une nouvelle lettre est utilisée, il prend un chiffre dans une liste des chiffres de 0 à 9 et le retire de la liste.

```
résoud(_lmot1 , _lmot2 , _lmotr) ->
    même_taille(_lmot1 , _lmotr , _lmot1) ,
    même_taille(_lmot2 , _lmotr , _lmot2) ,
    affiche(_lmot1 , _lmot2 , _lmotr , _mot1 , _mot2 , _motr) ,
    ajoute_mot(_mot1 , _mot2 , _motr , [0,1,2,3,4,5,6,7,8,9] , _x,0,0),
    solution(_lmot1 , _lmot2 , _lmotr) .
```

```
affiche(_lmot1 , _lmot2 , _lmotr , _mot1 , _mot2 , _motr) ->
    writeln(print) , /* afficher le problème à résoudre */
    writeln(print , ' ' , _lmot1) ,
    writeln(print , ' + ' , _lmot2) ,
    writeln(print , ' = ' , _lmotr) ,
    nb := 0 , /* nombre de solutions */
    writeln(print) ,
    revers(_lmot1 , _mot1) , /* inverser les 3 mots */
    revers(_lmot2 , _mot2) ,
    revers(_lmotr , _motr) .
```

```

même_taille(_opérande , _résultat , _opérande) ->
    length(_opérande) == length(_résultat) , / .
même_taille(_opérande , _résultat , [0 , !_opérande]) .

ajoute_mot([], [], [], _l , _l , 0 , 0) -> / .
ajoute_mot([_x,!_y] , [_a,!_b] , [_s,!_t] ,
    _chiffres_in , _chiffres_out , _retenue_in , _retenue_out) ->
    prend(_x , _chiffres_in , _chiffres_1) ,
    prend(_a , _chiffres_1 , _chiffres_2) ,
    prend(_s , _chiffres_2 , _chiffres_3) ,
    ajoute_car(_retenue_in , _x , _a , _s , _retenue_1) ,
    ajoute_mot(_y , _b , _t ,
        _chiffres_3 , _chiffres_out , _retenue_1 , _retenue_s) .

prend(_chiffre , _liste_chiffres , _liste_chiffres) -> number(_chiffre) , / .
prend(_x , [_x , !_y] , _y) .
prend(_x , [_y , !_z] , [_y , !_w]) ->
    prend(_x , _z , _w) .

ajoute_car(_r , _car1 , _car2 , _res , 0) ->      /* addition de deux caractères */
    _res := _r + _car1 + _car2 ,      /* avec propagation de la retenue */
    _res 10 , / .
ajoute_car(_r , _car1 , _car2 , _res , 1) ->
    _res := _r + _car1 + _car2 -> 10 .

solution(_lmot1 , _lmot2 , _lmotr) ->              /* afficher une solution */
    nb := nb + 1 , _nb := nb ,
    writeln(print) , writeln(print , solution , ' ' , _nb , ' : ' ) ,
    writewordsln(print , ' ' , _lmot1) ,
    writewordsln(print , ' + ' , _lmot2) ,
    writewordsln(print , ' = ' , _lmotr) ,
    beep .

```

On pose les problèmes sous la forme :

>myprol crypto

résoud([_p,_o,_m,_m,_e,_s] , [_p,_o,_i,_r,_e,_s] , [_f,_r,_u,_i,_t,_s]) ?

ou

résoud([_s,_e,_n,_d] , [_m,_o,_r,_e] , [_m,_o,_n,_e,_y]) ?

plutôt que de taper résoud(...), taper a? ou b?

Monsieur Jourdain

Pour écrire son petit billet à la belle marquise, il suffit de faire :

```
permut([_x] , [_x]) -> / .  
permut([_x , !_y] , _t) ->  
    permut(_y , _z) ,  
    insère(_x , _z , _t).
```

```
insère(_x, [] , [_x]) -> / .  
insère(_x, [_y , !_z] , [_x , _y , !_z]) .  
insère(_x, [_y , !_z] , [_y , !_t]) -> insère(_x , _z, _t) .
```

La méthode est basée sur la récurrence : si on sait permuter une liste de taille N, on sait permuter celle de taille N+1 en commençant par lui enlever un élément, en permutant le reste puis en insérant l'élément un peu partout.

>myprol crypto

```
permut(['belle marquise','vos beaux yeux','me font','mourir','d amour'],_x)?  
ou a(_x) ?
```

Attention, il y a $5*4*3*2=120$ solutions ! Molière n'en a retenu que quelques-unes.

Pour voir, essayez donc de faire la même chose en basic, ou même en pascal : permuter les N premiers éléments d'un tableau dimensionné à Nmax, N doit être variable, il faut par exemple le lire au clavier.(durée : 3 heures , coefficient : 4).

Une autre méthode pour permuter consiste à extraire un élément, à permuter le reste et à remettre l'élément devant :

```
permutation( [] , []) -> / .  
permutation(_x , [_y , !_z]) ->  
    select(_y , _x , _t) ,  
    permutation(_t , _z) .  
  
select(_x , [_x , !_reste] , _reste) .  
select(_x , [_y , !_suite] , [_y , !_fin]) ->  
    select(_x , _suite , _fin) .
```

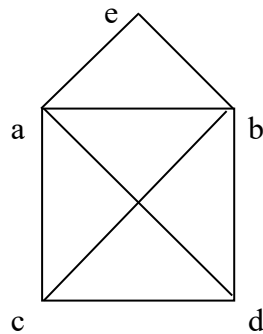
le sous-programme **select** permet d'extraire un élément _x d'une liste, et rend ce qui reste.

Dessine-moi un mouton

Il s'agit de le faire sans lever le crayon.

On représente le dessin par la liste des segments qui le compose :

[a-e, e-b, a-b, a-d, a-c, b-c, b-d, c-d]



et on attend comme résultat une liste de sommets :

a, b, e, a, ...

```
dessiner([], [_x]) -> / .
```

```
dessiner(_d, [_x, _y, !_h]) ->  
    extraire(_d, _x-_y, _r) ,  
    dessiner(_r, [_y, !_h]) .
```

```
extraire([_a-_b, !_y], _a-_b, _y).  
extraire([_a-_b, !_y], _b-_a, _y).  
extraire([_x, !_y], _z, [_x, !_r]) -> extraire(_y, _z, _r).
```

extraire sert à retirer de la liste un des segments, que l'on peut parcourir dans un sens ou dans l'autre.

d'où :

```
dessiner([a-e,e-b,a-b,a-d,a-c,d-b,d-c,b-c] , _resultat) ?
```

il y a 88 solutions, 44 allant de d à c et 44 de c à d.

Le mouton est dans sa boîte, évidemment.

Générateur de dictionnaire

A partir des significations de quelques préfixes et de quelques suffixes on va générer de nouveaux mots :

préfixe(1, logo , jeux).	préfixe(2, dermato, peau).
préfixe(3, auto , 'par lui même').	préfixe(4, icono, 'en dessinant').
préfixe(5, ultra , 'au dela').	préfixe(6, tachy, vitesse).
préfixe(7, vice , 'à la place de').	préfixe(8, pedo , pied).
suffixe(1, claste , casseur).	suffixe(2, able , 'susceptible de').
suffixe(3, oir , 'instrument qui sert à').	suffixe(4, graphe , 'qui écrit').
suffixe(5, pede , 'qui prend son pied').	

gènère ->

```
_praleat := random(8) , /* un nombre compris entre 1 et 8 */
préfixe(_praleat , _préfixe , _senspréfixe ) ,
_sualeat := random(5) ,
suffixe(_sualeat , _suffixe , _senssuffixe ) ,
write(_préfixe,_suffixe) ,
writeln(' : ' , _senssuffixe , ' ' , _senspréfixe).
```

d'où pour activer ce générateur de mots :

>myprol genmot

gènère ?

logographe : qui écrit des jeux
iconopede : qui prend son pied en dessinant
pedoclaste : casse pied

Il faut bien voir que l'appel de la règle préfixe (celui de suffixe aussi) entraîne un balayage complet jusqu'à correspondance du 1er paramètre avec le nombre aléatoire tiré , puis jusqu'à la fin : c'est une catastrophe ! pour faire mieux, il faut rajouter l'accès direct dans prolog par l'appel de la primitive index qui spécifie que pour la règle suivante, on se contentera d'analyser le _Xème élément. Oyez braves gens le hurlement profond des puristes le soir au fond des bois. On peut alors supprimer le 1er paramètre des faits préfixe et suffixe :

gènère ->

```
_praleat := random(8) , /* générateur de nombre aléatoire */
index(_praleat) , /* accès direct pour la prochaine règle */
préfixe(_préfixe , _senspréfixe ) ,
_sualeat := random(5) ,
index(_sualeat) , /* nouvel accès direct */
suffixe(_suffixe , _senssuffixe ) ,
write(_préfixe,_suffixe) ,
writeln(' : ' , _senssuffixe , ' ' , _senspréfixe).
```

Contre tout couple préfixe-suffixe dûment identifié et certifié par l'autorité lurique incompétente, le cachet de la poste faisant foi, il sera-t-envoyé au préalable et à propos, le tout sous huitaine et pli cacheté, une enveloppe in 8 et deux timbres pour la réponse.

Javanais, loucherbem, largonji, redegueu

```
voyelle(a).  
voyelle(e).  
voyelle(i).  
voyelle(o).
```

Cave savous pravogravamme adaguanadagualydygyse toudouguous lesdesgues
élémentuches de la listaille, lonbem à lavoissem.

```
trans ( _lettre ) ->  
    voyelle( _lettre ) ,  
    write( av , _lettre ) , / .  
trans ( _lettre ) -> write( _lettre ) , / .
```

```
javanais([]) -> / .  
javanais( [ _première , !_reste ] ) ->  
    trans( _première ) ,  
    javanais( _reste ) .
```

pour activer ce traducteur javanais :

```
>myprol javanais
```

```
javanais("e pericoloso sporgersi") ?
```

```
ave paveravicavolavosavo spavorgaversavi !
```

Jeux de mots

En ajoutant deux mots ou groupes de mots l'un à l'autre on obtient un mot composé, mais si la fin du premier mot est égal au début du deuxième, on a un jeu de mots :

```
hsh_coded(mot(_)).
mot("how do you do").
mot("tete de veau").
mot("savons de marseille").
mot("selle de cheval").
mot("pas mal and you").
mot("c est lamentable").
mot("nous le savons").
mot("douglas fairbank").
mot("table de chevet").
mot("estete").
mot("youl bruner").

jeux_de_mot_svp ->
    mot(_début) ,
    append( _deb1 , _commun , _début) ,
    length( _commun ) > 1 , _deb1 \= [] ,
    mot(_fin) ,
    append( _commun , _fin2 , _fin) , _fin2 \= [] ,
    putln( _deb1 , _commun , _fin2 ) .
```

```
append( [] , _liste , _liste) .
append( [ _début , !_z ] , _t , [ _début , !_fin ] ) -> append( _z , _t , _fin) .
```

Au passage, on remarquera le sous-programme **append** qui permet ici d'obtenir toutes les manières de couper une liste en deux.

```
>myprol jeuxmot
jeux_de_mot_svp ?
    how do you douglas fairbank
    pas mal and youl bruner
    c'est lamentable de chevet
    nous le savons de marseille
    esthete de veau
```

Le problème des 8 reines

Il s'agit de placer 8 reines sur un échiquier sans qu'elles se mangent. il n'y en a bien évidemment qu'une par ligne et qu'une par colonne. une solution se représente par une liste de 8 nombres : le Nème nombre est la position d'une reine dans la Nème colonne.

Il s'agit donc d'ordonner la liste des positions (liste de numéro de ligne) : [1,2,3,4,5,6,7,8]

On procède comme suit :

- prendre une des positions de la liste (grâce à "**select**") et mettre une reine sur la première colonne, à la ligne correspondant à la valeur de la position extraite.
- prendre une position restante et placer de même une dame sur la colonne suivante.
- continuer ainsi jusqu'à l'épuisement de la liste des 8 positions.

Tel que, cet algorithme ne tient compte que des déplacements verticaux et horizontaux des reines : il placerait correctement des tours.

Pour tenir compte du déplacement diagonal, lorsque l'on rajoute une reine, sa position dans la colonne doit être différente de 1 de celle de la précédente, de 2 de celle d'avant , de 3 de ... et c'est ce que fait le prédicat "attaque" pour éliminer ces positions.

```
reines(_resu) ->
    placer([1,2,3,4,5,6,7,8], [], _resu) .

placer([], _placées, _placées) -> / .
placer(_nonplacées, _placées, _resu) ->
    select(_reine, _nonplacées, _reste) ,
    bienplacée(_reine, _placées) ,
    placer(_reste, [_reine, !_placées], _resu) .

select(_x, [_x, !_reste], _reste) .
select(_x, [_y, !_suite], [_y, !_fin]) ->
    select(_x, _suite, _fin) .

attaque(_reine, _placées) ->
    attaque(_reine, 1, _placées) .

attaque(_reine, _depl, [_y, !_suite]) ->
    _reine := _y - _depl, / .
attaque(_reine, _depl, [_y, !_suite]) ->
    _reine := _y + _depl, / .
attaque(_reine, _depl, [_y, !_suite]) ->
    _depl1 := _depl + 1 ,
    attaque(_reine, _depl1, _suite) .

bienplacée(_reine, _placées) -> /* bienplacée c'est la négation de attaque */
    attaque(_reine, _placées), / , fail .
bienplacée(_reine, _placées).

>myprol reines
reines(_x) ?
```

Pour un échiquier de 8 , il y a 92 solutions, d'où il faudrait retirer les semblables par symétries et rotations., du coup il reste 12 solutions vraiment différentes.

Le coloriage d'une carte

On (Kenneth Appel et Wolfgang Haken en 1976) démontre, par ordinateur, et c'est la première démonstration de ce type, que 4 couleurs suffisent pour colorier une carte.

On définit pour chaque couleur les autres couleurs possibles pour les régions voisines :

autre(bleu , vert).	autre(bleu , rouge).	autre(bleu , jaune).	autre(vert , bleu).
autre(vert , rouge).	autre(vert , jaune).	autre(rouge , vert).	autre(rouge , bleu).
autre(rouge , jaune).	autre(jaune , vert).	autre(jaune , rouge).	autre(jaune , bleu).

Pour chaque province, on donne les contraintes concernant les couleurs des provinces voisines, qui doivent être différentes :

```
colorier(_bretagne , _normandie , _maine , _anjou , _poitou , _touraine) ->
    autre(_bretagne, _normandie),
    autre(_bretagne, _maine),
    autre(_bretagne, _anjou),
    autre(_bretagne, _poitou),

    autre(_normandie, _bretagne),
    autre(_normandie, _maine),

    autre(_maine, _bretagne),
    autre(_maine, _anjou),
    autre(_maine, _touraine),

    autre(_anjou, _bretagne),
    autre(_anjou, _poitou),
    autre(_anjou, _touraine),

    autre(_poitou, _bretagne),
    autre(_poitou, _anjou),
    autre(_poitou, _touraine),

    autre(_touraine, _maine),
    autre(_touraine, _anjou),
    autre(_touraine, _poitou) .
```

et voila, le tour est joué avec ces simples déclarations :

```
>myprol couleur
colorier(_bretagne , _normandie , _maine , _anjou , _poitou , _touraine) ?
```

Une autre solution moins verbeuse, mais moins élégante :

chaque région se représente par son nom, sa couleur et la liste des couleurs de ses adjacents :

```
region(bretagne , _bretagne , [ _normandie , _maine , _anjou , _poitou])
```

la carte est une liste de régions. il s'agit de colorier toutes les régions les unes après les autres. Lorsque l'on colorie une région,

- si ce n'est déjà fait on choisit une couleur pour cette région (grâce à **select**)
- on choisit une autre couleur pour chacun de ses adjacents ou, si le choix a déjà été fait pour ces régions, on vérifie que c'est une autre couleur.

```
colorier([], _couleurs) -> / .
```

```
colorier([_region , !_reste_regions] , _couleurs) ->  
    colorier_region(_region , _couleurs) ,  
    colorier(_reste_regions , _couleurs) .
```

```
colorier_region( region(_nom , _couleur , _adjacents) , _couleurs) ->  
    select(_couleur , _couleurs , _reste_couleur) ,  
    inclus(_adjacents , _reste_couleur) .
```

la liste des régions et leurs adjacents:

```
carte(france , [  
    region(bretagne , _bretagne , [ _normandie , _poitou , _maine , _anjou]) ,  
    region(normandie , _normandie , [ _bretagne , _maine]) ,  
    region(maine , _maine , [ _bretagne , _normandie , _anjou , _touraine]) ,  
    region(anjou , _anjou , [ _bretagne , _poitou , _maine , _touraine]) ,  
    region(poitou , _poitou , [ _bretagne , _anjou , _touraine]) ,  
    region(touraine , _touraine , [ _anjou , _poitou , _maine])  
]) .
```

la liste des couleurs :

```
couleurs([bleu , vert , rouge , jaune]) .
```

```
colorie(_pays , _carte) ->  
    couleurs(_couleurs) ,  
    carte(_pays , _carte) ,  
    colorier(_carte , _couleurs) .
```

colorie(france , _carte) ?

Carré magique

```
magique([_a,_b,_c,_d,_e,_f,_g,_h,_i]) ->
  permut([1,2,3,4,5,6,7,8,9] , [_a,_b,_c,_d,_e,_f,_g,_h,_i]),
  15==_a+_b+_c,
  15==_d+_e+_f,
  15==_g+_h+_i,
  15==_a+_d+_g,
  15==_b+_e+_h,
  15==_c+_f+_i,
  15==_a+_e+_i,
  15==_c+_e+_g.
```

```
permut([_x] , [_x]) -> / .          /* permutation */
permut([_x , !_y] , _t) ->
  permut(_y , _z) ,
  insere(_x , _z , _t).
```

```
insere(_x, [] , [_x]) -> / .
insere(_x, [_y , !_z] , [_x , _y , !_z]) .
insere(_x, [_y , !_z] , [_y , !_t]) -> insere(_x , _z , _t) .
```

Cette solution utilise la force brute : on analyse toutes les possibilités.
une deuxième solution, plus sélective :

```
carré_magique([_a,_b,_c,_d,_e,_f,_g,_h,_i]) ->
  select(_a,[1,2,3,4,5,6,7,8,9] , _ra) ,
  select(_b,_ra,_rb),          /* chaque select réduit la plage de choix */
  select(_c,_rb,_rc),
  15==_a+_b+_c,                /* la contrainte est vérifiée plus tôt */
  select(_d,_rc,_rd),
  select(_e,_rd,_re),
  select(_f,_re,_rf),
  15==_d+_e+_f,
  select(_g,_rf,_rg),
  select(_h,_rg,[_i]),
  15==_g+_h+_i,
  15==_a+_d+_g,
  15==_b+_e+_h,
  15==_c+_f+_i,
  15==_a+_e+_i,
  15==_c+_e+_g.

select(_x , [_x , !_fin] , _fin) .
select(_x , [_y , !_suite] , [_y , !_fin]) ->
  select(_x , _suite , _fin).
```

>myprol

carré_magique(_x) ?

index kwic

Etant donnés une liste de titre de livres, on désire créer un index de tous les mots clés de ces titres en conservant pour chaque mot son environnement dans le titre.

```
hsh_coded(kwic(_)).
```

```
kwic ->
```

```
    titre(_titre) ,  
    traiter_titre(_titre , _kwic) ,  
    assert(kwic(_kwic) , []).
```

```
kwic ->
```

```
    findall_diff(_mot , kwic([_mot , !_fin]) , _liste) ,  
    sort(_liste , _liste_triee) ,  
    edite(_liste_triee).
```

```
edite([_mot , !_fin]) ->
```

```
    writewordsln(_mot) ,  
    kwic([_mot , !_reste]) ,  
    write('      '),  
    writewordsln(_reste) ,  
    fail.
```

```
edite([_mot , !_fin]) ->
```

```
    edite(_fin) .
```

```
traiter_titre(_titre , _kwic) ->
```

```
    append(_début , [_mot , !_fin] , _titre) ,  
    utile(_mot) ,  
    append([_mot , !_début] , [_mot , !_fin] , _kwic).
```

```
append( [] , _liste , _liste) .
```

```
append( [_début , !_fin] , _liste , [_début , !_reste] ) ->
```

```
    append(_fin , _liste , _reste) .
```

```
utile(_mot) -> inutile(_mot) , / , fail.
```

/*utile est la négation de inutile /

```
utile(_mot).
```

```
hsh_coded(inutile(_)).
```

```
inutile(le).      inutile(la).      inutile(les).      inutile(un).      inutile(une).
```

```
inutile(des).     inutile(l).      inutile(de).      inutile(d).      inutile(a).
```

```
inutile(s).       inutile(se).     inutile(en).      inutile(est).     inutile(et).
```

```
inutile(au).
```

```
titre([le,rouge,et,le,noir]).
```

```
titre([tintin,en,amérique]).
```

```
titre([tintin,au,congo]).
```

```
titre([noir,aujourd'hui]).
```

```
titre([les,précieuses,ridicules]). titre([de,la,démocratie,en,amérique]).
```

```
>myprol kwic
```

```
kwic ?
```

Récursion

Il peut s'avérer que, du fait des empilements successifs dans les différentes piles lors de l'exécution des modules, il soit impossible d'exécuter :

programme -> module1 , module2 .

Ce programme est en effet de ce point de vue équivalent en pascal à :

```
procedure programme;
  procedure module1;
    procedure module2;
      begin (* code de module2 *)
        ...
      end;
    begin (* code de module1 *)
      ...
    module2;
    end;
  begin (* code de programme *)
    module1;
  end;
```

Il suffit alors de le transformer en utilisant les variables globales, les fichiers ou les directives assert et retract pour la passation de paramètres entre les deux modules. Cette transformation est immédiate si un module n'a que des paramètres d'entrée (cas des éditions) .

```
programme -> module1 , fail .
programme -> module2 , fail .
programme.
```

Ce qui équivaut à l'écriture pascal :

```
procedure programme;
  procedure module1;
    begin
    end;
  procedure module2;
    begin
    end;
  begin (* code de programme *)
    module1;
    module2;
  end;
```

Une autre formulation de la même idée est :

```
module1(_paramètres_entrés) -
  traitement_complicé ,
  fail .
module1(_paramètres_entrés) .
```

Récursion quand tu nous tient

J'en était sûr, encore un qui va nous parler de la factorielle :

```
fact_1(1, 1) - / .  
fact_1(_x, _y) -  
    _z := _x - 1 ,  
    fact_1(_z, _t) ,  
    _y := _x * _t .
```

une autre version , avec récursivité en dernier à droite :

```
fact(0, _f, _f) - / .  
fact(_n, _x, _f) -  
    _m := _n - 1 ,  
    _y := _x * _n ,  
    fact(_m, _y, _f) .
```

fact(_n, 1, _fact) ?

et une dernière pour la route :

```
fact(_n, _fn, _n, _fn) - / .  
fact(_x, _fx, _n, _fn) -  
    _nx := _x + 1 ,  
    _fnx := _nx * _fx ,  
    fact(_nx, _fnx, _n, _fn) .
```

fact(0, 1, _n, _fact) ?

Attention, l'ordre des règles est important, les inverser conduit à une boucle infinie. D'autre part la récursivité, c'est bien mais il ne faut pas en abuser : faire une boucle for i := 1 to 100000 basée sur la récursivité, c'est courir à une catastrophe certaine, pour la raison évoquée dans le paragraphe précédent. Ici, les variables intermédiaires `_z` et `_t` s'empilent dans l'environnement pendant chaque étape du calcul de la factorielle, comme dans tous les langages supportant la récursivité. Mais ils y restent aussi à la fin, car l'interpréteur doit pouvoir gérer les éventuels retours arrières ultérieurs. Et donc, bien que dans certains cas l'interpréteur se rende compte que ces retours arrières sont impossibles et qu'il peut ne pas empiler, on a tout intérêt à programmer :

```
fact_2(_x, _y) ->  
    fact_1(_x, _y) ,          /* méthode du paragraphe précédent */  
    resu := _y ,  
    fail.                    /* pour récupérer les empilements */  
fact_2(_x, _y) ->  
    _y := resu.              /* passer par une variable global ou un assert */
```

```

ou    fact_3(_x , _y) ->      /* pour ne pas empiler et donc */
        i := 1 ,                /* ne pas avoir à récupérer */
        resu := 1 ,            /* élémentaire mon cher Watson */
        repeat ,
            resu := resu * i,
            i := i + 1 ,
            i < _x ,
        _y := resu , / .

ou    fact_4(_x , _y) ->      /* comme en pascal */
        resu := 1,
        for(i , 1 , _x),
            resu := resu * i,
            ffor,
        _y := resu.

```

Et puis d'abord, vous avez mal lu la doc, il y a un prédicat tout fait qui calcule la factorielle y d'un nombre x (par la dernière méthode) :

```
_y := fact(_x).
```

La suite de Fibonacci

```
fib(1,1)->/.  
fib(2,1)->/.  
fib(_n,_x)->  
    _n1:=_n-1,  
    _n2:=_n-2,  
    fib(_n1,_x1),  
    fib(_n2,_x2),  
    _x:=_x1+_x2.
```

Cette première méthode de calculer la suite n'est pas efficace, puisque on appelle "fib" deux fois pour chaque nombre inférieur à n. il vaut mieux faire :

```
fib2(_n,_x)->forward_fib(_n,2,1,1,_x).  
  
forward_fib(_n,_m,_f1,_f2,_x)->  
    _m<_n,/,  
    _m1:=_m+1,  
    _f3:=_f1+_f2,  
    forward_fib(_n,_m1,_f2,_f3,_x).  
forward_fib(_n,_m,_f1,_f2,_f2).
```

Les tours de Hanoï

Déplacer N disques d'une pile de départ à une pile d'arrivée en se servant d'une pile vide revient, selon le bon vieux principe de la récurrence, à savoir déplacer N-1 disques, et à en déduire le déplacement de N disques. Comme le déplacement d'un disque est évident, le problème est alors résolu.

La solution consiste donc à savoir passer de N-1 disques à N disques :

- déplacer N-1 disques de la pile de départ vers la pile d'aide en se servant de la pile d'arrivée, qui est vide.
- mettre à sa place le disque restant.
- déplacer les N-1 disques de la pile d'aide vers la pile d'arrivée en se servant de la pile de départ, qui est maintenant vide ou contient des disques plus grands.

```
déplacer(1 , _pile_départ , _pile_arrivée , _pile_aide) -  
    putln('déplacer ', _pile_départ , ' sur ', _pile_arrivée) , / .  
déplacer(_nb_disque , _pile_départ , _pile_arrivée , _pile_aide) -  
    _nb_souspile := _nb_disque-1 ,  
    déplacer(_nb_souspile , _pile_départ , _pile_aide , _pile_arrivée) ,  
    déplacer(1 , _pile_départ , _pile_arrivée , _pile_aide) ,  
    déplacer(_nb_souspile , _pile_aide , _pile_arrivée , _pile_départ) .
```

Il suffit alors de poser la question :

```
>myprol hanoi  
déplacer(4 , a , b , c) ?
```

pour savoir comment déplacer 4 disques de A vers B en utilisant C .

Ce programme est éminemment récursif et il fait très rapidement exploser les tables de l'interpréteur, si celui-ci ne gère pas la récursivité avec les améliorations que l'on vient d'évoquer. Auquel cas il suffit (après avoir changé le dernier "." en ",") de rajouter :

```
fail .  
déplacer(_nb_disque , _pile_départ , _pile_arrivée , _pile_aide) .  
  
>myprol hanoi  
hanoi ?
```

Expressions algébriques

Nous allons définir un ensemble de prédicats pour manipuler des expressions mathématiques de manière symbolique. Par manipulation symbolique, on veut dire que l'on ne connaît pas les valeurs des variables, mais que l'on veut quand même simplifier ces expressions, les dériver ou les intégrer et résoudre des équations, en faisant référence aux noms des variables.

Une expression mathématique est constituée d'éléments liés par les opérateurs mathématiques classiques (+ , - , * , / , ^). Chaque élément peut être une constante (symbolique ou non), une variable, une expression entre parenthèse ou une fonction mathématique portant sur une expression (sinus, logarithme, ...).

Les polynômes sont des expressions simples, à une seule variable..

L'ensemble de ces prédicats algébriques se classent en :

- manipulation de polynômes
- simplification d'expression
- résolution d'équation
- dérivation
- intégration

calcul symbolique

Cet exemple, de taille respectable est complètement déclaratif et s'écrit donc dans un prolog très pur, presque sans utiliser de prédicats prédéfinis.

Les sous-programmes « dérivée » et "intégrale" ont trois paramètres : l'expression à dériver ou intégrer, la variable par rapport à laquelle on dérive ou intègre et le résultat. Chaque règle de ces 2 sous-programmes correspond à une formule mathématique de dérivation ou d'intégration, c'est ce qui rend ce système très pur car fondamentalement déclaratif. Par exemple, une règle montre que l'intégrale d'une constante a par rapport à X c'est : $a * X$. (dans tout ce qui suit, nous négligerons la constante d'intégration).

intégrale(_a, _x, _a * _x) -> const(_a) .

et une autre stipule que la dérivée d'une constante c'est 0

dérivée(_a, _x, 0) -> const(_a).

Au fait, c'est quoi une constante ? c'est soit un nombre ou un identificateur différent des lettres x, y et z, qui sont réputées être des variables, soit une fonction de constantes, les quatre opérations étant des cas particuliers de fonctions : $a+b$ est la notation de $+(a,b)$.

```
const(_x) -> constante(_x) , _x \= x , _x \= y , _x \= z , / .
const(_fonction(_x , _y)) -> const(_x) , const(_y) .
```

au passage, on remarque une variable (_fonction) en position de fonction.

Une autre règle montre que la dérivée du produit d'une constante a par une fonction u, c'est le produit de la constante par la dérivée de la fonction.

```
dérivée(_a * _u , _x , _a * _iu) ->
    const(_a) ,
    dérivée(_u , _x , _iu) .
```

Mais si, pour connaître la dérivée de $2 * x^2$, on posait la question :

```
dérivée(2 * x^2 , x , _i)?
```

vu que dérivée(x^2 , x, _r) doit rendre : $_r = 2 * x$, on obtiendrait comme réponse :

```
_i = 2 * 2 * x
```

ce qui n'est pas faux, mais lourd. il faut donc ajouter un mécanisme de simplification de sorte que constante * constante se simplifie, que $1 * u$ devienne u, que $0 + u$ devienne u, que les billets de mille pleuvent par paquets de 10, etc... Pour cela on déclare quelques règles de simplification:

```
produit(_const1, _const2, _résultat) ->
    number(_const1) , number(_const2),
    _résultat := _const1 * const2 , / .
produit(1, _a, _a) -> / .
produit(_a, 1, _a) -> / .
...
produit(_a, _b, _a * _b). /* pas de simplification */
```

```
et de même avec les autres opérations +, -, / , ^
somme(0, _a, _a) -> / .    somme(_a, 0, _a) -> / .
...
```

la règle "dérivée" devient alors :

```
dérivée(_a * _u , _x , _i) ->
    const(_a) ,
    dérivée(_u , _x , _iu) ,
    produit(_a , _iu , _i) .
```

D'où $_i = 4 * x$ ouf, on respire.

Ces simplifications ne règlent pas tous les cas :

ainsi $2 + y + 3$ passe au travers des gouttes et ne rend pas $5 + y$ lorsque l'on ajoute x à $y + z$, il faut d'abord regarder si $x + y$ se simplifie

```
somme(_x , _y + _z , _d) ->
    csomme(_x , _y , _s),          somme(_s , _z , _d) , / .
```

et si ça ne marche pas avec $x + y$, on a encore un joker avec $x + z$:

```
somme(_x , _y + _z , _d) ->
    csomme(_x , _z , _s),          somme(_s , _y , _d) , / .
    somme(_x , _y + _z , _x + _y + _z). /* ca ne se simplifie pas */
```

la différence entre somme et csomme étant que csomme ne marche que si une simplification est possible, somme n'échoue jamais et rend soit une expression simplifiée soit l'expression de départ (ici : $_x + _y + _z$).

voir les règles de simplification après les règles de dérivation et d'intégration.

Règles de dérivation

on déclare d'abord les dérivées des fonctions classiques. Chaque règle se termine par un « / » signifiant que la solution étant trouvée, il ne faut pas essayer les règles suivantes.

Règles générales de dérivation sur les 6 opérations

```
dérivée(_x, _x, 1) -> / .
dérivée(_const, _x, 0) -> const(_const), / .
dérivée(_u + _v, _x, _d) -> /* dérivée d'une somme */
    dérivée(_u, _x, _du),
    dérivée(_v, _x, _dv),
    somme(_du, _dv, _d), / .
dérivée(_u - _v, _x, _d) -> /* dérivée d'une différence */
    dérivée(_u, _x, _du),
    dérivée(_v, _x, _dv),
    différence(_du, _dv, _d), / .
dérivée(_cst*_u, _x, _d) -> /* dérivée du produit par une constante */
    const(_cst), /,
    dérivée(_u, _x, _du),
    produit(_cst, _du, _d).
dérivée(_u/_cst, _x, _d) ->
    const(_cst), /,
    dérivée(_u, _x, _du),
    quotient(_du, _cst, _d).
dérivée(_u * _v, _x, _d) -> /* dérivée d'un produit */
    dérivée(_u, _x, _du),
    dérivée(_v, _x, _dv),
    produit(_u, _dv, _da),
    produit(_v, _du, _db),
    somme(_da, _db, _d), / .
dérivée(_cst/_u, _x, _d) -> /* dérivée de l'inverse */
    const(_cst),
    dérivée(_u, _x, _du), /,
    puissance(_u, 2, _u2),
    produit(_cst, _du, _ddu),
    quotient(_ddu, _u2).
dérivée(_u / _v, _x, _d) -> /* dérivée d'un quotient */
    dérivée(_u, _x, _du),
    dérivée(_v, _x, _dv),
    produit(_u, _dv, _da),
    produit(_v, _du, _db),
    différence(_db, _da, _dc),
    puissance(_v, 2, _v2),
    quotient(_dc, _v2, _d), / .
dérivée(_u^_n, _x, _d) -> /* dérivée d'une puissance */
    const(_n), dérivée(_u, _x, _du),
    différence(_n, 1, _n1),
    puissance(_u, _n1, _da),
    produit(_n, _da, _d), / .
dérivée(-_u, _x, _d) ->
    dérivée(_u, _x, _du), moins(_du, _d), / .
```

dérivées de quelques fonctions classiques

```

dérivée(sin(_u) , _x , _d) ->      /* dérivée du sinus */
    dérivée(_u , _x , _du) ,
    produit(_du, cos(_u) , _d) , / .
dérivée(cos(_u) , _x , _d) ->      /* dérivée du cosinus */
    dérivée(_u , _x , _du) ,
    produit(_du , -sin(_u) , _d) , / .
dérivée(tg(_u) , _x , _d) ->      /* dérivée de la tangente */
    dérivée(_u , _x , _du) ,
    quotient(_du , cos(_u)^2 , _d) , / .
dérivée(cotg(_u) , _x , _d) ->      /* dérivée de la cotangente */
    dérivée(_u , _x , _du),
    moins(_du, _ddu),
    quotient(_ddu , sin(_u)^2 , _d) , / .
dérivée(arccos(_u),_x,_d)->      /* dérivée de arccos */
    dérivée(_u, _x, _du),
    moins(_du, _ddu),
    puissance(_u,2,_u2),
    différence(1,_u2,_a),
    quotient(_ddu,sqrt(_a),_d),/.      /* _ddu peut valoir 1 et se simplifie */
dérivée(arsin(_u),_x,_d)->      /* dérivée de arcsin */
    dérivée(_u, _x, _du),
    puissance(_u,2,_u2),
    différence(1,_u2,_a),
    quotient(_du,sqrt(_a),_d),/.
dérivée(arctg(_u),_x,_d)->      /* dérivée de arctg */
    dérivée(_u, _x, _du),
    puissance(_u,2,_u2),
    somme(1,_u2,_a),
    quotient(_du,_a,_d),/.
dérivée(ln(_u) , _x , _d) ->      /* dérivée du logarithme */
    dérivée(_u , _x , _du) ,
    quotient(_du , _u , _d) , / .
dérivée(sqrt(_u) , _x , _d) ->      /* dérivée de racine carrée */
    dérivée(_u , _x , _du) ,
    quotient(_du, sqrt(_u),_d), / .
dérivée(exp(_u) , _x , _d) ->      /* dérivée de l'exponentielle */
    dérivée(_u , _x , _du) ,
    produit(_du , exp(_u) , _d) , / .
dérivée(ch(_u),_x,_d)->      /* dérivées des fonctions hyperboliques */
    dérivée(_u, _x, _du),
    produit(_du, sh(_u),_d), / .
dérivée(sh(_u),_x,_d)->
    dérivée(_u, _x, _du),
    produit(_du, ch(_u),_d),/.
dérivée(th(_u),_x,_d)->
    dérivée(_u, _x, _du),
    quotient(_du, ch(_u)^2,_d),/.

```

```

dérivée(argsh(_u),_x,_d)->
  dérivée(_u,_x,_du),
  puissance(_u,2,_u2),
  quotient(_du, sqrt(1+_u2),_d),/.
dérivée(argch(_u),_x,_d)->
  dérivée(_u,_x,_du),
  puissance(_u,2,_u2),
  quotient(_du, sqrt(-1+_u2),_d),/.
dérivée(argth(_u),_x,_d)->
  dérivée(_u,_x,_du),
  puissance(_u,2,_u2),
  quotient(_du,1-_u2,_d),/.

```

dérivée d'une fonction de fonction

```

dérivée(_f(_u) , _x , _d) ->      /* f(u) -> u' * f'(u) */
  _u \= _x ,
  dérivée(_f(y) , y , _df) ,
  dérivée(_u , _x , _du) ,
  remplacer(_df, _u, _rf),          /* il faut remplacer y par (_u) dans _df */
  produit(_du , _rf , _d) , / .

```

```

remplacer(y,_u,_u) -> /.
remplacer(_fonc(y),_u,_fonc(_u) -> /.
remplacer(_fonc(_a),_u,_fonc(_ra)) -> remplacer(_a,_u,_ra),/.
remplacer(_fonc(_a,_b),_u,_fonc(_ra,_rb)) ->
  remplacer(_a,_u,_ra),
  remplacer(_b,_u,_rb),/.
remplacer(_a,_u,_a).

```

et cette règle qui doit être la dernière, quand on a tout essayé ;

```

dérivée(_u , _x , dérivée_inconnue(_u)).    /* dérivée inconnue */

```

Règles d'intégration

On déclare de même les intégrales des fonctions classiques :

règles d'intégration des 6 opérations

```
intégrale(_a , _x , _d) -> /* intégrale d'une constante */
    const(_a) ,
    produit(_a , _x , _d) , / .
intégrale(_const * _u , _x , _i) -> /* produit par une constante */
    const(_const) ,
    intégrale(_u , _x , _iu) ,
    produit(_const , _iu , _i) , / .
intégrale(_u * _const , _x , _i) ->
    const(_const) ,
    intégrale(_u , _x , _iu) ,
    produit(_const , _iu , _i) , / .
intégrale(_u / _const , _x , _i) ->
    const(_const) ,
    intégrale(_u , _x , _iu) ,
    quotient(_iu , _const , _i) , / .
intégrale(_u + _v , _x , _i) -> /* intégrale d'une somme */
    intégrale(_u , _x , _iu) ,
    intégrale(_v , _x , _iv) ,
    somme(_iu , _iv , _i) , / .
intégrale(_u - _v , _x , _i) -> /* intégrale d'une différence */
    intégrale(_u , _x , _iu) ,
    intégrale(_v , _x , _iv) ,
    différence(_iu , _iv , _i) , / .

intégrale(-_u , _x , _i) ->
    intégrale(_u , _x , _iu) ,
    moins(_iu , _i) , / .
```

Intégrales de quelques fonctions classiques

```
intégrale(_x , _x , 0.5 * _x^2) -> / .
intégrale(sin(_x) , _x , -cos(_x)) -> / .
intégrale(cos(_x) , _x , sin(_x)) -> / .
intégrale(tg(_x) , _x , -ln(cos(_x)) -> / .
intégrale(cotg(_x) , _x , ln(sin(x))) -> / .
intégrale(exp(_x) , _x , exp(_x)) -> / .
intégrale(ln(_x) , _x , _x*ln(_x)-_x) -> / .
intégrale(ch(_x) , _x , sh(_x)) -> / .
intégrale(sh(_x) , _x , ch(_x)) -> / .
intégrale(th(_x) , _x , ln(ch(_x)) -> / .
intégrale(arcsin(_x) , _x , _x*arcsin(_x)+sqrt(1-_x^2)) -> / .
intégrale(arccos(_x) , _x , _x*arccos(_x)-sqrt(1-_x^2)) -> / .
intégrale(arctg(_x) , _x , _x*arctg(_x)-ln(_x^2+1)/2) -> / .
intégrale(_cst^_x , _x , (_cst^_x)/ln(_cst)) -> const(_cst) , / .
intégrale(1/_x , _x , ln(_x)) -> / .
intégrale(_x^-1 , _x , ln(_x)) -> / .
```

```

intégrale(_cst / _x , _x , _cst * ln(_x)) -> const(_cst) , / .
intégrale(_x^_n , _x , _d) ->
    const(_n) , / ,
    somme(_n , 1 , _m) ,
    produit(_m , x^_m , _d) .
intégrale(_cst / _x^_n , _x , _d) ->
    const(_n) , const(_cst) ,
    difference(1 , _n , _p) ,
    quotient(_cst , _p , _m) ,
    produit(_m , _x^_p , _d) , / .

```

On reconnait la dérivée d'une fonction de fonction

dans certains cas, on reconnait, à une constante près, une certaine forme dans l'expression, et on peut en déduire l'intégrale :

$u' * u$	$u^2/2$
$u' * u^n$	$u^{n+1}/n+1$
u'/u	$\ln(u)$
u'/u^2	$-1/u$
$cst/\sqrt{u}$	$2*cst*\sqrt{u}$

```

intégrale(_du * _u , _x , _u^2/_uu) -> /* u'u */
    dérivée(_u , _x , _ddu),
    comp(_du, _ddu, _cst), / ,
    produit(2, _cst, _uu).
intégrale(_du / _u , _x , _uu) -> /* u'/u */
    dérivée(_u , _x , _ddu),
    comp(_du, _ddu, _cst), / ,
    quotient(ln(_u), _cst, _uu) .
intégrale(_du * _u^n , _x , (_u)^_m/_cu) -> /* u'*u^n */
    const(_n),
    dérivée(_u , _x , _ddu),
    comp(_du, _ddu, _cst), / ,
    somme(1, _n, _m),
    produit(_m, _cst, _cu) .
intégrale(_du / _u^n , _x , -_cst/_d) -> /* u'/(n-1)*u^n-1 */
    const(_n),
    dérivée(_u , _x , _ddu),
    comp(_du, _ddu, _cst),
    différence(_n , 1 , _m),
    produit(_m , _u^_m , _d), / .
integrale(_du*_exp(_u),_x,exp(_u)/_cst)->
    dérivée(_u, _x, _ddu),
    comp(_du, _ddu, _cst), / .
intégrale(_du / sqrt(_u) , _x , _uu*sqrt(_u)) -> /* u'*sqrt(u) */
    dérivée(_u , _x , _ddu),
    comp(_du, _ddu, _cst), / ,
    quotient(2, _cst, _uu).

comp(_du, -_du, -1)->/.
comp(_du, _du, 1)->/.

```

```
comp(_du, _cst*_du, _cst) -> const(_cst),/.  
comp(_du, _du/_cst, 1/_cst) -> const(_cst),/.
```

intégration de fonction de fonction simples avec changement de variable linéaire

```
intégrale(_f(-_u) , _u , ry(-_u,_iu)) ->
    intégrale(_f(y) , y , _i) , / ,
    moins(_i , _iu) .
```

mais $_i$ est exprimée en fonction de y car on a effectué le changement de variable $y=-u$
il faut donc remplacer y par cette valeur dans $_i$

```
intégrale(_f(-_u) , _u , _iu) ->
    intégrale(_f(y) , y , _i) , / ,
    remplacer(_i , -_u , _ri),
    moins(_ri , _iu) .
intégrale(_f(_cst*_u) , _u , _iu) ->
    const(_cst) ,
    intégrale(_f(y) , y , _i) , / ,
    remplacer(_i , _cst*_u , _ri),
    quotient(_ri , _cst , _iu) .
intégrale(_f(_cst*_u+_c) , _u , _iu) ->
    const(_cst) ,
    const(_c),
    intégrale(_f(y) , y , _i) , / ,
    remplacer(_i , _cst*_u+_c , _ri),
    quotient(_ri , _cst , _iu) .
```

```
intégrale(_f(_u*_cst) , _u , _iu) ->
    const(_cst) ,
    intégrale(_f(y) , y , _i) , / ,
    remplacer(_i , _u*_cst , _ri),
    quotient(_ri , _cst , _iu) .
intégrale(_f(_u/_cst) , _u , _iu) ->
    const(_cst) , / ,
    intégrale(_f(y) , y , _i) , / ,
    remplacer(_i , _u/_cst , _ri),
    produit(_ri , _cst , _iu) .
```

On reconnait aussi les intégrales suivantes, conséquences du tableau des dérivées :

$1/\cos(x)^2$	$\text{tg}(x)$
$1/\text{ch}(x)^2$	$\text{th}(x)$
$1/\text{sh}(x)^2$	$-1/\text{th}(x)$
$1/\cos(x)$	$\ln(\text{tg}(x/2+\pi/4))$
$1/\sin(x)$	$\ln(\text{tg}(x/2))$
$1/\text{tg}(x)$	$\ln(\sin(x))$
$1+\text{tg}(x)^2$	$\text{tg}(x)$

```
intégrale(_cst/cos(x)^2 , _x , _d) ->
    const(_cst),/ ,
    produit(_cst , tg(x)).
intégrale(_cst/sin(x)^2 , _x , _d) ->
    const(_cst),/ ,
    produit(_cst , -1,_p),
```

```

quotient(_p, tg(_x), _d).
intégrale(_cst/ch(_x)^2, _x, _d) ->
const(_cst),/
produit(_cst, th(_x)).
intégrale(_cst/sin(_x)^2, _x, _d) ->
const(_cst),/
produit(_cst, -tg(_x)).
intégrale(_cst/cos(_x), _x, _cst*ln(tg(x/2+pi/4))) ->
const(_cst),/
intégrale(_cst/sin(_x), _x, _cst*ln(tg(x/2))) ->
const(_cst),/
intégrale(_cst/tg(_x), _x, _cst*ln(sin(_x))) ->
const(_cst),/
intégrale(1+tg(_x)^2, _x, tg(_x) ->/

```

on reconnaît aussi les intégrales suivantes :

```

1/1+x^2      arctg(x)
1/1-x^2      arcth(x)
1/(1-x^2)^0.5 arcsin(x)
1/(1+x^2)^0.5 argsh(x)
1/(-1+x^2)^0.5 argch(x)

```

on étend cela à une constante près à $\text{cst}/(\pm a \pm b * x^2)$ et $\text{cst}/(\pm b * x^2 \pm a)$
la méthode consiste à tout diviser par a puis à changer de variable en $x*b/a$

```

xintégrale(_cst/(_a2 + _x2), _x, _v*arctg(_qx)) ->
const(_cst),
carré(_a2, _a),
carréx(_x2, _x, _b), / ,
quotient(_cst, _b, _w),
quotient(_w, _a, _v),
quotient(_b, _a, _p),
produit(_p, _x, _qx).
xintégrale(_cst/(_a2 + _x2), _x, _v*arctg(_q)) ->
intégrale(_cst/(_x2 + _a2), _x, _v*arctg(_q)), /.
xintégrale(_cst/(_a2 - _x2), _x, _v*argth(_qx)) ->
const(_cst),
carré(_a2, _a),
carréx(_x2, _x, _b), / ,
quotient(_cst, _b, _w),
quotient(_w, _a, _v),
quotient(_b, _a, _p),
produit(_p, _x, _qx).
xintégrale(_cst/(-_x2+_a2), _x, _v*argth(_q)) ->
intégrale(_cst/(_a2 - _x2), _x, _v*argth(_q)), /.
xintégrale(_cst/(_a2 - _x2)^0.5, _x, _v*arcsin(_qx)) ->
const(_cst),
carré(_a2, _a),
carréx(_x2, _x, _b), / ,
quotient(_cst, _b, _w),

```

```

    puissance(_a, 3, _a3),
    quotient(_w, _a3, _v),
    quotient(_b, _a, _p),
    produit(_p, _x, _qx).
xintégrale(_cst/(_a2 - _x2)^0.5, _x, _p*arcsin(_q)) ->
    intégrale(_cst/(-_x2 + _a2)^0.5, _x, *arcsin(_q)) ,/.
xintégrale(_cst/(_a2 + _x2)^0.5, _x, _v*argsh(_qx)) ->
    const(_cst),
    carré(_a2, _a),
    carréx(_x2, _x, _b), /,
    quotient(_cst, _b, _w),
    puissance(_a, 3, _a3),
    quotient(_w, _a3, _v),
    quotient(_b, _a, _p),
    produit(_p, _x, _qx).
xintégrale(_cst/(_a2 + _x2)^0.5, _x, _v*argsh(_q)) ->
    intégrale(_cst/(_x2 + _a2)^0.5, _x, _v*argsh(_q)) ,/.
xintégrale(_cst/(-_a2 + _x2)^0.5, _x, _p*argch(_qx)) ->
    const(_cst),
    carré(_a2, _a),
    carréx(_x2, _x, _b), /,
    quotient(_cst, _b, _w),
    puissance(_a, 3, _a3),
    quotient(_w, _a3, _v),
    quotient(_b, _a, _p),
    produit(_p, _x, _qx).
xintégrale(_cst/(-_a2 + _x2)^0.5, _x, _v*argch(_q)) ->
    intégrale(_cst/(x2 - _a2)^0.5, _x, _v*argch(_q)) ,/.

carré(_a^2, _a) -> const(_a), / .
carré(_a, _q) -> number(_a), _a > 0, _q == sqrt(_a), / .
carré(_a, sqrt(_a)) -> ident(_a) /.

carréx(_x^2, _x, 1) -> / .
carréx(_cst * _x^2, _x, _q) ->
    number(_cst), _cst > 0, _q == sqrt(_cst), / .
carréx(_cst * _x^2, _x, sqrt(_cst)) -> ident(_cst).

```

Intégration par partie

Une règle mérite un petit coup d'œil, c'est celle qui représente la méthode de l'intégration par partie, utilisée pour intégrer un produit dont on sait intégrer l'un des opérandes. Cette méthode est basée sur le fait que :

$$\text{dérivée}(u * v) = \text{dérivée}(u) * v + u * \text{dérivée}(v)$$

soit $(u v)' = u' v + v' u$

soit encore, si on intègre : $u v = \text{intégrale}(u' v) + \text{intégrale}(v' u)$

d'où $\text{intégrale}(u' v) = u v - \text{intégrale}(v' u)$

en supposant que l'on cherche à intégrer un produit de la forme $u' * v$, il apparaît que si par bonheur, on connaissait l'intégrale de $v' u$, on serait sauvé, dans la mesure où l'on sait certainement dériver V . l'un des cas d'utilisation classique est u (ou v)= x , puisque u' (ou

$v')=1$ et donc quand on cherche à intégrer $\text{fonc}(x)$ il est intéressant d'intégrer par partie $x*\text{fonc}(x)$.

```

/* intégration par partie d'un produit */
intègre_partie(_du * _v , _x , _i) ->
    intégrale(_du , _x , _u) ,
    dérivée(_v , _x , _dv) ,
    produit(_u , _dv , _udv) ,
    intégrale(_udv , _x , _iudv) ,
    produit(_u , _v , _uv) ,
    différence(_uv , _iudv , _i) .

```

Par exemple : $\text{intégrer}(\cos(x) * x, x, _i)?$
 donne : $_i = x * \sin(x) + \cos(x)$

Changement de variable

Plusieurs des règles ci-dessus ont utilisé implicitement un changement de variable

Par exemple on pose $t = \text{cst} * x$ d'où $dx = dt/\text{cst}$

L'idée d'un changement de variable est de supprimer les termes « compliqués » : (sqrt, exp, ...)

et de se ramener à une fraction de polynômes

et on calcule x et dx (en élevant au carré ou au $\ln()$...)

Si on a des fonctions exponentielles

$t = \ln(x)$ ainsi $x = \exp(t)$ et $dt = 1/t \, dx$

Si on a des fonctions trigonométriques,

si $f(-x) = f(x)$, on pose $t = \cos(x)$ et $dt = -\sin(x) \, dx$

si $f(\pi - x) = f(x)$, on pose $t = \sin(x)$ et $dt = \cos(x) \, dx$

si $f(\pi + x) = f(x)$, on pose $t = \text{tg}(x)$ et $dx = 1/(1+t^2) \, dt$

si deux des trois relations précédentes sont vraies (dans ce cas les trois relations sont vraies), on pose $t = \cos(2x)$

dans les autres cas, on pose $t = \text{tg}(x/2)$ et $dx = 2/(1+t^2) \, dt$

de même, si on a des fonctions hyperboliques, on pose $t = \text{ch}(x)$, $\text{sh}(h)$, $\text{th}(x)$, $t = \text{th}(x/2)$ ou $t = \exp(x)$, et on applique les égalités des fonctions trigonométriques, ou hyperboliques, pour se ramener à l'intégrale d'une fraction rationnelle.

Intégrale des fractions polynomiales

En plusieurs occasions, on a été amené ci-dessus à dupliquer des règles prenant en compte $\text{sqrt}(_x)$ ou $x^{1/2}$, $1/_x$ ou x^{-1} , $_cst * _x$ ou $_x * _cst$, $_x$ et $1 * _x$, ...

Il faudrait mieux faire la transformation d'une expression algébriques en « forme normale » avec des produits « constante » « variable » organisés en puissance décroissante de x , permettant une comparaison efficace et limitant alors les règles nécessaires.

Un polynôme en X est une expression ne contenant que des addition, soustraction, multiplication portant sur la variable X et des constantes.

Un polynôme peut s'écrire de plusieurs façons :

$$3*x^2 + 2*x + 3$$

ou $3 + x^2 + x*x*3$

Pour le représenter de manière unique, le programme met le polynôme en « forme normale », comme une liste ordonnée de monômes de puissances décroissantes de x : $\text{cst} * x^n$

$$[(3,2), (2,1), (3,0)] \text{ pour } 3x^2 + 2x + 3$$

Chaque monôme est représenté par un couple : la constante multiplicative (éventuellement nulle) et l'exposant.

On peut alors ajouter ou multiplier 2 polynômes, même les diviser.

Division de polynomes

Le but est de transformer une fraction de 2 polynômes en une addition d'un polynôme et de fractions plus simples, faciles à intégrer.

Si le degré du numérateur est supérieur à celui du dénominateur, on va extraire toutes les puissances supérieures car, considérant A/B , il existe un unique couple (Quotient, Reste) tel que $A/B = \text{Quotient} + \text{Reste}/B$ et tel que le degré de Reste est strictement plus petit que celui de B .

Ayant ordonné les 2 polynômes en puissance décroissante, la division se pose comme pour des nombres.

Le programme réalise les opérations ci-dessus, resterait à faire les suivantes :

Voir poly.pro et equation.pro

Décomposition du reste en éléments simples de degré 1 ou 2

Le reste se décompose ensuite en une somme d'éléments simples, facilement intégrables

Connaissant les racines du dénominateur (il y a des formules bien connues pour le degré 2 mais il y en a aussi pour le degré 3 : les formules de Cardan) on peut alors exprimer le reste en une somme de fractions de degré 1 (si le dénominateur a des racines complexes, on laisse le polynôme correspondant de degré 2)

$$A/B = csta/(x-a) + cstb/(x-b) + \dots + \frac{ax+b}{x^2-sx+p}$$

En multipliant tout par $(x-a)$ et en posant $x=a$, on trouve facilement $csta$, et etc

Si il y a racine double ,on a 2 éléments simples, de degré 2 et 1:

$$csta/(x-a) + cstb1/(x-b)^2 + cstb2/(x-b) + \dots$$

Là il faut multiplier par $(x-b)^2$ pour avoir $cstb2$

On peut alors multiplier par x et faire tendre x à l'infini, ça va donner une égalité entre les coefficients. Et on en obtient d'autres, autant qu'il en faut, en posant $x=0$, ...

methodemaths.fr donne deux exemples de decomposition :

$$\frac{1}{x^2-1} = \frac{1}{2} \left(\frac{1}{x-1} - \frac{1}{x+1} \right)$$

$$f(x) = \frac{2x^3 + 3x^2 - 44x - 97}{x^2 - 3x - 10} = 2x + 9 + \frac{8}{7} \left(\frac{1}{x-5} \right) + \frac{13}{7} \left(\frac{1}{x+2} \right)$$

Règles de simplification des 6 opérations

```
somme(_x , _y , _d) -> csomme(_x , _y , _d), / .
somme(_x , _y , _x+_y). /* pas de simplification */

csomme(_x , _y , _d) -> ssomme(_x , _y , _d), / .
xcsomme(_x , _y , _d) -> ssomme(_y , _x , _d), / . /* addition commutative, utile si x ou y est
complexe */

/* les règles x... ne sont pas utilisés
xcsomme est inutile, les règles associées ont été dupliquées. */

ssomme(_x , 0 , _x) -> / .
ssomme(0 , _x , _x) -> / .
ssomme(_x , -_y , _z) -> différence(_x , _y , _z), / . /* + par - égal - */
ssomme(-_x , _y , _z) -> différence(_y , _x , _z), / .
ssomme(_x , _y , _z) -> number(_x) , number(_y) , _z ::= _x+_y , / .
ssomme(_x , _y+_z , _d) ->
    csomme(_x , _z , _s) ,
    somme(_s , _y , _d), / .
ssomme(_x , _y+_z , _d) ->
    csomme(_x , _y , _s) ,
    somme(_s , _z , _d), / .
ssomme(_x , _y-_z , _d) ->
    sdifférence(_x , _z , _s) ,
    somme(_s , _y , _d), / .
ssomme(_x , _y-_z , _d) ->
    ssomme(_x , _y , _s) ,
    différence(_s , _z , _d), / .
ssomme(_a*_x , _b*_x , _s) ->
    somme(_a , _b , _m) ,
    produit(_m , _x , _s), / . /* _m peut valoir 1 ou 0, alors ca se simplifie */
ssomme(_x , _a*_x , _s) ->
    somme(_a , 1 , _m) , / ,
    produit(_m , _x , _s).
ssomme(_a*_x , _x , _s) ->
    somme(_a , 1 , _m) , / ,
    produit(_m , _x , _s).
ssomme(_x , _x , _s) -> produit(2 , _x , _s), / .

différence(_x , _y , _d) -> sdifférence(_x , _y , _d), / .
différence(_x , _y , _d) -> sdifférence(_y , _x , _w) , moins(_w , _z), / .
différence(_x , _y , _x-_y). /* pas de simplification */

sdifférence(_x , 0 , _x) -> / .
sdifférence(0 , -_x , _x) -> / .
sdifférence(_x , _x , 0) -> / .
sdifférence(0 , _x , _d) -> moins(_x , _d), / .
sdifférence(_x , -_y , _z) -> somme(_x , _y , _z), / . /* - par - égal + */
sdifférence(-_x , _y , _z) -> somme(_x , _y , _w) , moins(_w , _z), / .
```

```

sdifference(_x, _y, _z) -> number(_x), number(_y), _z := _x - _y, /.
sdifference(_x, _y + _z, _d) ->
    sdifference(_x, _y, _s),          /* x - y se simplifie */
    difference(_s, _z, _d), /.
sdifference(_x, _y + _z, _d) ->
    sdifference(_x, _z, _s),          /* x - z se simplifie */
    difference(_s, _y, _d), /.
sdifference(_x, _y - _z, _d) ->
    sdifference(_x, _y, _s),          /* x - y se simplifie */
    somme(_s, _z, _d), /.
sdifference(_x, _y - _z, _d) ->
    ssomme(_x, _z, _s),              /* x + z se simplifie */
    difference(_s, _y, _d), /.
sdifference(_a*_x, _b*_x, _d) ->
    difference(_a, _b, _s),
    produit(_s, _x, _d), /.
sdifference(_a*_x, _x, _d) ->
    difference(_a, 1, _s),
    produit(_s, _x, _d), /.
sdifference(_x, _a*_x, _d) ->
    difference(1, _a, _s),
    produit(_s, _x, _d), /.

moins(-_x, _x) -> /.
moins(-_cst, _cst) -> const(_cst), /.
moins(_x, _d) -> number(_x), /, _d := -_x.
moins(_x, -_x).

produit(_x, _y, _d) -> cproduit(_x, _y, _d), /.
produit(_x, _y, _d) -> cproduit(_y, _x, _d), /.
produit(_x, _y, _x*_y).              /* pas de simplification */

cproduit(_x, _y, _d) -> sproduit(_x, _y, _d), /.
xcproduit(_x, _y, _d) -> sproduit(_y, _x, _d), /* produit commutatif */

sproduit(0, _x, 0) -> /.
sproduit(_x, 0, 0) -> /.
sproduit(1, _x, _x) -> /.
sproduit(_x, 1, _x) -> /.
sproduit(-_x, -_y, _p) -> produit(_x, _y, _p), /.
sproduit(-_x, _y, _p) -> produit(_x, _y, _q), moins(_q, _p), /.
sproduit(_x, -_y, _p) -> produit(_x, _y, _q), moins(_q, _p), /.
sproduit(_x, _y, _z) -> number(_x), number(_y), _z := _x*_y, /.
sproduit(_x, _x, _x^2) -> /.
sproduit(_x^n, _x^m, _d) ->
    somme(_n, _m, _p),
    puissance(_x, _p, _d), /.
sproduit(_x, _x^n, _d) -> sproduit(_x^1, _x^n, _d), /.
sproduit(_x^n, _x, _d) -> sproduit(_x^1, _x^n, _d), /.

```

```

sproduit(_x, _y*_z, _d) ->
  cproduit(_x, _y, _s), /* x * y se simplifie */
  produit(_s, _z, _d), /.
sproduit(_x, _y*_z, _d) ->
  cproduit(_x, _z, _s), /* x * z se simplifie */
  produit(_s, _y, _d), /.
sproduit(_x, _y/_z, _d) ->
  cproduit(_x, _y, _s), /* x * y se simplifie */
  quotient(_s, _z, _d).
sproduit(_x, _y/_z, _d) ->
  cquotient(_x, _z, _s), /* x / z se simplifie */
  produit(_s, _y, _d).

quotient(_x, _y, _d) -> squotient(_x, _y, _d), /.
quotient(_x, _y, _x/_y). /* pas de simplification */

squotient(0, _x, 0) -> /.      squotient(_x, 1, _x) -> /.
squotient(_x, _x, 1) -> /.
squotient(_x, _y, _z) -> number(_x), number(_y), _z := _x/_y, /.
squotient(-_x, -_y, _p) -> quotient(_x, _y, _q), /.
squotient(-_x, _y, _p) -> quotient(_x, _y, _q), moins(_q, _p), /.
squotient(_x, -_y, _p) -> quotient(_x, _y, _q), moins(_q, _p), /.
squotient(_x^_n, _x^_m, _d) ->
  différence(_n, _m, _p),
  puissance(_x, _p, _d), /.
squotient(_x, _x^_n, _d) -> squotient(_x^1, _x^_n, _d), /.
squotient(_x^_n, _x, _d) -> squotient(_x^_n, _x^1, _d), /.
squotient(_x, _y*_z, _d) -> /* x / y se simplifie */
  cquotient(_x, _y, _s), quotient(_s, _z, _d), /.
squotient(_x, _y*_z, _d) -> /* x / z se simplifie */
  cquotient(_x, _z, _s), quotient(_s, _y, _d).
squotient(_x, _y/_z, _d) -> /* x * z se simplifie */
  cproduit(_x, _z, _s), quotient(_s, _y, _d).
squotient(_x, _y/_z, _d) -> /* x * z se simplifie */
  quotient(_x, _y, _s), produit(_s, _z, _d).

squotient(sin(_x)/cos(_x), tg(_x)) -> /.
squotient(cos(_x)/sin(_x), cotg(_x)) -> /.
squotient(sh(_x)/ch(_x), th(_x)) -> /.
squotient(ch(_x)/sh(_x), 1/th(_x)) -> /.

puissance(_x, 0, 1) -> /.      puissance(_x, 1, _x) -> /.
puissance(1, _x, 1) -> /.      puissance(0, _x, 0) -> /.
puissance(_x, _y, _z) -> number(_x), number(_y), _z := _x^_y, /.

```

```

puissance(_x , _y , _z) ->
  number(_x) ,
  _x < 0 ,
  number(_y) ,
  _y := trunc(_y) ,
  mod(_y , 2) == 0 , / ,
  moins(_x , _xx) ,
  puissance(_xx , _y , _z) .
puissance(_x , _y , _z) ->
  number(_x) ,
  _x < 0 ,
  number(_y) ,
  _y := trunc(_y) ,
  mod(_y , 2) == 1 , / ,
  moins(_x , _xx) ,
  puissance(_xx , _y , _d) ,
  moins(_d , _z) .
puissance(-_x , _y , _z) ->
  number(_y) ,
  _y := trunc(_y) ,
  mod(_y , 2) == 0 , / ,
  puissance(_x , _y , _z) .
puissance(-_x , _y , _z) ->
  number(_y) ,
  _y := trunc(_y) ,
  mod(_y , 2) == 1 , / ,
  puissance(_x , _y , _d) ,
  moins(_d , _z) .
puissance(_cst * _x , _n , _d) ->
  number(_cst) ,
  puissance(_cst , _n , _k) ,
  produit(_k , _x^_n , _d) .
puissance(_x , _y , _z) -> number(_x) , number(_y) , _z := _x^_y , / .
puissance((_x^_y)^_n , _d) -> / ,
  produit(_n , _y , _p) ,                puissance(_x , _p , _d) .
puissance(_x , _y , _x^_y).

```

Règles de simplification des fonctions classiques

```

ln(1 , 0) -> / .
ln(exp(_x) , _x) -> / .
ln(_x , _d) -> number(_x) , / , _d := ln(_x) .
ln(_x*_a , _d) -> / , ln(_x , _sx) , ln(_a , _sa) ,
  somme(_sx , _sa , _d) .
ln(_x^_n , _d) -> / , ln(_x , _s) , produit(_n , _s , _d) .
ln(_x , ln(_x)) -> / .

```

```

log(1 , 0) -> / .
log(_x , _d) -> number(_x) , _d := log(_x) , / .
log(_x , ln(_x)/ln(10)) -> / .

```

```

exp(0 , 1) -> / .
exp(ln(_x) , _x) -> / .
exp(_x , _d) -> number(_x) , / , _d := exp(_x) .
exp(_x , exp(_x)).

```

```

hsh_coded(angle(_, _)).
angle(2*pi+_x , _x) -> / .
angle(2*pi-_x , -_x) -> / .
angle(_x+2*pi , _x) -> / .
angle(_x-2*pi , _x) -> / .
angle(pi+_x , pi+_x) -> / .
angle(pi-_x , pi-_x) -> / .
angle(pi/2+_x , pi/2+_x) -> / .
angle(pi/2-_x , pi/2-_x) -> / .
angle(_x+pi , pi+_x) -> / .
angle(_x-pi , pi+_x) -> / .
angle(-pi+_x , pi+_x) -> / .
angle(-_x+pi , pi-_x) -> / .
angle(_x+pi/2 , pi/2+_x) -> / .
angle(-_x+pi/2 , pi/2-_x) -> / .
angle(_x-pi/2 , -pi/2+_x) -> / .
angle(_x , _y) -> number(_x) ,
    _x > 2*pi , / ,
    _z := _x-2*pi ,
    angle(_z , _y) .
angle(_x , _y) -> number(_x) , / ,
    _x < -2*pi , / ,
    _z := _x+2*pi ,
    angle(_z , _y) .
angle(_x , _x).

```

```

hsh_coded(sin(_, _)).
sin(0 , 0) -> / .
sin(pi, 0) -> /.
sin(pi/2, 1) -> /.
sin(-_x , -sin(_x)) -> / .
sin(_x , _d) -> number(_x) , / , _d := sin(_x).
sin(arcsin(_x) , _x) -> / .
sin(pi-_x , sin(_x)) -> / .
sin(pi+_x , -sin(_x)) -> / .
sin(pi/2-_x , cos(_x)) -> / .
sin(pi/2+_x , cos(_x)) -> / .
sin(-pi/2+_x , -cos(_x)) -> / .
sin(_x , sin(_x)).

```

```

hsh_coded(cos(_, _)).
cos(0 , 1) -> / .
cos(pi, 0) -> /.
cos(pi/2, 1) -> /.

```

```

cos(-_x , cos(_x)) -> /.
cos(_x , _d) -> number(_x) , / , _d := cos(_x) .
cos(arccos(_x) , _x) -> /.
cos(pi-_x , -cos(_x) ) -> /.
cos(pi+_x , -cos(_x) ) -> /.
cos(pi/2-_x , sin(_x) ) -> /.
cos(pi/2+_x , -sin(_x) ) -> /.
cos(-pi/2+_x , sin(_x) ) -> /.
cos(_x, cos(_x)).

```

```

hsh_coded(tg(_ , _)).
tg(0 , 0) -> /.
tg(pi, 0) -> /.
tg(pi/4, 1) -> /.
tg(-_x , -tg(_x)) -> /.
tg(_x , _d) -> number(_x) , / , _d := tg(_x) .
tg(arctg(_x) , _x) -> /.
tg(pi-_x , -tg(_x) ) -> /.
tg(pi+_x , tg(_x) ) -> /.
tg(pi/2-_x , cotg(_x) ) -> /.
tg(pi/2+_x , -cotg(_x) ) -> /.
tg(-pi/2+_x , -cotg(_x) ) -> /.
tg(_x, tg(_x)).

```

```

hsh_coded(cotg(_ , _)).
cotg(pi/2, 0) -> /.
cotg(-_x , -cotg(_x)) -> /.
cotg(_x , _d) -> number(_x) , / , _d := cotg(_x) .
cotg(arccotg(_x) , _x) -> /.
cotg(pi-_x , -cotg(_x) ) -> /.
cotg(pi+_x , cotg(_x) ) -> /.
cotg(pi/2-_x , tg(_x) ) -> /.
cotg(pi/2+_x , -tg(_x) ) -> /.
cotg(-pi/2+_x , -tg(_x) ) -> /.
cotg(_x , 1/tg(_x)) -> /.
cotg(_x, cotg(_x)).

```

```

arcsin(0 , 0) -> /.
arcsin(1 , pi/2) -> /.
arcsin(-_x , _d) -> / , arcsin(_x , _s) , moins(_s , _d).
arcsin(_x , _d) -> number(_x) , / , _d := arcsin(_x) .
arcsin(sinus(_x) , _x) -> /.
arcsin(_x, arcsin(_x)).

```

```

arccos(0 , 1) -> /.
arccos(1, pi/2) -> /.
arccos(-_x , -arccos(_x)) -> /.
arccos(_x , _d) -> number(_x) , / , _d := arccos(_x) .
arccos(cos(_x) , _x) -> /.
arccos(_x, arccos(_x)).

```

```

arctg(0 , 0) -> /.
arctg(-_x , -arctg(_x)) -> /.
arctg(_x , _d) -> number(_x) , / , _d := arctg(_x) .
arctg(tg(_x) , _x) -> /.
arctg(_x,arctg(_x)).

arccotg(pi/2 , 0) -> /.
arccotg(-_x , -arccotg(_x)) -> /.
arccotg(_x , _d) -> number(_x) , / , _d := arccotg(_x) .
arccotg(tg(_x) , _x) -> /.
arccotg(_x, arccotg(_x)).

sh(0, 0) -> /.
sh(-_x, _d) -> moins(_x, _s), sh(_s, _d), /.
sh(_x, sh(_x)) -> /.

ch(0, 1) -> /.
ch(-_x, _d) -> ch(_x, _d)/.
ch(_x, ch(_x)) -> /.

th(0, 0) -> /.
th(-_x, _d) -> moins(_x, _s), th(_s, _d), /.
th(_x, th(_x)) -> /.

```

C'est pour offrir ?, bon, je vous l'emballe :

Manipulation de polynomes

```
/* est-ce un polynome ? */
is_polynome(_exp , _x) ->
    polynome(_exp , _x) ,
    assert(pol(oui) , []) ,
    fail.

is_polynome(_exp , _x) -> /* pour récupérer du stack */
    retract(pol(oui) , []) .

polynome(_x , _x) -> / .
polynome(_a , _x) -> const(_a) , / .
polynome(-_a , _x) -> polynome(_a , _x) .
polynome(_a + _b , _x) -> / ,
    polynome(_a , _x) , polynome(_b , _x) .
polynome(_a - _b , _x) -> / ,
    polynome(_a , _x) , polynome(_b , _x) .
polynome(_a * _b , _x) -> / ,
    polynome(_a , _x) , polynome(_b , _x) .
polynome(_a / _b , _x) -> / ,
    polynome(_a , _x) , polynome(_b) .
polynome(_a ^ _b , _x) ->
    number(_b) ,
    _b > 0 , _b == trunc(_b+0.5) ,
    polynome(_a , _x) .

/* mise du polynome en forme normale */
en_forme_normale(_exp , _x , _fn) ->
    forme_normale(_exp , _x , _fn) ,
    assert(fn(_fn) , []) ,
    fail .

en_forme_normale(_exp , _x , _fn) -> /* pour recuperer du stack */
    retract(fn(_fn) , []) ,
    supprime_head_zero(_fn , _fnn) ,
    add_zero(_fnn , _sfn) ,
    assert(fnn(_sfn) , []) ,
    fail .

en_forme_normale(_exp , _x , _fn) -> /* pour récupérer du stack */
    retract(fnn(_fn) , []) .

forme_normale(_a*_x^n , _x , [f(_a,_n)]) -> const(_a) , const(_n) , / .
forme_normale(_x^n*_a , _x , [f(_a,_n)]) -> const(_a) , const(_n) , / .
forme_normale(_x^n , _x , [f(1,_n)]) -> const(_n) , / .
forme_normale(_x , _x , [f(1,1)]) -> / .
forme_normale(_a , _x , [f(_a,0)]) -> const(_a) , / .

forme_normale(_a + _b , _x , _d) ->
    forme_normale(_a , _x , _fa) ,
```

```

    forme_normale(_b, _x, _fb),
    add(_fa, _fb, _d).
forme_normale(_a*_b, _x, _d) ->
    forme_normale(_a, _x, _fa),
    forme_normale(_b, _x, _fb),
    mul(_fa, _fb, _d).
forme_normale(_a-_b, _x, _d) ->
    forme_normale(_a, _x, _fa),
    forme_normale(_b, _x, _fb),
    _m1 := -1,
    mul_one(f(_m1, 0), _fb, _mfb),
    add(_fa, _mfb, _d).
forme_normale(-_a, _x, _d) ->
    forme_normale(_a, _x, _fa),
    _m1 := -1,
    mul_one(f(_m1, 0), _fa, _d).
forme_normale(_a^_n, _x, _d) ->
    forme_normale(_a, _x, _f),
    puissance_forme(_f, _n, _d).

puissance_forme(_f, 1, _f) -> /.
puissance_forme(_f, _n, _fn) ->
    number(_n), _n > 1,
    _m := _n - 1,
    puissance_forme(_f, _m, _fm),
    mul(_f, _fm, _fn).

/* addition de 2 polynomes en forme normale */
add([], _fa, _fa) -> /.
add(_fa, [], _fa) -> /.
add([f(_a, _e), !_suite], [f(_b, _e), !_reste], [f(_d, _e), !_queue]) -> /,
    simplif(_a+_b, _d),
    add(_suite, _reste, _queue).
add([f(_a, _e), !_suite], [f(_b, _f), !_reste], [f(_b, _f), !_queue]) ->
    _e < _f, /,
    add([f(_a, _e), !_suite], _reste, _queue).
add([f(_a, _e), !_suite], [f(_b, _f), !_reste], [f(_a, _e), !_queue]) ->
    _e > _f, /,
    add(_suite, [f(_b, _f), !_reste], _queue).

/* multiplication de 2 polynomes en forme normale */
mul([], _p, []) -> /.
mul([f(_a, _n), !_fin], _pol, _d) ->
    mul_one(f(_a, _n), _pol, _mp),
    mul(_fin, _pol, _m),
    add(_mp, _m, _d).

/* multiplication d'un polynome par un monome */
mul_one(_x, [], []) -> /.
mul_one(f(_a, _b), [f(_x, _y), !_suite], [f(_m, _n), !_fin]) ->

```

```

simplif(_a*_x , _m) ,
simplif(_b+_y , _n) ,
mul_one(f(_a,_b) , _suite , _fn) .

/* suppression des monomes de degrés supérieurs nuls */
supprime_head_zero([], []) -> / .
supprime_head_zero([f(0 , _x) , !_suite] , _fn) -> / ,
    supprime_head_zero(_suite , _fn) .
supprime_head_zero(_fn , _fn) .

/* completion par les monomes de degrés inférieurs nuls */
add_zero( [ f(_n,_p) , !_suite] , _fn) ->
    add_zero(_p , [f(_n,_p) , !_suite] , _fn) .

add_zero(0 , [] , [f(0,0)]) -> / .
add_zero(0 , [f(_a,0)] , [f(_a,0)]) -> / .
add_zero(_n , [] , [ f(0,_n) , !_fn]) -> / ,
    _m := _n - 1 ,
    add_zero(_m , [] , _fn) .
add_zero(_p , [ f(_n , _p) , !_suite] , [f(_n , _p) , !_s]) -> / ,
    _m := _p - 1 ,
    add_zero(_m , _suite , _s) .
add_zero(_q , [ f(_n , _p) , !_suite] , [f(0 , _q) , !_s]) ->
    _m := _q - 1 ,
    add_zero(_m , [ f(_n , _p) , !_suite] , _s) .

/* substitution de la variable par une expression */
substitue(_x , _x , _newx , _newx) -> / .
substitue(_cst , _x , _newx , _cst) -> const(_cst) , / .
substitue(- _a , _x , _newx , _new) ->
    substitue(_a , _x , _newx , _newa) ,
    moins(_newa , _new) .
substitue(_a + _b , _x , _newx , _new) -> / ,
    substitue(_a , _x , _newx , _newa) ,
    substitue(_b , _x , _newx , _newb) ,
    somme(_newa , _newb , _new) .
substitue(_a - _b , _x , _newx , _new) -> / ,
    substitue(_a , _x , _newx , _newa) ,
    substitue(_b , _x , _newx , _newb) ,
    différence(_newa , _newb , _new) .
substitue(_a * _b , _x , _newx , _new) -> / ,
    substitue(_a , _x , _newx , _newa) ,
    substitue(_b , _x , _newx , _newb) ,
    produit(_newa , _newb , _new) .
substitue(_a / _b , _x , _newx , _new) -> / ,
    substitue(_a , _x , _newx , _newa) ,
    substitue(_b , _x , _newx , _newb) ,
    quotient(_newa , _newb , _new) .
substitue(_a ^ _b , _x , _newx , _new) -> / ,
    substitue(_a , _x , _newx , _newa) ,

```

```

    substitue(_b , _x , _newx , _newb) ,
    puissance( _newa , _newb , _new) .
substitue( f( _a) , _x , _newx , _new ) -> / ,
    substitue( _a , _x , _newx , _newa) ,
    simplif( _f( _newa) , _new) .

```

/* division de deux polynomes en forme normale */

```

divise_polynome( _fn1 , _fn2 , _x , _partiel + _q) ->
    degré( _fn1 , _a , _m) ,
    degré( _fn2 , _b , _n) ,
    _m >= _n , / ,
    _d := _m - _n ,
    simplif( _a / _b , _k) ,
    moins( _k , _mk) ,
    mul_one( f( _mk , _d) , _fn2 , _fn3) ,
    add( _fn1 , _fn3 , [ _debut , !_fn4] ) ,
    supprime_head_zero( _fn4 , _fnn) ,
    divise_polynome( _fnn , _fn2 , _x , _q) ,
    vaut( f( _k , _d) , _x , _partiel) .

```

```

divise_polynome( _fn1 , _fn2 , _x , _a / _pa + _b / _pb) ->
    factorise( _fn2 , _x , _fact) , fail .
divise_polynome( _fn1 , _fn2 , _x , _f1 / _f2) ->
    vaut_fn( _fn1 , _x , _f1) ,
    vaut_fn( _fn2 , _x , _f2) .

```

/* transformation d'une forme normale en forme polynomiale */

```

vaut( f( 0 , _m) , _x , 0) -> / .
vaut( f( _k , 0) , _x , _k) -> / .
vaut( f( 1 , 1) , _x , _x) -> / .
vaut( f( 1 , _m) , _x , _x^_m) -> / .
vaut( f( _k , 1) , _x , _k*_x) -> / .
vaut( f( _k , _m) , _x , _k*_x^_m) .

vaut_fn( [ _d] , _x , _r) -> / , vaut( _d , _x , _r) .
vaut_fn( [ _d , !_reste] , _x , _fin) ->
    vaut( _d , _x , 0) , / ,
    vaut_fn( _reste , _x , _fin) .
vaut_fn( [ _d , !_reste] , _x , _partiel + _fin) ->
    vaut( _d , _x , _partiel) ,
    vaut_fn( _reste , _x , _fin) .

```

```

degré( [ f( _a , _n) , !_fin] , _a , _n) .

```

/* factorisation d'un polynome par recherche de ses racines */

```

factorise( _fn , _x , _fact) ->
    degré( _fn , _k , _b) ,
    assert( facteurs( _k , 0) , [] ) ,
    résoudre_poly( _fn , _x , _r) ,
    que_vaut( _fact , _m) ,

```

```

simplif(_x - _r, _f),
_d := _m + 1,
assert(facteurs(_fact * _f, _d), []),
fail.
factorise(_fn, _x, _fact) ->
degré(_fn, _a, _b),
que_vaut(_fact, _n).

que_vaut(_fact, _degré) ->
retract(facteurs(_fact, _degré), []), /.

```

Résolution d'équations

```

résoudre(_expression) ->
résoudre(_expression, x, _r),
writeln('x = ', _r).

résoudre(_exp=_n, _x, _d) ->
simplif(_exp-_n, _s),
résoudre(_s, _x, _d).

/* l'équation est un produit = 0 */
résoudre(_exp1*_exp2, _x, _d) -> /,
résoudre_fact(_exp1*_exp2, _x, _d).

/* l'équation ne contient x qu'une seule fois */
résoudre(_exp, _x, _d) ->
compte(_exp, _x, 1), /,
résoudre_isole(_exp=0, _x, _d).

/* l'équation est un polynome en x */
résoudre(_exp, _x, _d) ->
is_polynome(_exp, _x), /,
en_forme_normale(_exp, _x, _exp2),
résoudre_poly(_exp2, _x, _d).

/* on va changer de variable */
résoudre(_exp, _x, _d) ->
résoudre_substitution(_exp, _x, _d), / .

/* l'équation est un produit = 0 */
résoudre_fact(_a * _b, _x, _r) -> résoudre(_a, _x, _r).
résoudre_fact(_a * _b, _x, _r) -> résoudre(_b, _x, _r).

/* l'équation ne contient x qu'une seule fois */
résoudre_isole(t=_d, _t, _d) -> / .
résoudre_isole(p(_x)=_z, _t, _d) ->
inverse(p(_), _invp(_)),
simplif(_invp(_z), _q),
résoudre_isole(_x=_q, _t, _d).
résoudre_isole(p(_x, _y)=_z, _t, _d) ->

```

```

    compte(_y, _t, 0), /,
    inverse(1, _p(_z, _y), _r),
    simplif(_r, _m),
    résoudre_isole(_x=_m, _t, _d).
résoudre_isole(_p(_x, _y)=_z, _t, _d) ->
    compte(_x, _t, 0),
    inverse(2, _p(_z, _x), _r),
    simplif(_r, _m),
    résoudre_isole(_y=_m, _t, _d).

    /* l'équation est un polynome en x */
résoudre_poly([f(_a,1), f(_b,0)], _x, _d) -> /,
    simplif(-_b, _e),
    simplif(_e/_a, _d).

résoudre_poly([f(_a,2), f(_b,1), f(_c,0)], _x, _d) ->
    simplifie(_b*_b-4*_a*_c, _delta),
    poly_2(_a, _b, _c, _d, _delta).

résoudre_poly([f(_a,4), f(0,3), f(_b,2), f(0,1), f(_c,0)], _x, _d) ->
    _c /= 0,
    simplifie(_b*_b-4*_a*_c, _delta),
    poly_2(_a, _b, _c, _f, _delta),
    résoudre(_x^2-_f, _x, _d).

xrésoudre_poly([f(_a,3), f(_b,2), f(_c,1), f(_d,0)], _x, _r) ->
    _d /= 0,
    simplifie((3*_a*_c-b^2)/(9*a^2), _p),
    simplifie((2*_b^3/27*_a^2-_b*_c/(3*_a^2)+_d/_a)/2, _q),
    simplifie(_p^3+_q^2, _delta),
    poly_3(_p, _q, _f, _delta),
    résoudre(_x+_b/3*_a=_f).

résoudre_poly(_fn, _x, 0) ->
    x_in_all(_fn).
résoudre_poly(_fn, _x, _r) ->
    x_in_all(_fn), /,
    decx_in_all(_fn, _q),
    résoudre_poly(_q, _x, _r).

x_in_all([f(0, 0)]) -> /.
x_in_all([f(_k, _d), !_suite]) ->
    x_in_all(_suite).

decx_in_all([f(0,0)], []) -> /.
decx_in_all([f(0, _n), !_suite], [f(0, _n), !_q]) -> /,
    decx_in_all(_suite, _q).
decx_in_all([f(_k, _n), !_suite], [f(_k, _m), !_q]) ->
    simplif(_n-1, _m),
    decx_in_all(_suite, _q).

```

```

poly_2(_a, _b, _c, _d, _delta) ->
    number(_delta),
    _delta < 0, /, fail.
poly_2(_a, _b, _c, _d, _delta) ->
    simplifie((-_b+_delta^0.5)/(2*_a), _d).
poly_2(_a, _b, _c, _d, _delta) ->
    simplifie((-_b-_delta^0.5)/(2*_a), _d).

poly_3(_p, _q, _f, _delta) ->
    _delta > 0,
    simplifie((-_q+(_delta)^0.5)^1/3
        + (-_q-(_delta)^0.5)^1/3, _f).
poly_3(_p, _q, _f, _delta) ->
    _delta < 0.

compte(_x, _x, 1) -> /.
compte(_y, _x, 0) -> const(_y), /.
compte(_p(_a, _b), _x, _n) ->
    compte(_a, _x, _n1),

    compte(_b, _x, _n2),
    _n := _n1 + _n2.
compte(_p(_a), _x, _n) ->
    compte(_a, _x, _n).

hsh_coded(inverse(_)).
inverse(-(_x), -(_x)) -> /.
inverse(sin(_x), arcsin(_x)) -> /.
inverse(cos(_x), arccos(_x)) -> /.
inverse(tg(_x), arctg(_x)) -> /.
inverse(cotg(_x), arccotg(_x)) -> /.
inverse(ln(_x), exp(_x)) -> /.

inverse(1, _a+_b, _a-_b) -> /.
inverse(2, _a+_b, _a-_b) -> /.
inverse(1, _a-_b, _a+_b) -> /.
inverse(2, _a-_b, _b-_a) -> /.
inverse(1, _a*_b, _a/_b) -> /.
inverse(2, _a*_b, _a/_b) -> /.
inverse(1, _a/_b, _a*_b) -> /.
inverse(2, _a/_b, _b/_a) -> /.
inverse(1, _a^_b, _a^_n) -> simplif(1/_b, _n).
inverse(1, _a^_b, _dd) ->
    _b := trunc(_b),
    mod(_b, 2) == 0,
    simplif(1/_b, _n),
    simplif(_a^_n, _d),
    moins(_d, _dd), /.

```

```

/* on va changer de variable */
résoudre_substitution(_exp , _x , _d) ->
    substitution(_exp , _x , _newx ) ,
    substitue(_exp , _x , _newx , _newexp) ,
    résoudre(_newexp , _x , _d1) ,
    résoudre(_newx-_d1 , _x , _d).

substitution(_exp , _x , _newx , _newexp) ->
    liste_gros(_exp , _x , _gros) ,
    substituant(_x , _gros , _newx) ,
    substitue(_exp , _x , _newx , _newexp).

```

programme algebre.pro

Ce programme intègre tous ces modules algébriques, et permet donc intégration, dérivation et résolution d'équation.

```

load('m_integre.pro').
load('m_derivee.pro').
load('m_simplif.pro').
load('m_equation.pro').
load('m_poly.pro').

```

```

pour dériver une expression , faire :
    dérive (expression) ?
pour en intégrer une :
    intègre (expression) ?
pour résoudre une équation , faire :
    résoudre(expression) ?

```

par exemple :

```

    intègre (x*cos(x)) ?
    intègre (x*cotg(x^2)) ?
    intègre(sin(x)^2*cos(x)) ?

```

```

ou      intègre ( (sin(x)-cos(x)) / (sin(x)+cos(x))) ?

```

Scène 3

Analyse grammaticale

Le robot de Winograd

Tout programmeur Lisp a commis une version d'un programme gérant un robot capable de manipuler des blocs (la première version, "Shdlru", est due à Terry Winograd). Pour changer, en voici une version Prolog.

Considérons un monde constitué d'une table et de quelques objets (des boîtes, des sphères et des pyramides). Ces objets sont caractérisés par leur forme, leur taille, leur couleur et leur poids. Pour corser l'affaire, il y a un robot muni d'une main articulée et le problème consiste à commander ce robot pour qu'il puisse, sur ordre, manipuler ces fameux objets, c'est à dire en prendre un, le mettre sur un autre ou dans un autre, en étant capable de déterminer ce qu'il faut faire pour que cela soit possible, c'est à dire en déplaçant éventuellement au préalable un ou plusieurs autres objets qui empêcheraient la manœuvre demandée.

définition du microcosme des blocs

Notre monde est régi par les règles suivantes :

- un objet peut être une boîte cubique, une sphère ou une pyramide
- un objet peut être rouge, orange ou vert
- un objet peut être petit, gros ou minuscule
- il peut y avoir plusieurs objets de mêmes caractéristiques, et
 - on peut leur donner un nom pour les distinguer
- on ne peut poser qu'un objet sur une boîte, laquelle doit être
 - fermée et de taille supérieur ou égale à celle de l'objet
- pour prendre un objet, il faut le dégager si nécessaire
- on peut placer un objet dans une boîte, laquelle doit être ouverte
- une grosse boîte peut contenir plusieurs objets minuscules
- une grosse boîte peut contenir un petit objet
- une petite boîte peut contenir un objet minuscule
- la main du robot ne peut contenir qu'un seul objet
- la main du robot doit être vide pour ouvrir ou fermer une boîte
- la table peut supporter N objets

phrases comprises par le robot

Le robot comprend et exécute des ordres du genre :

pose la sphère sur la grosse boîte rouge

ou prends la sphère
pose la sur la grosse boîte rouge

ou pose la sphère dans la boîte
ouvre la boîte
mets la sphère dedans

mets la boîte rouge sur la verte
prends la boîte qui est dans la grosse boîte
ouvre la boîte qui pèse plus de dix kilos
ferme la boîte faisant entre 2 et 3 kilos
ouvre la boîte qui fait au moins 2.6 kilos
prends la boîte dans laquelle il y a une petite sphère
pose la sphère qui est plus lourde que la boîte sur la table
prends la sphère que supporte la boîte de au moins 1 kilo 5
donne la boîte qui contient la pyramide à l'opérateur
donne à l'opérateur la boîte de même poids que la sphère
prends la boîte qui supporte une sphère et qui contient une boîte

On voit donc que l'on peut caractériser les objets de plusieurs façons :

- complètement :

la grosse boîte rouge de 4 kilos

- incomplètement :

la boîte rouge

la rouge

l'objet rouge

- avec une proposition relative précisant une caractéristique de l'objet

une des caractéristiques symboliques :

la boîte dont la couleur est rouge

la boîte de couleur rouge

la boîte qui est de couleur rouge

l'objet cubique qui est rouge

la boîte qui a la couleur rouge

le petit objet rouge de forme cubique

une caractéristique numérique :

la boîte de 2 kilos 500 grammes

la boîte de deux kilos cinq cent grammes

la boîte pesant 2 kilos 5

la boîte faisant 2.005 kilos

la position :

la boîte qui est sur la table

la boîte qui contient la sphère

la sphère que supporte la boîte rouge

la sphère que la boîte rouge supporte

la boîte qui est recouverte par la sphère

la boîte supportant la sphère

la boîte dans laquelle est la sphère
la sphère dans la boîte

- avec une comparaison absolue

la boîte qui pèse au moins 4 kilos 500 grammes
la boîte de poids supérieur à 4 Kg
la boîte dont le poids est de plus de 4 Kg
la boîte qui fait plus de 4 kilos 5
la boîte qui pèse plus de 4 kilos 500
la boîte faisant plus de 4 kilos 500 grammes
la boîte pesant plus de 4 kilos 5 grammes
la boîte pesant au moins quatre cent cinquante kilos
la boîte pesant entre 2 et 3 kilos
la boîte qui pèse de 2 à 5 kilos

- avec une comparaison relative à un autre objet

la boîte plus petite que la rouge
la boîte qui est plus grande que la rouge
la boîte dont le poids est supérieur à celui de la sphère
la boîte de même poids que la pyramide
la boîte qui a même taille que A
l'objet dont le poids est le même que celui de A
l'objet dont le poids est le même que A
l'objet dont le poids est le même que le poids de A
la boîte qui pèse trois fois et demi comme la sphère
la boîte qui pèse trois fois moins que la sphère
la boîte qui pèse plus de trois fois plus que la sphère
la boîte qui pèse plus de la moitié du poids de la sphère
la boîte qui pèse plus du triple de la sphère
la boîte qui pèse au moins le double de la sphère

- avec deux de ces moyens

la boîte de poids égal à 4 kilos qui est verte
la boîte qui pèse 4 kilos et qui est verte
la boîte plus lourde que la verte et dont la taille est petite

- par les pronoms "la", "dedans" ou "dessous" et "celui" ou "celle", qui
font référence à la dernière manipulation effectuée :

prends la sphère
pose la sur la boîte

Pour faciliter, on peut donner des noms aux objets existants :

nomme A la grosse boîte rouge qui est ouverte

On peut rajouter ou supprimer des objets :

génère une petite pyramide orange pesant 1 kg
il faut ici donner toutes les caractéristiques de l'objet
génère B une petite pyramide rouge de 2 kg
désintègre A

Cette suppression peut être temporaire :

donne la petite boîte à l'opérateur
donne à l'opérateur la boîte qui est sur A

Enfin, on peut demander au robot des informations :

quelle est la couleur de l'objet
quel est son poids
 (ou sa taille, sa forme, son poids, sa position, son nom)
où est l'objet
comment est l'objet
que contient l'objet
 ou que supporte, ou que recouvre
qui contient l'objet
liste donne l'état de tous les objets
regarde donne l'état des objets visibles
regarde l'objet donne l'état de l'objet
qui a t il sur l'objet
qui a t il dans l'objet
qui a t il sous l'objet
combien y a t il d'objets ...
combien pèse l'objet ...
combien pèse-t-il
combien de kilos pèse l'objet ...
quel est l'objet ...
y a t il un objet ...
y a t il plus de 2 objets ...
y a t il au moins 3 objets ...
y a t il plus de objet1 que de objet2

Forme de Bakus

Le problème se divise facilement en deux : il faut d'abord comprendre la phrase, et la traduire en une représentation interne, puis il faut exécuter ce qui est demandé. La première partie, l'analyse de la phrase, doit être suffisamment générale pour être réutilisable avec une autre application. Notre propos est de comprendre les phrases sans laisser de côté le moindre mot. C'est à dire que l'analyse ne consiste pas, comme dans les exemples précédents, en une phase d'élimination de mots réputés vides de sens (les articles, les prépositions, ...) puis en une recherche dans une table de mots ou d'expressions clé. Bien au contraire, l'analyse doit être complète, et les accords seront vérifiés ainsi que l'accentuation (5 fautes : zéro !!).

Nous sommes donc amenés à formaliser ce que l'on entend par une phrase :

<phrase> :=
 / <commande de gestion>
 / <ordre à l'impératif>
 / <question>
 / <règle>
 / <affirmation>

(dans ce formalisme, le "/" a le sens de "ou")

<commande de gestion> := liste / efface / help / fin

<ordre à l'impératif> := <verbe><compléments>

<affirmation> := <proposition>
<règle> := si <proposition> et <proposition>) alors <proposition>
<question> :=
 / est ce que <proposition>
 / <proposition> ?
 / <proposition> (avec verbe-t-pronom ou qui,que,quoi,quel)
 / combien y a t il de <groupe nominal> (sans article)
 / combien <verbe de mesure> <groupe nominal>
 / combien de <unité de mesure> <verbe de mesure> <groupe nominal>
 / où <être> <groupe nominal>
 / y a t il <specif nombre><groupe nominal> (sans article)
 / y a t il <comparateur> de <groupe nominal> que de <groupe nominal>
 / y a t il <comparateur> de <groupe nominal>
 / qui a t il <position>
 / quel <être> <article> <nom_de> de <groupe nominal>
 / quel <être> <possessif> <nom_de>
 / qui <verbe> <groupe nominal>
 / que <verbe> <groupe nominal>
 / qui est ce <proposition> (sujet = qui)
 / qu est ce que <verbe><groupe nominal>
 / qu est ce que <groupe nominal><verbe><complément2>

<proposition> :=
 / <groupe nominal><verbe><compléments>
 / <groupe nominal><être><verbe passif><complément> par <groupe nominal>

On ne s'occupera pas ici des compléments circonstanciels facultatifs de temps, de lieu, de cause, de moyen, de manière, de but ... qui, précédés d'une préposition spécifique, peuvent s'intercaler un peu partout. De même on négligera les adverbes et on ne tiendra pas non plus compte du temps des verbes.

<compléments> :=
 / <préposition 1 du verbe> <groupe nominal>
 / <groupe nominal> <complément 2>
<complément 2> := <préposition 2 du verbe><groupe nominal>

<verbe de position> := contenir / supporter / recouvrir
<verbe de mesure> := peser / mesurer / faire
<verbe d'ordre> := fermer / ouvrir / prendre / poser / donner / reprendre
 / générer / désintégrer / nommer / regarder

<préposition du verbe> := à / <préposition de position>
<préposition de position> := dans / sur / sous / avec

<groupe nominal> :=
 / nom_propre
 / <pronom personnel>
 / <pronom>
 / <pronom question>

/ <pronom relatif><relative>
 / <pronom incomplet><préadjectif><nom><postadjectif><relative>
 / <article><spécif adjectif><nom><postadjectif><relative>

<pronom> := le / la
 <pronom de position> := dedans / dessus
 <pronom question>:= qui / quoi / lequel / laquelle
 <pronom incomplet> := quel / quelle
 <pronom relatif> := celui / celle
 <pronom personnel> := je / tu / il

<article> := le / la / un / une
 <possessif>:= sa / son
 <être> := est / sont
 <avoir> := a / ont
 <préadjectif> := petite / minuscule / grosse
 <postadjectif>:=rouge / verte / orange / cubique / sphérique / pyramidale
 lourde / légère / ouverte / fermée
 <comparateur que> := plus / moins / autant / aussi
 <comparateur à> := inférieur / supérieur / égal / identique
 <nom> := boîte / sphère / pyramide / objet / main / table / opérateur
 <nom_de> := couleur / taille / forme / poids / position / nom

<relative> := / <relative élémentaire> et <relative élémentaire>
 / <relative élémentaire> <relative élémentaire>
 / <relative élémentaire>

<relative élémentaire> :=
 / qui <être> de <nom_de - adjectif>
 / qui <être> <attribut>
 / qui <avoir> <article> <nom_de - adjectif>
 / dont <article> <nom_de> <être> <attribut>
 / <prép. position> <pronom question> <être> <groupe nominal>
 / de <spécif de quantité>
 / <participe présent de mesure> <spécif de quantité>
 / <participe présent de position> <groupe nominal>
 / qui <verbe de mesure> <spécif de quantité>
 / qui <verbe de position> <groupe nominal>
 / que <verbe de position> <groupe nominal>
 / que <groupe nominal> <verbe de position>
 / qui <être> <participe de position> par <groupe nominal>
 / de <nom_de - adjectif>
 / <comparateur que> <adjectif> que <groupe nominal>
 / <position> sauf si le verbe se construit avec
 une préposition de position

<nom_de - adjectif> := / <préadjectif> <nom_de>
 / <nom_de> <postadjectif>
 / <nom_de> <spécif adjectif>

<attribut> := / <adjectif>
 / <position>
 / <spécif adjectif>

/ de <specif de quantité>

<spécif adjectif> :=

/ <comparateur que> <adjectif> que <groupe nominal>

/ <comparateur que> <adjectif> que <celle> de <groupe nominal>

/ <comparateur à> à <celle> de <groupe nominal>

/ <article> même que <celle> de <groupe nominal>

/ <article> même que <groupe nominal>

<position> := / <préposition de position> <groupe nominal>

/ <pronom de position>

<specification de quantité> :=

/ <quantité comparée>

/ <quantité multipliée>

/ <quantité entre bornes>

<quantité comparée> :=

/ <comparateur que> de <quantité multipliée>

/ au <comparateur que> <quantité multipliée>

<quantité multipliée> :=

/ <nombre> fois <quantité élémentaire>

/ <article> <fraction> de <quantité élémentaire>

/ <quantité élémentaire>

<quantité entre bornes> :=

/ entre <nombre> et <quantité> / entre <quantité> et <quantité>

/ de <nombre> à <quantité> / de <quantité> à <quantité>

<quantité élémentaire> :=

/ <quantité>

/ celui de <groupe nominal>

/ <article> <nomde> <groupe nominal>

<quantité> :=

/ <nombre> <unité> <quantité>

/ <nombre> <unité>

/ <nombre> <unité> <nombre>

cela permet de distinguer : 3 kilos 5 grammes

de 3 kilos 500 grammes , 3 kilos 500 , et 3 kilos 5

Le problème consiste à trouver l'unité du dernier nombre lorsqu'elle

n'est pas spécifiée. L'unité sous-entendue est celle de rang

immédiatement inférieure si le nombre est supérieur à 9, et c'est

une fraction par 10 de l'unité spécifiée avec le nombre précédent si

le nombre est inférieur à 10.

<unité> := kilo / gramme / tonne / litre / mètre / kilomètre

<specif de nombre> :=

/ <nombre>

/ <comparateur que> de <nombre>

/ au <comparateur que> <nombre>

<nombre> :=

/ <nombre de millier> <nombre de centaine> <nombre de dizaine-unité>

/ <nombre> <nombre de groupe>

<nombre de millier> := <valeur> mille trois mille

<nombre de centaine> := <valeur> cent trois cent, quinze cent

<nombre de dizaine et unité> :=

 <valeur> et un

 trente et un

 <valeur> <valeur>

 trente deux, dix huit

 <valeur>

 huit, seize, vingt

<nombre de groupe> := dizaine / douzaine / vingtaine / centaine

<fraction> := moitié / tiers / quart / double / triple

<valeur> :=

 un, deux, trois, dix, vingt, quatrevingt ... , onze , douze , treize

 / <un nombre entier ou réel sous forme de chiffres>

La transformation de ce formalisme en prolog se fait simplement :

 <proposition> := <sujet> <verbe> <compléments>

 / <groupe><verbe passif><complément>par<groupe>

devenant globalement :

 proposition -> sujet, verbe, compléments , / .

 proposition -> groupe , verbe_passif , complément , par , groupe .

ou proposition ->

 sujet ,

 verbe ,

 compléments , / .

ce qui revient au même.

On ajoute 3 paramètres par entité : le premier est la liste de mots supposée commencée par l' entité, le second est la liste amputée de cette entité et le troisième est la structure sous laquelle on désire représenter cette entité pour la suite des opérations.

 proposition(_phrase , _reste , [_sujet , _verbe , _complément]) ->

 sujet(_phrase , _x , _sujet) ,

 verbe(_x , _y , _verbe) ,

 complément(_y , _reste , _complément).

Où il apparait on ne peut plus clairement, n'est-ce pas ? eh c'est pas l'heure de la sieste, que la proposition amputée de son sujet, c'est _x, qui doit commencer par un verbe et _x amputé de ce verbe, c'est _y, lequel doit commencer par un complément, et _y amputé de ce complément, c'est _reste. On remarque aussi que le résultat rendu par le sous programme "proposition" est la concaténation des résultats rendus par les sous programmes *sujet*, *verbe* et *complément*.

les constructions des verbes

Certains verbes nécessitent un ou deux compléments, introduits ou non par des prépositions. il est donc nécessaire de déclarer la construction des verbes :

```
construction (manger , complément) .  
construction (donner , complément , à , complément).
```

et cela est pris en compte par :

```
proposition(_phrase,_reste,[_sujet,_verbe,_compl1,_prép,_compl2])->  
    sujet(_phrase , _x , _sujet) ,  
    verbe(_x , _y , _verbe) ,  
    compléments(_y , _reste , _verbe , _compl1 , _prép , _compl2).
```

```
compléments(_p , _r , _nomverbe , _complément1 , ' ' , ' ') ->  
    construction(_nomverbe , complément),  
    complément(_p , _r , ' ' , _complément1) , / .  
compléments(_p , _r , _nomverbe , _compl1 , _prép , _compl2) ->  
    construction(_nomverbe , complément , _prép , complément),  
    complément(_p , _y , ' ' , _compl1) .  
    complément(_y , _r , _prép , _compl2) , / .
```

"donner quelquechose à quelqu'un" et "donner à quelqu'un quelquechose"

c'est du pareil au même, on rajoute donc :

```
compléments(_p , _r , _nomverbe , _compl1 , _prép , _compl2) ->  
    construction(_nomverbe , complément , _prép , complément),  
    complément(_p , _y , _prép , _compl2) ,  
    complément(_y , _r , ' ' , _compl1) .
```

les terminaux

Les terminaux sont les mots de la langue française : les articles, les noms, les adjectifs, les verbes ... Considérons par exemple les articles, qui apparaissent dans une règle du genre :

```
<sujet> := <article> <nom>  
<article> := le / la / ...
```

La règle "article" se note selon le même principe que "proposition" :

```
article(_x,_reste,_résultat) -> quoi ?
```

et oui, -> quoi ? et bien -> rien ! puisque l'article ne se décompose pas.

soit : `article(_x , _reste , _résultat) .`

et là il y a une kolossale astuce pour faire comprendre à l'esclave qui remue les octets là dedans que `_x` commence par "le" et finit par `_reste` :

```
article([le , !_reste] , _reste , _résultat).
```

Arrivé là, vous avez gagné une petite bière. On souffle un peu, et puis on fait aussi quelques exercices d'assouplissement du neurone : attention, on y va : et 1, 2, 3 , 1, 2, 3 . recommencez trois fois.

Nous disions donc que :

```
article([le,!_reste] , _reste , _résultat).
```

Ne pas oublier que cette règle est appelée depuis la partie droite d'une autre règle par quelquechose comme :

```
xxx( ... ) -> ... , article(_phrase , _suite_ , _résultat_article) ,  
...
```

Il faut enfin préciser ce que l'on veut comme résultat :

```
article([le,!_reste] , _reste , [défini, masculin, singulier]).
```

de même pour les autres articles, en mettant des "/" pour accélérer en évitant de parcourir les règles « article » suivantes, qui ne donneront rien :

```
article([la,!_reste] , _reste , [défini, féminin, singulier]) -> / .
```

En effet lorsqu'un article a été reconnu, il est inutile de revenir en arrière pour chercher une solution avec un autre article à cette même position.

Pour les synonymes des noms, des adjectifs et des verbes il suffit de spécifier des résultats identiques, au genre et au nombre près :

```
verbe(supporte , [supporter , présent , singulier]) .  
verbe(soutient , [supporter , présent , singulier]) .
```

Les restrictions syntaxiques

Il reste à résoudre un certain nombre de problèmes. Par exemple, le verbe doit s'accorder en nombre avec le sujet. pour cela, on va supposer que les résultats rendus à chaque niveau contiennent le nombre des éléments correspondants. Je traduis : le sous programme sujet va rendre une structure contenant entre autres le nombre du dit sujet, et le sous programme verbe va rendre le nombre du dit verbe et alors, "y'a qu'a" comparer ces deux valeurs. Notre règle "proposition" devient donc :

```
proposition(_prase,_reste,[_sujet,_verbe,_compl1,_prép,_compl2]) ->  
sujet(_phrase,_x,_sujet) ,  
_sujet = [_nombresujet , _dsujet] , /* on extrait le nombre */  
verbe(_x,_y,_verbe) ,  
_verbe = [_nombreverbe , _verbe] , /* on extrait le nombre */  
_nombresujet = _nombreverbe , /* on les compare */  
compléments(_y,_reste,_verbe,_compl1,_prép,_compl2).
```

on va même se passer de la comparaison explicite :

```
proposition(_prase,_reste,[_sujet,_verbe,_compl1,_prép,_compl2]) ->  
sujet(_phrase,_x,_sujet) ,  
_sujet = [_nombre , _dsujet] , /* on extrait le nombre */  
verbe(_x,_y,_verbe) ,  
_verbe = [_nombre , _dverbe] , /* on extrait le nombre */  
/* la comparaison est alors implicite par l'unification de _nombre*/  
compléments(_y,_reste,_verbe,_compl1,_prép,_compl2).
```

On va utiliser à fond le mécanisme d'unification des paramètres :

```

proposition(_phrase, _reste, [_sujet, _verbe, _compl1, _prép, _compl2]) ->
    sujet(_phrase, _x, [_nombre, _dsujet]),
    verbe(_x, _y, [_nombre, _dverbe]),
    /* l'extraction et la comparaison sont alors implicites */
    compléments(_y, _reste, _verbe, _compl1, _prép, _compl2).

```

Application à la grammaire du français

Demandez la grammaire, elle est belle elle est fraîche ma grammaire !

robot ->

```

module(robot),
initialisation,
repeat,                /* boucle principale */
    traduit_phrase.

```

traduit_phrase ->

```

readwords(_phrase),      /* lire une phrase */
type := ordre,
traduisible := oui,
pronom_quest := non,
constr_position := non,
analyse(_phrase, _résultat), /* analyser la phrase */
_type := type,
_type \= commande,
executer(_résultat, _type). /*executer si traduisible*/
/* type = affirmation/question/ordre/commande/règle */

/* analyse de la phrase */

```

```

analyse(_phrase, " ) -> /* c'est une commande de gestion */
    commande(_phrase), /,
    type := commande.

```

```

analyse(_phrase, _résultat) -> /* c'est un ordre à l'impératif */
    ordre(_phrase, _résultat), /,
    type := ordre.

```

```

analyse(_phrase, _résultat) -> /* c'est une question */
    question(_phrase, _reste, _résultat),
    ana_reste(_reste),
    type := question, / .

```

```

analyse(_phrase, _résultat) -> /* c'est une affirmation */
    type := affirmation,
    proposition(_phrase, _reste, _résultat),
    ana_reste(_reste), / .

```

```

analyse([si, !_x], [_conclusion, _conditions]) ->
    type := règle, /* c'est une règle */
    proposition(_x, _y, _cond),
    etsi(_y, [alors, !_z], [_cond], _conditions),
    proposition(_z, [], _conclusion), / .

```

```

analyse(_phrase, _résultat) ->

```

```

type('robot.hlp') , fail .

ana_reste([]) -> / .
ana_reste(['?']) -> type := question .

```

/* analyse et execution des commandes de gestion */

```

hsh_coded(commande(_)) .
commande( _x ) -> ioresult <> 0 , main , window(1,1) , exit.
commande( [fin] ) -> main , window(1,1) , exit.
commande( [list] ) -> list_module , / .
commande( [list , _nom] ) -> list(_nom) , / .
commande( [efface] ) -> clear_module , initialisation , / .
commande( [help] ) -> window(1,1) , hlp('robot.hlp') , dessine_tout , / .

```

/* analyse des ordres à l'impératif */

```

ordre([_verbe , _nom , !_compléments] , _ordre) ->
    avec_surnom(_verbe , _nomv) ,
    groupe_nominal(_compléments , [] , _compl2) ,
    formule_ordre(_ordre , _nomv , _nom , ' , _compl2) , / .
avec_surnom(nomme , nommer) .
avec_surnom(appelle , nommer) .
avec_surnom(génère , générer) .
ordre([_verbe , !_compléments] , _ordre) ->
    impératif(_verbe , _nomv) , / ,
    compléments(_compléments , [] , _nomv , _compl1 , _prép , _compl2) ,
    formule_ordre(_ordre , _nomv , _compl1 , _prép , _compl2).

formule_ordre(execute_ordre(_nomv) ,
    _nomv , ' , ' , ' ) -> / .
formule_ordre(execute_ordre(_nomv , _compl1) ,
    _nomv , _compl1 , ' , ' ) -> / .
formule_ordre(execute_ordre(_nomv , _compl1 , _prép , _compl2) ,
    _nomv , _compl1 , _prép , _compl2).

```

/* analyse des questions */

```

hsh_coded(question(_ , _ , _)) .
question([est,ce,que , !_proposition] , _reste , estceque(_résultat)) -> / ,
    proposition(_proposition , _reste , _résultat) .
question([combien,y,a,t,il,de , !_groupe] , _reste , combien(_résultat))->/ ,
    groupe_nominals(_groupe , _reste , _résultat) .
question([combien , _verbe , !_groupe] , _reste , quel(_entité , _résultat))->
    verbe(_verbe , [_v , _nombre] ,
    verbe_mesure(_v , _entité) ,
    groupe_nominal(_groupe , _reste , _résultat) , / .
question([combien , _verbe , !_groupe] , _reste , quel(_entité , _résultat))->
    verbe(_verbe , [_v , _nombre] , / ,
    verbe_mesure(_v , _entité) ,

```

```

t(_groupe , [_pronom]) ,
pro_personnel(_pronom , [_genre , _nombre]) ,
groupe_nominal([le] , [] , _résultat) .
question([combien, de, _unité, _verbe, !_groupe], _reste,
        quel(_entité , _rapport , _résultat))->
    unité(_entité , _unité , _rapport) ,
    verbe(_verbe , [_v , _nombre]) , / ,
    verbe_mesure(_v , _entité) ,
    groupe_nominal(_groupe , _reste , _résultat) .
question([où , _est , !_groupe] , _reste , où(_résultat)) -> / ,
    être(_est , _nombre) ,
    groupe_nominal(_groupe , _reste , _résultat) .
question([y,a,t,il, !_specif_nombre], _reste, existe(_nombre,_résultat))->
    specif_nombre(_specif_nombre , _gn , _nombre) ,
    groupe_nominals(_gn , _reste , _résultat) , / .
question([y,a,t,il,_comp,de,!_specif], _reste, existe(_vcomp,_résu1,_résu2))->
    comparateur(_comp , _vcomp) ,
    groupe_nominals(_specif, [que, de , !_nom] , _résu1) ,
    groupe_nominals(_nom, _reste , _résu2) , / .
question([y,a,t,il,_comp,de,!_nom], _reste, existe(_vcomp, _résultat))->
    comparateur(_comp , _vcomp) ,
    groupe_nominals(_nom, _reste , _résultat) , / .
question([qui,a,t,il,_prép, !_groupe], _reste, quoi(_prép,_résultat)) ->/,
    prép_position(_prép) ,
    groupe_nominal(_groupe , _reste , _résultat) .
question([que , _verbe , !_groupe], _reste, quoi(_prép , _résultat)) ->
    verbe (_verbe , [_v , _nombre]) ,
    verbe_position(_v , _prép , _prépinv) , / ,
    groupe_nominal(_groupe , _reste , _résultat) .
question([qui , _verbe , !_groupe], _reste, quoi(_prépinv , _résultat)) ->
    verbe (_verbe , [_v , _nombre]) ,
    verbe_position(_v , _prép , _prépinv) , / ,
    groupe_nominal(_groupe , _reste , _résultat) .
question([qui , _est , _participe , _par , !_groupe] , _reste ,
        quoi(_prép , _résultat)) ->
    être(_est , _nombre) ,
    participe (_participe , [_verbe , _genre , _nombre]) ,
    verbe_position(_verbe , _prép , _prépinv) , / ,
    groupe_nominal(_groupe , _reste , _résultat) .
question([_quel , _est , _art , _nomde , de , !_grouperelative] , _reste ,
        quel(_vnomde , _résultat)) ->
    questionif(_quel , [_v , _genre , _nombre]) ,
    être(_est , _nombre) ,
    détermine(_art , [_déf , _genre , _nombre] ) ,
    nomde(_nomde , [_vnomde , _genre , _nombre]) ,
    groupe_nominal(_grouperelative , _reste , _résultat) , / .
question([_quel , _est , _art , _nomde , _de , !_grouperelative] , _reste ,
        quel(_vnomde , _résultat)) ->
    questionif(_quel , [_v , _genre , _nombre]) ,
    être(_est , _nombre) ,

```

```

de(_de , _nombr) ,
détermine(_art , [_déf , _genre , _nombre] ),
nomde(_nomde , [_vnomde , _genre , _nombre]) ,
groupe_nominals(_grouperelative , _reste , _résultat) , / .
question([_quel , _est , _poss , _nomde] , [] ,
          quel(_vnomde , _résultat)) ->
questionif(_quel , [_v , _genre , _nombre] ) ,
être(_est , _nombre) ,
détermine(_poss , [possessif , _genre , _nombre] ),
nomde(_nomde , [_vnomde , _genre , _nombre]) ,
groupe_nominal([le] , [] , _résultat) , / .
question([_quel , _est , !_grouperelative] , _reste , quel(_résultat)) ->
questionif(_quel , [_v , _genre , _nombre] ) ,
être(_est , _nombre) ,
groupe_nominal(_grouperelative , _reste , _résultat) , / .
question([qui,est,ce, !_proposition] , _reste , _résultat) -> / ,
proposition(_proposition , _reste , _résultat ) .
question([qu,est,ce,que, !_proposition] , _reste , _résultat) ->
proposition(_proposition , _reste , _résultat ) .
question([qu,est,ce, !_proposition] , _reste , _résultat) ->
proposition(_proposition , _reste , _résultat ) .
question([_question, !_proposition] , _reste ,
          questionnant(_question , _résultat)) ->
questionnant(_question) ,
proposition(_proposition , _reste , _résultat ) .

```

/* **proposition** */

```

proposition(_p , _reste , _formule ) ->
groupe_nominal(_p , _x , _sujet) ,
_sujet = [_adjsuj , _adjp , _ajoud , _nomsuj , _defsuj , _genre , _nombre] ,
groupe_verbe(_x , _y , _verbe) ,
_verbe = [_nomvb , _nombre , _aff , _genre , _nombre] ,
compléments(_y , _reste , _nomvb , _complément1 , _prép , _complément2) ,
_résultat = [_sujet , _verbe , _complément1 , _prép , _complément2] ,
formule_prop(_résultat , _formule) , / .
proposition(_p , _reste , _formule ) ->
groupe_nominal(_p , _x , _complément1) ,
_complément = [_adj , _adjp , _ajd , _nomsuj , _defsuj , _genre , _nombre] ,
verbe_passif(_x , _y , _verbe) ,
_verbe = [_nomvb , _nombre , _aff , _genre , _genre] ,
groupe_nominal(_y , _z , _sujet) ,
complemen2(_z , _reste , _nomvb , _prép , _complément2) ,
_résultat=[_sujet , _verbe , _complément1 , _prép , _complément2] ,
formule_prop(_résultat , _formule) , / .

```

/* **si ... et si ...** */

```

etsi([et,si, !_x] , _r , _formulein , _formuleout) ->

```

```

    proposition(_x , _y , _formule ) ,
    conc(_formulein , [_formule] , _fl) ,
    etsi(_y , _r , _fl , _formuleout) , / .
etsi(_p , _p , _s , _s) .

```

```

ousi([ou,si,!_x] , _r , _ousiin , _ousiout) ->
    proposition(_x , _y , _ousi ) ,
    conc(_ousiin , [_ousi] , _ous) ,
    ousi(_y , _r , _ous , _ousiout) , / .
ousi(_p , _p , _s , _s).

```

```

etalors(_p , _r , _etalorsin , _etalorsout) ->
    et(_p , _x) ,
    proposition(_x , _y , _etalors ) ,
    conc(_etalorsin , [_etalors] , _etal) ,
    etalors(_y , _r , _etal , _etalorsout) , / .
etalors(_p , _p , _s , _s).

```

```

sinon([], [] , ' ') -> / .
sinon([sinon, _x] , _r , _sinon) ->
    proposition(_x , _y , _sinon ) .
etsinon([et,sinon,!_x] , _r , _etsinonin , _etsinonout) ->
    proposition(_x , _y , _etsinon ) ,
    conc(_etsinonin , [_etsinon] , _ets) ,
    etsinon(_y , _r , _ets , _etsinonout) , / .
etsinon(_p , _p , _s , _s).

```

/* **groupe nominal** */

```

groupe_nominal([_p,!_x],_r,[_adj,_adjp,_rel,_nomnom,
                                _defar,_genre,_nombre])->
    détermine(_p , [_defar , _genre , _nombre] ) ,
    préadjectif(_x , _y , [_adj , _genre , _nombre] ) ,
    nom(_y , _z , [_nomnom , _genre , _nombre] ) ,
    postadjectif(_z , _t , [_adjp , _genre , _nombre] ) ,
    relative(_t , _r , _rel , _nombre) , / .
groupe_nominal([_p , !_x] , _r ,
    [_adj,'',_rel,_nomnom,_defar,_genre,_nombre])->
    questionif(_p , [_adj , _genre , _nombre] ) ,
    type := question ,
    nom(_x , _y , [_nomnom , _genre , _nombre] ) ,
    postadjectif(_y , _z , [_adjp , _genre , _nombre] ) ,
    relative(_z , _r , _rel , _nombre) , / .
groupe_nominal([_p,!_r],_r,['','',rel('','')],
    _pronom,question,_genre,_nombre)) ->
    pronomquest (_p , _pronom) ,
    pronom_quest := oui ,
    type := question , / .

```

```

groupe_nominal([_p , !_r] , _r , ['','rel('',''),
                                pronom,'',_genre,_nombre]) ->
    pro_personnel(_p , [_genre , _nombre]) , / .
groupe_nominal([_nom, !_x], _r , ['','rel,_nomnom','',_genre,singulier])->
    nompropre(_nom , [_nomnom , _genre]) ,
    relative(_x , _r , _rel , singulier) , / .
groupe_nominal([_p,!_x],_r , ['','','rel, pronom,_defar,_genre,_nombre])->
    pronom_démo(_p , [_pronom , _genre , _nombre]) ,
    relative(_x , _r , _rel , _nombre) , / .

```

```

groupe_nominals(_p , _r , _résu) ->      /* groupe nominal sans article */
    groupe_nominal([de , !_p] , _r , _résu) .

```

/* proposition relative */

```

relative(_p , _r , rel(_rel1 , _rel2 ) , _nombre) ->
    relative_e(_p , _x , _rel1 , _nombre) ,
    relative2(_x , _r , _rel2 , _nombre) , / .
relative(_r , _r , rel('','') , _nombre) .
relative2([et , !_p] , _r , _rel2 , _nombre) ->
    relative_e(_p , _r , _rel2 , _nombre) , / .
relative2(_p , _r , _rel2 , _nombre) ->
    relative_e(_p , _r , _rel2 , _nombre) , / .
relative2(_r , _r , '' , _nombre) .

hsh_coded(relative_e(.,.,.)) .
relative_e([de, _nomde, _adj, !_r], _r, de(_vnom , _vadj), _nombre) ->
    nomde(_nomde , [_vnom , _genre , _nombr]) ,
    postadjectif(_adj , [_vadj , _genre , _nombr]) , / .
relative_e([de, _adj, _nomde, !_r], _r, de(_vnom , _vadj), _nombre) ->
    préadjectif(_adj , [_vadj , _genre , _nombr]) , / ,
    nomde(_nomde , [_vnom , _genre , _nombr]) , / .
relative_e([de, _nomde, !_quant ], _r , _résu , _nombre) ->
    nomde(_nomde , [_entité , _genre , _nombr]) , / ,
    mesure(_entité , _quant , _r , _résu) , / .
relative_e([de, !_x], _r, de(_résu), _nombre) ->
    groupe_nominal(_x , _r , _résu) .
relative_e([de, !_quant ], _r , _résu , _nombre)->
    mesure(_entité , _quant , _r , _résu) , / .

relative_e([qui, _est, _verbe, par, !_nom], _r, position(_prép, _résu), _nombre)->
    être(_est , _nombre) ,
    participe(_verbe , [_v , _genre , _nombre]) ,
    verbe_position(_v , _prép , _prépinv) , / ,
    groupe_nominal(_nom , _r , _résu) .
relative_e([qui, _est, _adj, !_r], _r, quiest(_vadj), _nombre) ->
    être(_est , _nombre) ,
    adjectif(_adj , [_vadj , _genre , _nombre]) , / .
relative_e([qui, _est, de, _nomde, _adj, !_r], _r, de(_vnom , _vadj), _nombre) ->

```

```

        être(_est , _nombre) ,
        nomde(_nomde , [_vnom , _genre , _nombr]) ,
        postadjectif(_adj , [_vadj , _genre , _nombr]) , / .
relative_e([qui,_est,de,_adj,_nomde,!_r],_r,de(_vnom , _vadj),_nombre) ->
        être(_est , _nombre) ,
        préadjectif(_adj , [_vadj , _genre , _nombr]) , / ,
        nomde(_nomde , [_vnom , _genre , _nombr]) , / .
relative_e([qui , _est , de , !_quant],_r , _résu , _nombre) ->
        être(_est , _nombre) ,
        mesure(_entité , _quant , _r , _résu) , / .
relative_e([qui , _est , de , _nomde , !_quant],_r , _résu , _nombre) ->
        être(_est , _nombre) ,
        nomde(_nomde , [_vnom , _genre , _nombr]) , / ,
        mesure(_vnom , _quant , _r , _résu) , / .
relative_e([qui , _est , _comp , _adj , que , !_nom],_r ,
        de_nombre(_vnom , _vcomp , _résu),_nombre) ->
        être(_est , _nombre) ,
        comparateur(_comp , _zcomp) ,
        adjectif(_adj , [_vadj , _genre , _nombre]) ,
        correspond(_vadj , _vnom) ,
        associe(_zcomp , _vadj , _vcomp) ,
        groupe_nominal(_nom , _r , _résu) , / .
relative_e([qui,_est,_prép,!_x],_r,position(_prép , _résu),_nombre) ->
        être(_est , _nombre) ,
        prép_position(_prép) , / ,
        groupe_nominal(_x , _r , _résu) .
relative_e([qui,_a,_art,_adj,_nomde,!_r],_r,de(_vnom,_vadj),_nombre)->
        avoir(_a , _nombre) ,
        détermine(_art , [_typ , _genre , _nombr]) ,
        préadjectif(_adj , [_vadj , _genre , _nombr]) ,
        nomde(_nomde , [_vnom , _genre , _nombr]) , / .
relative_e([qui,_a,_art,_nomde,_adj,!_r],_r,de(_vnom , _vadj) , _nombre)->
        avoir(_a , _nombre) ,
        détermine(_art , [_typ , _genre , _nombr]) ,
        nomde(_nomde , [_vnom , _genre , _nombr]) ,
        postadjectif(_adj , [_vadj , _genre , _nombr]) , / .
relative_e([qui,_a,_art,_nomde,!_quant ],_r , _résu , _nombre) ->
        avoir(_a , _nombre) ,
        détermine(_art , [_typ , _genre , _nombr]) ,
        nomde(_nomde , [_entité , _genre , _nombr]) , / ,
        mesure(_entité , _quant , _r , _résu) , / .
relative_e([qui , _a , !_quant] , _r , _résu , _nombre) ->
        avoir(_a , _nombre) ,
        mesure(_entité , _quant , _r , _résu) , / .
relative_e([qui,_verbe,!_nom],_r,position(_prépinv , _résu) , _nombre)->
        verbe(_verbe , [_v , _nombre]) ,
        verbe_position(_v , _prép , _prépinv) , / ,
        groupe_nominal(_nom , _r , _résu) .
relative_e([qui , _verbe , !_quant],_r , _résu , _nombre)->
        verbe(_verbe , [_v , _nombre]) , / ,

```

```

    verbe_mesure(_v , _entité) ,
    mesure(_entité , _quant , _r , _résu) , / .
relative_e([dont, _art, _nomde, est, _adj, !_r], _r, de(_vnom, _vadj), _nombre) ->
    détermine(_art , [_typ , _genre , _nombr]) ,
    nomde(_nomde , [_vnom , _genre , _nombr]) ,
    postadjectif(_adj , [_vadj , _genre , _nombr]) , / .
relative_e([dont, _art, _nomde, est, _adj, !_r], _r, de(_vnom, _vadj), _nombre) ->
    détermine(_art , [_typ , _genre , _nombr]) ,
    nomde(_nomde , [_vnom , _genre , _nombr]) ,
    préadjectif(_adj , [_vadj , _genre , _nombr]) , / .
relative_e([dont, _art, _nomde, est, de, !_quant] , _r , _quantité , _nomb) ->
    détermine(_art , [_typ , _genre , _nombr]) ,
    nomde(_nomde , [_vnom , _genre , _nombr]) ,
    mesure(_vnom , _quant , _r , _quantité) , / .
relative_e([dont, _art, _nomde, est, !_quant] , _r , _quantité , _nombre) ->
    détermine(_art , [_typ , _genre , _nombr]) ,
    nomde(_nomde , [_vnom , _genre , _nombr]) ,
    mesure(_vnom , _quant , _r , _quantité) , / .

relative_e([que, !_x], _r, position(_prép , _résu) , _nombre) ->
    groupe_nominal(_x , [_verbe , !_r] , _résu) ,
    verbe(_verbe , [_v , _nomb]) ,
    verbe_position(_v , _prép , _prépinv) , / .
relative_e([que, _verbe , !_x], _r, position(_prép , _résu) , _nombre) ->
    verbe(_verbe , [_v , _nomb]) ,
    verbe_position(_v , _prép , _prépinv) , / ,
    groupe_nominal(_x , _r , _résu) , / .

relative_e([_prép, _qui, est, !_x], _r, position(_prépinv, _résu), _nombre) ->
    prép_position(_prép) ,
    pronomquest(_qui , _y) ,
    être(_est , _nombre) ,
    verbe_position(_v , _prép , _prépinv) , / ,
    groupe_nominal(_x , _r , _résu) , / .
relative_e([_prép, _qui, il, y, a, !_x], _r, position(_prépinv, _résu), _nombre) ->
    prép_position(_prép) ,
    pronomquest(_qui , _y) ,
    verbe_position(_v , _prép , _prépinv) , / ,
    groupe_nominal(_x , _r , _résu) , / .

relative_e([_comp , _adj , que , !_nom] , _r ,
    de_nombre(_entité , _vcomp , _résu) , _nombre) ->
    comparateur(_comp , _zcomp) , / ,
    adjectif(_adj , [_vadj , _genre , _nombre]) ,
    correspond(_vadj , _entité) ,
    associe(_zcomp , _vadj , _vcomp) ,
    groupe_nominal(_nom , _r , _résu) , / .

relative_e([_prép , !_nom] , _r , position(_prép , _résu) , _nombre) ->
    constr_position == non , /* impossible si verbe=poser */

```

```

    prép_position(_prép) ,
    groupe_nominal(_nom , _r , _résu) , / .

relative_e([_part_présent, !_quant], _r , _résu , _nombre)->
    part_présent(_part_présent , _verbe) ,
    verbe_mesure(_verbe , _entité) , / ,
    mesure(_entité , _quant , _r , _résu) , / .
relative_e([_part_présent, !_nom], _r , position(_prépinv, _résu) , _nombre)->
    part_présent(_part_présent , _verbe) ,
    verbe_position(_verbe , _prép , _prépinv) , / ,
    groupe_nominal(_nom , _r , _résu) .

relative_e([dont, !_x], _r , dont(_résu), _nombre) -> / ,
    proposition(_x , _r , _résu) .

relative_e([qui, !_x], _r , qui(_résu) , _nombre)->
    proposition([X, !_x], _r , _résu) .
relative_e([que, !_x], _r, [_verbe, _sujet, _prép, _complément2], _nombre) ->
    groupe_nominal(_x , _y , _sujet) ,
    groupe_verbe(_y , _z , _verbe) ,
    _verbe = [_nomvb, _nombre, _aff, _genrepronom, _nombrepro] ,
    _sujet = [_adjsuj, _adjp, _ajoud, _nomsuj, _defsuj, _genresuj, _nombre] ,
    complemen2(_z , _r , _nomvb , _prép , _complément2) , / .
relative_e([que, !_x] , _r , [_verbe , _sujet , ''] , _nombre) ->
    groupe_verbe(_x , _y , _verbe) ,
    groupe_nominal(_y , _r , _sujet) ,
    _verbe = [_nomvb, _nombre, _aff, _genrepronom, _nombrepro] ,
    _sujet=[_adjsuj, _adjp, _ajoud, _nomsuj, _defsuj, _genresuj, _nombre],/.

/*      verbe      */

groupe_verbe([_v , !_x], _r, [_nomv, _nombre, vrai, _genre, _nombre]) ->
    verbe(_v , [_nomv , _nombre]) , t(_x , [_y, !_r]) ,
    pro_personnel(_y , [_genre , _nombre]) , / .
groupe_verbe([_v, !_r], _r, [_nomv, _nombre, vrai, _genrp, _nombrp])->
    verbe(_v , [_nomv, _nombre]) , / .
groupe_verbe(_p, _r, [_nomv, _nombre, faux, _genrep, _nombrep]) ->
    ne(_p , [_verbe , pas , !_r]) ,
    verbe(_verbe , [_nomv , _nombre]) , / .
groupe_verbe(_p , _r , [_nomv, _nombre, vrai, _genrep, _nombre]) ->
    ne(_p , [_verbe , !_y]) ,
    verbe(_verbe , [_nomv , _nombre]) , t(_y , [_z , pas , !_r]) ,
    pro_personnel(_z , [_genrep , _nombre]) .

verbe_passif([_est , _participe , par , !_r] , _r ,
    [_nomv, _nombre, vrai, _genre, _genre]) ->
    être(_est , _nombre) ,
    participe(_participe , [_nomv , _genre , _nombre]) , / .
verbe_passif(_p , _r , [_nomv, _nombre, vrai, _genre, _genre]) ->

```

```

ne(_p , [_est , !_m]), être(_est, _nombre) ,
t(_m , [_n , pas , _participe , par , !_r]) ,
pro_personnel(_n , [_genre, _nombre]),
participe( _participe , [_nomv , _genre , _nombre]), / .
verbe_passif(_p , _r , [_nomv , _nombre, faux , _genre, _genre]) ->
ne(_p , [_est , pas , _participe , par , !_r]) ,
être(_est , _nombre) ,
participe( _participe , [_nomv , _genre , _nombre]) .

```

/* **compléments** */

```

compléments([] , [] , _nomvb , '' , '' , '') -> / , construction(_nomvb) .
compléments(_p , _r , _nomvb , _complément1 , _prép2 , _complément2) ->
    construction( _nomvb , complément , _prép2 , complément) ,
    complemen(_p , _z , '' , _complément1 ) ,
    complemen(_z , _r , _prép2 , _complément2 ) , / .
compléments(_p , _r , _nomvb , _complément1 , '' , '' ) ->
    construction( _nomvb , complément) ,
    complemen(_p , _r , '' , _complément1 ) , / .
compléments(_p , _r , _nomvb , _complément1 , '' , '' ) ->
    construction( _nomvb , _prép , complément) ,
    complemen(_p , _r , _prép , _complément1 ) , / .
compléments(_p , _r , _nomvb , _complément1 , _prép2 , _complément2) ->
    construction( _nomvb , complément , _prép2 , complément) ,
    complemen(_p , _z , _prép2 , _complément2 ) ,
    complemen(_z , _r , '' , _complément1 ) , / .
compléments(_p , _r , _nomvb , _complément1 , _prép2 , _complément2) ->
    construction( _nomvb , complément , complément) ,
    complemen(_p , _z , '' , _complément1 ) ,
    complemen(_z , _r , '' , _complément2 ) , / .
compléments(_p , _r , _nomvb , _complément1 , _prép2 , _complément2) ->
    construction( _nomvb , _prép1 , complément , _prép2 , complément) ,
    complemen(_p , _z , _prép1 , _complément1 ) ,
    complemen(_z , _r , _prép2 , _complément2 ) .

```

```

complemen(_p , _r , '' , _complément ) ->
    groupe_nominal(_p , _r , _complément ) , / .
complemen([_prép,!_p] , _r , _prép , _complément ) ->
    groupe_nominal(_p , _r , _complément ) , / .
complemen(_p , _p , '' , ['','','','','','']).

```

```

complemen2(_p , _reste , _nomvb , _prép2 , _complément2) ->
    construction( _nomvb , complément , _prép2 , complément) ,
    complemen(_p , _reste , _prép2 , _complément2 ) , / .
complemen2(_p , _reste , _nomvb , _prép2 , _complément2) ->
    construction( _nomvb , _prép1 , complément , _prép2 , complément) ,
    complemen(_p , _reste , _prép2 , _complément2 ) , / .
complemen2(_p , _p , _nomvb , '' , '' ) .

```

adjectif(_adj , _résultat) -> **préadjectif**(_adj , _résultat) , / .
adjectif(_adj , _résultat) -> **postadjectif**(_adj , _résultat) , / .
adjectif(_adj , _résultat) -> **participe**(_adj , _résultat) , / .

préadjectif([_adj , !_r] , _r , _résu) -> **préadjectif**(_adj , _résu) , / .
préadjectif([_comp , _adj , !_r] , _r , _résu) ->
 superlatif(_comp) , / , **préadjectif**(_adj , _résu) , / .
préadjectif([_art , _comp , _adj , !_r] , _r , _résu) ->
 détermine(_art , [défini , _genre , _nombre]) ,
 comparateur(_comp , _vcomp) , / , **préadjectif**(_adj , _résu) , / .
préadjectif(_r , _r , [' , _a , _b]) -> .

postadjectif([_adj , !_r] , _r , _résu) -> **postadjectif**(_adj , _résu) , / .
postadjectif([_adj , !_r] , _r , _résu) -> **participe**(_adj , _résu) , / .
postadjectif([_comp , !_adj] , _r , _résu) ->
 superlatif(_comp) , / ,
 postadjectif(_adj , _r , _résu) , / .
postadjectif([_art , _comp , !_adj] , _r , _résu) ->
 détermine(_art , [défini , _genre , _nombre]) ,
 comparateur(_comp , _vcomp) , / ,
 postadjectif(_adj , _r , _résu) , / .
postadjectif(_r , _r , [' , _a , _b]) -> .

nom([_nom , !_reste] , _reste , _vnom) -> **nom**(_nom , _vnom) , / .
nom(_r , _r , [' , _a , _b]).

grammaire des nombres et quantités

nombre([_nombre , !_reste] , _reste , _nombre) ->
 number(_nombre) , / .
nombre(_nombre , _reste , _valeur) ->
 nombre_mille(_nombre , [millions , !_suite] , _vm) ,
 nombre_mille(_suite , _reste , _v) ,
 _number \= _reste ,
 _valeur := _vm * 1000000 + _v , / .
nombre(_nombre , _reste , _valeur) ->
 nombre_mille(_nombre , _reste , _valeur) ,
 _number \= _reste .

nombre_mille([mille , !_suite] , _reste , _valeur) ->
 nombre_cent(_suite , _reste , _v) ,
 _valeur := 1000 + _v , / .
nombre_mille(_nombre , _reste , _valeur) ->
 nombre_cent(_nombre , [mille , !_suite] , _vm) ,
 nombre_cent(_suite , _reste , _v) ,
 _valeur := _vm * 1000 + _v , / .
nombre_mille(_nombre , _reste , _valeur) ->
 nombre_cent(_nombre , _reste , _valeur) .

```

nombre_cent([cent , !_suite] , _reste , _valeur) ->
    nombre_reste( _suite , _reste , _v) ,
    _valeur := 100 + _v , / .
nombre_cent(_nombre , _reste , _valeur) ->
    nombre_reste( _nombre , [cent , !_suite] , _vm) ,
    nombre_reste( _suite , _reste , _v) ,
    _valeur := _vm * 100 + _v , / .
nombre_cent(_nombre , _reste , _valeur) ->
    nombre_reste( _nombre , _reste , _valeur) .

nombre_reste( [ _nombre , !_reste] , _reste , _nombre) ->
    number( _nombre) , / .
nombre_reste([ _nombre1 , _nombre2 , !_reste] , _reste , _valeur) ->
    dizaine( _nombre1 , _dizaine) ,
    unité( _nombre2 , _unité) ,
    _valeur := _dizaine + _unité , / .
nombre_reste([ _nombre1 , et , _nombre2 , !_reste] , _reste , _valeur) ->
    dizaine( _nombre1 , _dizaine) ,
    unité( _nombre2 , _unité) ,
    _valeur := _dizaine + _unité , / .
nombre_reste([ _nombre , !_reste] , _reste , _valeur) ->
    dizaine( _nombre , _valeur) , / .
nombre_reste([ _nombre , !_reste] , _reste , _valeur) ->
    unité( _nombre , _valeur) , / .
nombre_reste( _nombre , _nombre , 0) .

specif_nombre([au , _comp , !_suite] , _reste ,
    comp(au( _vcomp) , _valeur)) -> / ,
    compareteur( _comp , _vcomp) ,
    nombre( _suite , _reste , _valeur) , / .
specif_nombre([ _comp , de , !_suite] , _reste , comp( _vcomp , _valeur)) -> / ,
    compareteur( _comp , _vcomp) ,
    nombre( _suite , _reste , _valeur) , / .
specif_nombre( _suite , _reste , nombre( _valeur)) ->
    nombre( _suite , _reste , _valeur) , / .

hsh_coded(unité( , )) .
unité(zéro , 0) -> / .    unité(un , 1) -> / .    unité(une , 1) -> / .
unité(deux , 2) -> / .    unité(trois , 3) -> / .    unité( quatre , 4) -> / .
unité(cinq , 5) -> / .    unité(six , 6) -> / .    unité( sept , 7) -> / .
unité(huit , 8) -> / .    unité(neuf , 9) -> / .    unité( dix , 10) -> / .

/* quelques cas particuliers */
unité(onze , 11) -> / .    unité(douze , 12) -> / .    unité( treize , 13) -> / .
unité(quatorze , 14) -> / .    unité( quinze , 15) -> / .    unité( seize , 16) -> / .

hsh_coded(dizaine( , )) .
dizaine(dix , 10) -> / .    dizaine(vingt , 20) -> / .

```

dizaine(trente , 30) -> / . dizaine(quarante , 40) -> / .
dizaine(cinquante , 50) -> / . dizaine(soixante , 60) -> / .
dizaine(soixantedix , 70) -> / . dizaine(septante , 70) -> / .
dizaine(quaatrevingt , 80) -> / . dizaine(octante , 80) -> / .
dizaine(quaatrevingtdix , 90) -> / . dizaine(nonante , 90) .

hsh_coded(dizain(_ , _)) .

dizain(dizaine , 10) . dizain(douzaine , 12).
dizain(quinzaine , 15). dizain(vingtaine , 20).
dizain(trentaine , 30). dizain(quarantaine , 40).
dizain(cinquantaine , 50). dizain(soixantaine , 60).
dizain(centaine , 100). dizain(millier , 1000).
dizain(million , 1000000).
dizain(milliard , 1.0E+9).

hsh_coded(fraction(_ , _)).

fraction(double , [masculin,2]). fraction(triple , [masculin,3]).
fraction(quaduple , [masculin,4]). fraction(centuple , [masculin,100]).
fraction(moitié , [féminin,1/2]). fraction(tiers , [masculin,1/3]).
fraction(quart , [masculin,1/4]). fraction(cinquième , [masculin,1/5]).
fraction(sixième , [masculin,1/6]). fraction(septième , [masculin,1/7]).
fraction(huitième , [masculin,1/8]). fraction(neuvième , [masculin,1/9]).
fraction(dixième , [masculin,1/10]). fraction(onzième , [masculin,1/11]).
fraction(centième , [masculin,1/100]). fraction(millième,[masculin,1/1000]).

unité(poids , _mot , _rapport) -> unité_poids(_mot , _rapport) , / .
unité(volume , _mot , _rapport) -> unité_volume(_mot , _rapport) , / .
unité(longueur , _mot , _rapport) -> unité_longueur(_mot , _rapport) .

hsh_coded(unité_poids(_ , _)) .

hsh_coded(unité_volume(_ , _)) .

hsh_coded(unité_longueur(_ , _)) .

unité_poids(tonne , 1000000) . unité_poids(tonnes , 1000000).
unité_poids(t , 1000000) . unité_poids(kilo , 1000).
unité_poids(kilos , 1000) . unité_poids(kg , 1000).
unité_poids(gramme , 1) . unité_poids(grammes , 1).
unité_poids(g , 1). unité_poids(milligramme , 0.001).
unité_poids(milligrammes , 0.001). unité_poids(mg , 0.001).
unité_volume(litre , 1). unité_volume(litres , 1).
unité_volume(l , 1). unité_longueur(kilomètre , 1000).
unité_longueur(kilomètres , 1000). unité_longueur(km , 1000).
unité_longueur(mètre , 1). unité_longueur(mètres , 1).
unité_longueur(m , 1).

/* **quantité mesurables avec une unité** */

quantité(_entité , _quant , _r , _nombre , _rapport) -> /* 24 kilos 500 grammes */
nombre(_quant , [_unit , !_reste] , _vall) ,
unité(_entité , _unit , _rapport) , / ,

```

quantité2(_entité , _reste , _r , _val2 , _rapport) ,
  _nombre := _val1 * _rapport + _val2 , / .
quantité(_entité , _quant , _r , _nombre , _rapp) -> /* deux douzaine de kilo */
nombre(_quant , [_nomb , de , !_r] , _val1) ,
dizain(_nomb , _val2) ,
unité(_entité , _unit , _rapport) ,
_rapp := _rapport * _val2 ,
_nombre := _val1 * _val2 * _rapport.

```

```

quantité2(_entité , _quant , _r , _nombre , _rapport) ->
  quantité(_entité , _quant , _r , _nombre , _rapp) , / .
quantité2(_entité , _quant , _r , _nombre , _rapport) -> /* 24 kilos 500 */
  nombre(_quant , _r , _val2) ,
  unit_sousentend(_val2 , _nombre , _rapport) , / .
quantité2(_entité , _quant , _quant , 0 , _rapport) .

```

```

unit_sousentend(_val2 , _nombre , _rapport) ->
  _val2 > 9 , _nombre := _val2 * _rapport / 1000 , / .
unit_sousentend(_val2 , _nombre , _rapport) ->
  _val2 < 10 , _nombre := _val2 * _rapport / 10 .

```

```

quantit(_entité , _suite , _reste , comp(_valeur , _vcomp , _résu)) ->
  quantité(_entité , _suite , [de , _comp , que , !_nom] , _valeur , _ra) ,
  comparateur(_comp , _vcomp) ,
  groupe_nominal(_nom , _reste , _résu) , / .
quantit(_entité , _suite , _reste , _valeur) ->
  quantité(_entité , _suite , _reste , _valeur , _rapp) .

```

/* **spécification de quantité** */

```

mesure(_entité , _quant , _r , de_nombre(_entité , nombre(_quantité)) ) ->
  quantité(_entité , _quant , _r , _quantité , _rapp) , / .
mesure(_entité , [_comp , que , !_nom] , _r ,
  de_nombre(_entité , _vcomp , _résu)) ->
  comparateur(_comp , _vcomp) ,
  mesure1(_entité , _nom , _r , _résu) , / .
mesure(_entité , [_comp , _adj , que , !_nom] , _r ,
  de_nombre(_entité , _vcomp , _résu)) ->
  comparateur(_comp , _zcomp) ,
  adjectif(_adj , [_vadj , _genre , _nombre]) ,
  mesure1(_entité , _nom , _r , _résu) ,
  associe(_zcomp , _vadj , _vcomp) , / .
mesure(_entité , [_comp , de , !_nom] , _r ,
  de_nombre(_entité , _vcomp , _résu)) ->
  comparateur(_comp , _vcomp) ,
  mesure1(_entité , _nom , _r , _résu) , / .
mesure(_entité , [_comp , à , !_nom] , _r ,
  de_nombre(_entité , _vcomp , _résu)) ->
  compà(_comp , _vcomp) ,
  mesure1(_entité , _nom , _r , _résu) , / .

```

```

mesure(_entité, [au , _comp , !_nom] , _r ,
        de_nombre(_entité , au(_vcomp) , _résu )) ->
    compateur(_comp , _vcomp) ,
    mesure1(_entité , _nom , _r , _résu) , / .
mesure(_entité, [comme , !_nom] , _r ,
        de_nombre(_entité , autant , _résu )) ->
    groupe_nominal(_nom , _r , _résu) , / .

mesure(_entité , [_art , même , _entit , que , !_nom] , _r ,
        de_nombre(_entité , autant , _résu )) ->
    nomde(_entit , [_entité , _genre , _nombre]) ,
    mesure1(_entité , _nom , _r , _résu) , / .
mesure(_entité , [même , _entit , que , !_nom] , _r ,
        de_nombre(_entité , autant , _résu )) ->
    nomde(_entit , [_entité , _genre , _nombre]) ,
    mesure1(_entité , _nom , _r , _résu) , / .
mesure(_entité , [entre , !_quant] , _r , _résu ) ->
    mesure_entre(et , _entité , _quant , _r , _résu) , / .
mesure(_entité , [de , !_quant] , _r , _résu ) ->
    mesure_entre(à , _entité , _quant , _r , _résu) , / .
mesure(_entité , _quant , _r , de_nombre(_entité , autant , _résu )) ->
    mesure1(_entité , _quant , _r , _résu) , / .
mesure(_entité , _quant , _r , de_nombre(_entité , autant , _résu )) ->
    mesure2(_entité , _quant , _r , _résu) .

mesure1(_entité, _quant , _r , _résu ) ->
    fois(_quant , _reste , _valeur , _comp) , / ,
    mesure_fois(_entité, _reste , _r , _résu , _valeur , _comp) .
mesure1(_entité, _quant , _r , _résu ) ->
    mesure2(_entité, _quant , _r , _résu) .

mesure2(_entité, [_celui , de , !_nom] , _r , _résu ) ->
    pronom_démo(_celui , _x) ,
    groupe_nominal(_nom , _r , _résu) , / .
mesure2(_entité, [_art , _entit , de , !_nom] , _r , _résu ) ->
    détermine(_art , [_typ , _genre , _nombre]) ,
    nomde(_entit , [_entité , _genre , _nombre]) ,
    groupe_nominal(_nom , _r , _résu) , / .
mesure2(_entité, [_art , même , _entit , que , !_nom] , _r , _résu ) ->
    détermine(_art , [_typ , _genre , _nombre]) ,
    nomde(_entit , [_entité , _genre , _nombre]) ,
    groupe_nominal(_nom , _r , _résu) , / .
mesure2(_entité, _nom , _r , _résu ) ->
    groupe_nominal(_nom , _r , _résu) , / .
mesure2(_entité, _quant , _r , nombre(_quantité )) ->
    quantité(_entité , _quant , _r , _quantité , _rapp) , / .

mesure_entre(_prép , _entité , _quant , _r ,
        de_entre(_entité , _quant1 , _quant2)) ->

```

```

quantité(_entité , _quant , [_prép , !_quan] , _quant1 , _rapp) ,
quantité(_entité , _quan , _r , _quant2 , _ra) , / .
mesure_entre(_prép , _entité , _quant , _r ,
              de_entre(_entité , _quant1 , _quant2)) ->
nombre(_quant , [_prép , !_quan] , _quant0) ,
quantité(_entité , _quan , _r , _quant2 , _rapport) ,
_quant1 := _quant0 * _rapport , / .

mesure_fois(_entité , _quant , _r , _résu , _fois , _comp) ->
mesure2(_entité , _quant , _r , _résul) ,
trans_fois(_résul , _résu , _fois , _comp) , / .
mesure_fois(_entité , _quant , _r , fois(_foi , _résu) , _fois , _comp) ->
calcule_fois(_comp , _fois , _foi) ,
groupe_nominal(_quant , _reste , _résu) .

trans_fois( nombre(_valeur) , nombre(_produit) , _fois , _comp) ->
calcule_fois(_comp , _fois , _foi) ,
_produit := _valeur * _foi , / .
trans_fois(_résu , fois(_foi , _résu) , _fois , _comp) -> / ,
calcule_fois(_comp , _fois , _foi) .

calcule_fois(plus , _fois , _foi) -> _foi := _fois , / .
calcule_fois(autant , _fois , _foi) -> _foi := _fois , / .
calcule_fois(moins , _fois , _foi) -> _foi := 1/_fois , / .

fois(_quant , _reste , _valeur , _comp) ->
nombre(_quant , [fois , et , demi , !_r] , _val) ,
fin_fois(_r , _reste , _comp) ,
_valeur := _val + 0.5 , / .
fois(_quant , _reste , _valeur , _comp) ->
nombre(_quant , [fois , !_r] , _valeur) ,
fin_fois(_r , _reste , _comp) , / .
fois([_art , _fraction , de , !_reste] , _reste , _valeur , autant) ->
détermine(_art , [_typ , _genre , _nombre]) ,
fraction(_fraction , [_genre , _valeur]) , / .
fois([_art , _fraction , !_reste] , _reste , _valeur , autant) ->
détermine(_art , [_typ , _genre , _nombre]) ,
fraction(_fraction , [_genre , _valeur]) .

fin_fois([plus , que , !_reste] , _reste , plus) -> / .
fin_fois([moins , que , !_reste] , _reste , moins) -> / .
fin_fois([autant , que , !_reste] , _reste , plus) -> / .
fin_fois(_reste , _reste , plus) -> / .

```

le dictionnaire des mots outils

```

hsh_coded(détermine(_ , _)) .
détermine( le , [défini,masculin,singulier]) -> / .
détermine( la , [défini,féminin,singulier]) -> / .
détermine( un , [indéfini,masculin,singulier]) -> / .

```

détermine(une , [indéfini,féminin,singulier]) -> / .
 détermine(l , [défini,_,singulier]) -> / .
 détermine(les , [défini,_,pluriel]) -> / .
 détermine(des , [indéfini,_,pluriel]) -> / .
 détermine(de , [partitif,_,_a]) -> / .
 détermine(du , [partitif,_,_a]) -> / .
 détermine(ce , [démonstratif,masculin,singulier]) -> / .
 détermine(cette , [démonstratif,féminin,singulier]) -> / .
 détermine(ces , [démonstratif,_,pluriel]) -> / .
 détermine(sa , [possessif,féminin,singulier]) -> / .
 détermine(son , [possessif,masculin,singulier]) -> / .
 détermine(leur , [possessif,masculin,pluriel]) -> / .
 détermine(mon , [possessif,masculin,singulier]) -> / .
 détermine(ma , [possessif,féminin,singulier]) -> / .
 détermine(ton , [possessif,masculin,singulier]) -> / .

hsh_coded(questionif(,_)).
questionif(quel , [_quel,masculin,singulier]) -> marq(_quel),/.
 questionif(quels , [_quels,masculin,pluriel]) -> marq(_quels),/.
 questionif(quelle , [_quelle,féminin,singulier]) -> marq(_quelle),/.
 questionif(quelles , [_quelles,féminin,pluriel]) -> marq(_quelles),/.

hsh_coded(questionnant(_)).
questionnant(où) -> / . questionnant(quand) -> / .
 questionnant(comment) -> / . questionnant(pourquoi) -> / .

ne([ne,!_r] , _r) -> / .
 ne([n,!_r] , _r).
 t([t,!_r] , _r) -> type := question , / .
 t([t,!_r] , _r) -> type := question , / .
 de(du , singulier) .
 de(des , pluriel) .

être(est , singulier) -> / .
 être(sont , pluriel).
 avoir(a , singulier) -> / .
 avoir(ont , pluriel) .

hsh_coded(pronomquest(,_)).
pronomquest(qui , _qui) -> marq(_qui) , / .
 pronomquest(que , _que) -> marq(_que) , / .
 pronomquest(quoi , _quoi) -> marq(_quoi) , / .
 pronomquest(lequel , _lequel) -> marq(_lequel) , / .
 pronomquest(laquelle , _laquelle) -> marq(_laquelle) .

hsh_coded(pronom_démo(,_)).
pronom_démo(celui , [pronom , masculin , singulier]) -> / .
 pronom_démo(celle , [pronom , féminin , singulier]) -> / .
 pronom_démo(celles , [pronom , féminin , pluriel]) -> / .
 pronom_démo(ceux , [pronom , masculin , pluriel]) -> / .

hsh_coded(pro_personnel(_,_)) .
pro_personnel(je , [masculin,singulier]) -> / .
pro_personnel(tu , [masculin,singulier]) -> / .
pro_personnel(il , [masculin,singulier]) -> / .
pro_personnel(ils , [masculin,pluriel]) -> / .
pro_personnel(elle , [féminin,singulier]) -> / .
pro_personnel(elles , [féminin,pluriel]) -> / .

hsh_coded(prép_position(_)) .
prép_position(dans) -> / . prép_position(sur) -> / .
prép_position(sous) -> / . prép_position(avec) .

hsh_coded(superlatif(_)) .
superlatif(très) . superlatif(extrêmement) .
superlatif(excessivement) . superlatif(fort) .
superlatif(bien) . superlatif(excessivement) .

hsh_coded(comparateur(_,_)) .
comparateur(plus , plus) -> / . comparateur(moins , moins) -> / .
comparateur(autant , autant) -> / . comparateur(aussi , autant) .

inverse(plus , moins) -> / . inverse(moins , plus) -> / .
inverse(autant , autant) .

associe(_comp , _vadj , _comp) -> haut_échelle(_vadj) , / .
associe(_comp , _vadj , _vcomp) -> inverse(_comp , _vcomp) .

hsh_coded(compque(_,_)) .
compque(pire , moins) . compque(mieux , plus) .
compque(meilleur , plus) . compque(meilleure , plus) .

hsh_coded(compà(_,_)) .
compà(inférieur , moins) . compà(inférieure , moins) .
compà(supérieur , plus) . compà(supérieure , plus) .
compà(égal , autant) . compà(égale , autant) .
compà(identique , autant) .

le dictionnaire du robot

hsh_coded(préadjectif(_,_)) .
préadjectif(petit , [petite , masculin , singulier]) .
préadjectif(petits , [petite , masculin , pluriel]) .
préadjectif(petite , [petite , féminin , singulier]) .
préadjectif(petites , [petite , féminin , pluriel]) .
préadjectif(gros , [grosse , masculin , _nombre]) .
préadjectif(grosse , [grosse , féminin , singulier]) .
préadjectif(grosses , [grosse , féminin , pluriel]) .
préadjectif(grand , [grosse , masculin , singulier]) .

préadjectif(grande , [grosse , féminin , singulier]) .
 préadjectif(grandes , [grosse , féminin , pluriel]) .
 préadjectif(grands , [grosse , masculin , pluriel]) .
 préadjectif(minuscule , [minuscule , _genre , singulier]) .
 préadjectif(minuscules , [minuscule , _genre , pluriel]) .

hsh_coded(postadjectif(,)) .

postadjectif(rouge , [rouge , _genre , singulier]) .
 postadjectif(rouges , [rouge , _genre , pluriel]) .
 postadjectif(vert , [verte , masculin , singulier]) .
 postadjectif(verte , [verte , féminin , singulier]) .
 postadjectif(verts , [verte , masculin , pluriel]) .
 postadjectif(vertes , [verte , féminin , pluriel]) .
 postadjectif(orange , [orange , _genre , singulier]) .
 postadjectif(oranges , [orange , _genre , pluriel]) .
 postadjectif(cubique , [boite , _genre , singulier]) .
 postadjectif(cubiques , [boite , _genre , pluriel]) .
 postadjectif(sphérique , [sphère , _genre , singulier]) .
 postadjectif(sphériques , [sphère , _genre , pluriel]) .
 postadjectif(pyramidal , [pyramide , masculin , singulier]) .
 postadjectif(pyramidale , [pyramide , féminin , singulier]) .
 postadjectif(pyramidaux , [pyramide , masculin , pluriel]) .
 postadjectif(pyramidales , [pyramide , féminin , pluriel]) .
 postadjectif(lourde , [lourde , féminin , singulier]) .
 postadjectif(lourdes , [lourde , féminin , pluriel]) .
 postadjectif(lourd , [lourde , masculin , singulier]) .
 postadjectif(légère , [légère , féminin , singulier]) .
 postadjectif(légères , [légère , féminin , pluriel]) .
 postadjectif(léger , [légère , masculin , singulier]) .

hsh_coded(nom(,)) .

nom(boite , [boite , féminin , singulier]) .
 nom(boites , [boite , féminin , pluriel]) .
 nom(sphère , [sphère , féminin , singulier]) .
 nom(sphères , [sphère , féminin , pluriel]) .
 nom(pyramide , [pyramide , féminin , singulier]) .
 nom(pyramides , [pyramide , féminin , pluriel]) .
 nom(objet , [objet , masculin , singulier]) .
 nom(objets , [objet , masculin , pluriel]) .
 nom(main , [main , féminin , singulier]) .
 nom(table , [table , féminin , singulier]) .
 nom(robot , [robot , masculin , singulier]) .
 nom(opérateur , [opérateur , masculin , singulier]) .
 nom(pile , [pile , féminin , singulier]) .
 nom(X , [_x,_a,_b]) .
 nom(Y , [_y,_a,_b]) .

```

hsh_coded(nomde(_, _)) .
nomde( couleur , [couleur , féminin , singulier]) .
nomde( taille , [taille , féminin , singulier]) .
nomde( forme , [forme , féminin , singulier]) .
nomde( poids , [poids , masculin , singulier]) .
nomde( volume , [volume , masculin , singulier]) .
nomde( nom , [nom , masculin , singulier]) .
nomde( X , [_x,_g,_]) .
nomde( Y , [_y,_g,_]) .

hsh_coded(nompropre(_, _)) .
nompropre( _surnom , [surnom( _numéro),_])->surnom(_surnom,_numéro),/.
nompropre( X , [_X,_]) -> / .
nompropre( Y , [_Y,_]) -> / .

hsh_coded(verbe(_, _)) .
verbe( est , [être,singulier]) -> / .
verbe( sont , [être,pluriel]) -> / .
verbe( contient , [contenir,singulier]) -> / .
verbe( contiennent , [contenir,pluriel]) -> / .
verbe( supporte , [supporter,singulier]) -> / .
verbe( supportent , [supporter,pluriel]) -> / .
verbe( recouvre , [recouvrir,singulier]) -> / .
verbe( recouvrent , [recouvrir,pluriel]) -> / .
verbe( pèse , [peser,singulier]) -> / .
verbe( pèsent , [peser,pluriel]) -> / .
verbe( mesure , [mesurer,singulier]) -> / .
verbe( fait , [faire,singulier]) -> / .
verbe( font , [faire,pluriel]) -> / .
verbe( _verbe , [_nomv,singulier]) -> impératif(_verbe , _nomv) , / .

hsh_coded(impératif(_, _)) .
impératif(pose , poser) . impératif(mets , poser) .
impératif(place , poser) . impératif(prends , prendre) .
impératif(ouvre , ouvrir) . impératif(ferme , fermer) .
impératif(donne , donner) . impératif(reprends , reprendre) .
impératif(génère , générer) . impératif(nomme , nommer) .
impératif(vide , vider) . impératif(désintègre , désintégrer) .
impératif(regarde , regarder) . impératif(liste , lister) .
impératif(dessine , dessiner) .

hsh_coded(part_présent(_, _)) .
part_présent(pesant , peser) . part_présent(faisant , faire) .
part_présent(mesurant , mesurer) . part_présent(contenant , contenir) .
part_présent(supportant , supporter) . part_présent(recouvrant , recouvrir) .

hsh_coded(participe(_, _)) .

```

```

hsh_coded(construction(_)) .
hsh_coded(construction(_,_)) .
hsh_coded(construction(_,_,_)) .
hsh_coded(construction(_,_,_,_)) .
construction( poser      , complément , sur , complément) ->
                                                    constr_position := oui.
construction( poser      , complément , dans , complément) ->
                                                    constr_position := oui.

construction( donner      , complément , à , complément ) .
construction( nommer      , complément , complément ) .
construction( poser      , complément).  construction( prendre      , complément).
construction( ouvrir      , complément).  construction( fermer      , complément).
construction( générer      , complément).  construction( regarder      , complément).
construction( reprendre    , complément).  construction( vider      , complément).
construction( recouvrir    , complément).  construction( contenir      , complément).
construction( supporter    , complément).  construction(désintégrer,complément).
construction( dessiner     , complément).  construction( dessiner) .
construction( regarder) .                construction( lister) .

```

Avant de régler son compte au robot, et pour voir réagir cette grammaire, on peut se proposer de transformer des phrases simples en expressions prolog, faits ou questions et, selon le cas, de rajouter ces faits dans la base de faits grâce à la primitive "assert" ou de faire rechercher les réponses aux questions. les pronoms de la phrase étant les variables de l'expression prolog.

le robot ouvre la boîte
le robot n ouvre pas la sphère
si le robot ouvre la boîte alors la boîte est ouverte
qui ouvre la boîte rouge?

297

```

ouvrir(robot , boîte , vrai ).
ouvrir(robot , sphère , faux ).
être(boîte , ouverte , vrai ) -> ouvrir(robot , boîte , vrai).
ouvrir(_qui , boîte(rouge) , vrai) ?

```

la formule prolog

le résultat de l'analyse d'une proposition est une clause Prolog évaluée ici très sommairement par le sous programme "formule_prop" soit :

```
_formule = _nomverbe(_nomsujet, _nomcpl1, _prép, _nomcpl2, _vérité)
```

Et c'est la clause constituée par cette formule que l'on exécute en fin d'analyse. Si c'est un fait, ou une règle, on le rajoute à la base de fait par la primitive "assert". Si c'est une question, c'est à dire si la phrase se termine par un "?" ou si elle contient les pronoms "qui", "quoi", "quel", ... on fait exécuter cette clause dans laquelle les pronoms ont été remplacés par des variables sur lesquelles on a utilisé le prédicat "mark" qui fera éditer leur valeur à la fin de l'exécution de la clause.

le programme qui comprend - comprend.pro

```

load('grammair.pro') .
load('nombre.pro') .
load('outil.pro') .
load('dico.pro') .

/*      formule prolog      */

formule_prop([_sujet, _verbe, _complément, ' ', ''],
              _nomv(_fsujet, _fcompl, _aff)) ->
    _verbe = [_nomverbe, _nombrevb, _aff, _genrep, _genrevb],
    function(_nomv(, , ,), _nomverbe),
    formule_gn(_sujet, _fsujet),
    formule_gn(_complément, _fcompl), /.
formule_prop([_sujet, _verbe, _complément1, _prép, _complément2],
              _nomv(_fsujet, _fcompl1, _prép, _fcompl2, _aff)) ->
    _verbe = [_nomverbe, _nombrevb, _aff, _genrep, _genrevb],
    function(_nomv(, , , ,), _nomverbe),
    formule_gn(_sujet, _fsujet),
    formule_gn(_complément1, _fcompl1),
    formule_gn(_complément2, _fcompl2), /.
formule_prop(_x, ' ') ->
    traduisible := non.      /* non traduisible en prolog */

formule_gn([' ', ' ', rel(' ', ' '), _nom, _def, _genre, _nombre], _nom) -> /.
formule_gn([_x, ' ', rel(' ', ' '), _nom, _def, _genre, _nombre], _nomgn(_x)) ->
    _x \= ' ', function(_nomgn(, ), _nom), /.
formule_gn([' ', _x, rel(' ', ' '), _nom, _def, _genre, _nombre], _nomgn(_x)) ->
    _x \= ' ', function(_nomgn(, ), _nom), /.
formule_gn([_x, _y, rel(' ', ' '), _nom, _def, _genre, _nombre], _nomgn(_x, _y)) ->

```

```

_x \=' ', _y \=' ', function(_nomgn(_), _nom), /.
formule_gn( [' ', ' ', rel(de(_ajoutde), ' '), _nom, _def, _genre, _nombre] ,
            _nomgn(_nom2) ) ->
_ajoutde = [' ', ' ', rel(' ', ' '), _nom2, _def2, _genre2, _nombre2] ,
function(_nomgn(_), _nom) .

/* execution de la clause prolog correspondant à la phrase */

executer( _formule , _type) ->
    writeln(_type) , /* afficher le résultat d'analyse */
    writeln(formule, ' = ', _formule) ,
    traduisible == oui ,
    enprolog(_formule , _type) .

enprolog( _formule , ordre) -> /.
enprolog( _formule , affirmation) -> /* une affirmation */
    putln("compris, je rajoute ce fait") ,
    assert( _formule , [] ) , /.
enprolog([_conclusion , _conditions] , règle) -> /* une règle */
    putln("compris, je rajoute cette règle") ,
    assert(_conclusion , _conditions) , /.
enprolog( _formule , question) -> /* une question */
    putln("je recherche les reponses") ,
    trouvé := non ,
    _formule ,
    trouvé := oui ,
    répond_oui .
enprolog( _formule , question) ->
    trouvé == non ,
    putln("je ne sais pas") ,
    unmark.

initialisation. /* programme d'initialisation : neant */
marq(_x) -> mark(_x) . /* marquer les pronoms qui, quel,.. pour la réponse */

répond_oui -> pronom_quest == oui , / . /* ne dire oui que s'il n'y a pas */
répond_oui -> putln("oui"). /* de qui, que ou quoi dans la question */

```

On lance le programme par la question :

robot?

résultats d'analyse

```

le robot ouvre la boîte rouge
et la boîte rouge est ouverte par le robot
type de phrase = affirmation
sujet          = robot, singulier, défini
verbe          = ouvrir, affirmatif
complément     = boîte, singulier, défini

```

ajout du compl. = rouge
formule = ouvrir(robot , boite(rouge) , vrai)

les sphères ne supportent pas les boites
et les boites ne sont pas supportées par les sphères
type de phrase = affirmation
sujet = sphère,pluriel,défini
verbe = supporter,negatif
complément = boite,pluriel,défini
formule = supporter(sphère , boite , faux)

est ce que le robot ouvre la pyramide
le robot ouvre t-il la pyramide
le robot n ouvre t-il pas la pyramide
la pyramide est elle ouverte par le robot
la pyramide n est t-elle pas ouverte par le robot
et est ce que la pyramide est ouverte par le robot
type de phrase = question
sujet = robot,singulier,défini
verbe = ouvrir,affirmatif
complément = pyramide,singulier,défini
formule = ouvrir(robot , pyramide , vrai)?

le robot ferme quoi
que ferme le robot
formule = fermer(robot , _quoi , vrai)?

qui ferme la boite
formule = fermer(_qui , boite , vrai)?

quelle boite ouvre le robot ?
formule = ouvrir(robot , boite(_quelle) , vrai) ?

le robot qui désintègre la boite est rouge
type de phrase = affirmation
sujet = robot,singulier,défini
relative en qui = verbe = désintégrer,affirmatif
complément = boite,singulier,défini
verbe = être
complément = rouge
formule = être(robot(désintégrer(boite)) , rouge , vrai)

la sphère que le gros objet rouge supporte est verte
la sphère que supporte le gros objet rouge est verte
type de phrase = affirmation
sujet = sphère,singulier,défini
relative en que = sujet = objet,singulier,défini
ajout du sujet = gros,rouge
verbe = supporte
300

verbe = être
 complément = verte
 formule = être(sphère(supporter(objet(gros,rouge))), verte, vrai)

le robot et les blocs

représentation du monde

Chaque objet est représenté en interne par son numéro d'ordre de création, le premier créé, le deuxième créé, ... et trois relations suffisent pour décrire l'état de notre petit monde :

```

objet(numéro , article , taille , forme , couleur , poids)
état(numéro , état , préposition , objet , pos(x , y))
    l'état peut prendre les valeurs "ouvert" ou "fermé"
    la préposition et le numéro d'objet spécifient la position
    x et y sont les coordonnées de l'objet, pour le dessiner
surnom(surnom , numéro)
  
```

Par exemple une petite boîte rouge ouverte posée sur la table, pesant 5 Kg et ayant été appelée Titine est représentée par :

```

objet(3 , une , petite , boîte , rouge , 5000)
état(3 , ouverte , sur , 1 , pos(24 , 40))
surnom(Titine , 3)
  
```

L'objet 1 étant la table et l'objet 2 étant la main du robot.
la plupart des actions du robot n'affectent que la relation "état".

contraintes sémantiques - semantic.pro

C'est ici que les mots prennent leurs sens par les relations qu'ils ont les uns par rapport aux autres : Les adjectifs sont classés par les relations "taille", "poids", "forme", "couleur", et "état" selon la quantité qu'ils définissent. Les quantités comparables selon une échelle numérique sont précisées par les relations "haut_échelle" et "verbe_mesure". les verbes de position sont associées à leur préposition et à son inverse par la relation "verbe_position". Les objets sont de mêmes classés selon leurs caractéristiques par les relations "prenable", "ouvrable" et "socle".

```

vtaille(petite , 1) -> / .
vtaille(grosse , 2) -> / .
vtaille(minuscule,0) .
taille(_ taille , _t) -> vtaille(_ taille , _t) , / .
taille(_ taille , _t) ->      putln(_ taille , " n'est pas une taille") , fail.
  
```

```

vpoids(lourde) .
vpoids(légère) .
  
```

```

haut_échelle(lourde) .          /* valeur haute des échelles numériques */
haut_échelle(grosse) .
  
```

```

correspond(_x , taille) -> vtaille(_x , _t) , / .
  
```

```

correspond(_x , poids) -> vpoids(_x ) , / .
correspond(petite , _tout) .          /* toute quantité peut être petite */
correspond(grande , _tout) .
correspond(grosse , _tout) .

verbe_mesure(peser , poids) .  /* correspondance verbe et entité mesurée */
verbe_mesure(faire , _jesaispas) .    /* entité sera indiquée par l'unité */
verbe_mesure(mesurer , longueur) .

verbe_position(contenir , dans , autour)./*correspondance verbe préposition*/
verbe_position(supporter , sur , sous) .
verbe_position(recouvrir , sous , sur) .

vforme(boite) -> / .
vforme(pyramide) -> / .
vforme(sphère) .

forme(_forme) -> vforme(_forme) , / .
forme(_forme) -> putln(_forme , " n'est pas une forme") , fail.

vcouleur(rouge) -> / .
vcouleur(verte) -> / .
vcouleur(orange) .

couleur(_couleur) -> vcouleur(_couleur) , / .
couleur(_couleur) -> putln(_couleur , " n'est pas une couleur") , fail .

état(ouvrir , ouverte) .
état(fermer , fermée) .

prenable(boite) -> / .          /* prendre objet */
prenable(sphère) -> / .
prenable(pyramide) -> / .
prenable(_objet) -> putln(_objet , " n'est pas prenable") , fail.

ouvrable(boite) -> / .          /* ouvre objet  ou poser xx dans objet */
ouvrable(main) -> / .
ouvrable(_objet) -> putln("ouvrir ou fermer " , _objet , " c'est dur") , fail.

socle(boite) -> / .             /* poser xx sur objet */
socle(table) -> / .
socle(_objet)->putln("empiler qqchose sur " , _objet , " est impossible") , fail.

/* sémantisme de la taille du socle pour y mettre dedans */
veriftaille(_taille , grosse) -> putln("trop grosse pour entrer") , / , fail.
veriftaille(minuscule , _taille)->putln("trop grosse pour entrer") , / , fail.
veriftaille(_taille , _taille) -> putln("c'est trop juste") , / , fail.
veriftaille(_taillec , _taille).

```

```

/* sémantisme de la taille du socle pour empiler dessus */
veroftaille(_taille , _taille) -> / .
veroftaille(_taille , grosse) ->putln("ce n'est pas stable") , / , fail.
veroftaille(minuscule , _x) -> putln("ce n'est pas stable") , / , fail.
veroftaille(_taillec , _taille) .

verifdiff(_objet , _objet) -> / ,
    putln("à propos, tu connais l'histoire du disque rayé") , fail .
verifdiff(_socle , _objet) .

```

formule des groupes nominaux

L'analyse rend des groupes nominaux sous la forme de listes :

_groupe_nominal = [_préadj , _postadj , rel(_rel1,_rel2) , _nom]

_nom peut être :

- un nom de forme : boîte , cube ou pyramide
- les noms spéciaux : objet, main et opérateur
- un surnom de forme : surnom(numéro)
- la chaîne ' ' , si il n'a pas été spécifié dans la phrase

Les deux relatives, _rel1 et _rel2, peuvent être de la forme :

quiest(_adjectif)

- correspondant à : qui est ouverte
- ou à : qui est rouge

de(_nomde , _adjectif)

- de couleur rouge
- dont la couleur est rouge

de(_nomde , _groupe_nominal)

- dont la couleur est celle de la sphère

de_nombre(_nomde , nombre(_nombre))

- dont le poids est de 3 kilos
- qui pèse 3 kilos

de_nombre(_nomde , _comp , nombre(_nombre))

- qui pèse plus de 3 kilos

de_nombre(_nomde , au(_comp) , nombre(_nombre))

- qui pèse au moins 3 kilos

de_nombre(_nomde , _comp , _groupe_nominal)

- dont le poids est celui de la sphère
- qui pèse plus que la sphère
- qui pèse au moins le poids de la sphère

de_nombre(_nomde , _comp , fois(_nombre , _groupe_nominal))

- qui pèse 3 fois le poids de la sphère
- qui pèse moins de 3 fois le poids de la sphère
- qui pèse moins de 3 fois plus que la sphère
- qui pèse moins de 3 fois moins que la sphère
- qui pèse au plus 3 fois le poids de la sphère

de_entre(_nomde , _valeur_min , _valeur_max)

- qui pèse entre 2 et 6 kilos

la chaîne '' si il n'y a pas de spécification dans la phrase

_préadj, _postadj et _adjectif peuvent être tous les adjectifs :
petite, rouge, cubique, ouverte
_comp est un comparateur et vaut : plus, moins , autant
ou au(plus) , au(moins)

A partir de cela, il faut retrouver les caractéristiques de l'objet :

_taille , _forme , _couleur , _poids , _état , _prép , _numobjet

Il faut noter que d'une part certaines caractéristiques peuvent ne pas être spécifiées dans la phrase et donc restent à l'état de variables prolog et que d'autre part _préadj et _postadj doivent tous être une taille, une forme, une couleur ou un état et il faut donc étudier chaque cas et initialiser l'élément correspondant de la liste des caractéristiques.

Le traitement des pronoms s'effectue comme suit :

Si l'objet est nommé par le pronom "la" (donc _préadj, _postadj, _nom et _rel sont des variables) le numéro cherché se trouve dans la variable globale "pronom". Si _forme reste à l'état de variable et que _nom n'est pas le mot "objet" c'est que la phrase sous-entend une forme : ("la bleue" ou "celle qui est bleue") et la valeur correspondante pour _forme se trouve dans la variable globale "pro". Cette liste, plus ou moins complète, de caractéristiques permet alors de retrouver les numéros du ou des objets ayant ces caractéristiques et sur lesquels il faut agir.

/* calcul de la formule d'un groupe nominal */

```
formule_gn(_formule,_num,_taille,_forme,_couleur,_poids,_état,_prép,_objet)->
  _formule = [ _tail,_coul,_rel, surnom(_num) ,
              _def,_genr,_nomb] , / ,
  objet(_num , _art , _taille , _forme , _couleur , _poids),
  état(_num , _état , _prép , _objet , _pos) , / .
formule_gn( _formule, _num, _taille, _forme, _couleur, _poids, _état, _prép, _objet)->
  _formule = [ _préadj, _postadj, _rel, _nom, _def, _genre, _nombre] ,
  _préadj='', _postadj='', _nom='',
  _num := pronom , / ,
  objet(_num , _art , _taille , _forme , _couleur , _poids),
  état(_num , _état , _prép , _objet , _pos) .
formule_gn( _formule, _num, _taille, _forme, _couleur, _poids, _état, _prép, _objet)->
  _formule=[ _préadj, _postadj,rel( _rel1, _rel2), _nom, _def, _genre, _nombre],
  place( _préadj , _taille , _forme , _couleur , _état) ,
  place( _postadj , _taille , _forme , _couleur , _état) ,
  place_nom( _nom , _forme ) ,
  pra := pro , pro := _forme ,
  place_rel( _rel1, _taille, _forme, _couleur, _poids, _état, _prép, _objet),
  place_rel( _rel2, _taille, _forme, _couleur, _poids, _état, _prép, _objet),
  pro := pra ,
  objet(_num , _art , _taille , _forme , _couleur , _poids),
  état(_num , _état , _prép , _objet , _pos) .
```



```

place_rel( position(_prép , _formule) ,
           _taille , _forme , _couleur , _poids , _état , _prép , _objet) -> / ,
           formule_gn( _formule , _objet , _t , _f , _c , _po , _é , _p , _o) ,
           objet( _n , _a , _taille , _forme , _couleur , _poids) ,
           état( _n , _et , _prép , _objet , _pos) .
place_rel( de_nombre(poids , _comp , fois( _fois , _formule)) ,
           _taille , _forme , _couleur , _poids , _état , _prép , _objet) -> / ,
           formule_gn( _formule , _obj , _t , _f , _c , _po , _é , _p , _o) ,
           objet( _n , _a , _taille , _forme , _couleur , _poids) ,
           _poid := _po * _fois ,
           verif_de( _comp , _poids , _poid ) .
place_rel( de_nombre(poids , _comp , _formule) ,
           _taille , _forme , _couleur , _poids , _état , _prép , _objet) -> / ,
           formule_gn( _formule , _obj , _t , _f , _c , _po , _é , _p , _o) ,
           objet( _n , _a , _taille , _forme , _couleur , _poids) ,
           verif_de( _comp , _poids , _po ) .
place_rel( de_nombre(taille , _comp , _formule) ,
           _taille , _forme , _couleur , _poids , _état , _prép , _objet) ->
           formule_gn( _formule , _obj , _t , _f , _c , _po , _é , _p , _o) ,
           objet( _n , _a , _taille , _forme , _couleur , _poids) ,
           taille( _taille , _v1) ,
           taille( _t , _v2) ,
           verif_de( _comp , _v1 , _v2 ) .
place_rel( de_nombre(forme , autant , _formule) ,
           _taille , _forme , _couleur , _poids , _état , _prép , _objet) -> / ,
           formule_gn( _formule , _obj , _t , _forme , _c , _po , _é , _sur , _n) .
place_rel( de_nombre(couleur , autant , _formule) ,
           _taille , _forme , _couleur , _poids , _état , _prép , _objet) -> / ,
           formule_gn( _formule , _obj , _t , _f , _couleur , _po , _é , _sur , _n) .

verif_de( autant , _poids , _poids) -> / .
verif_de( plus , _poids , _poid) -> / , _poids > _poid .
verif_de( moins , _poids , _poid) -> / , _poids < _poid .
verif_de( au(plus) , _poids , _poid) -> / , _poids <= _poid .
verif_de( au(moins) , _poids , _poid) -> / , _poids >= _poid .

```

```

formule_prop( _proposition , _proposition) .

```

les actions du robot- action.pro

L'analyse de la phrase rend des structures qui sont :

- des ordres :

```

execute_ordre( _verbe )
execute_ordre( _verbe , _formule)
execute_ordre( _verbe , _formule1 , _prép , _formule2)

```

- des questions :

```

estceque( _formule)
combien( _formule)

```

```
quoi( _prép , _formule)
où( _formule)
quel( _entité , _formule)
existe( _spécif_nombre , _formule)
existe( _comparateur , _formule)
existe( _comparateur , _formule1 , _formule2)
```

"formule" représente un groupe nominal, tel qu'il a été décrit au paragraphe précédent.

- **des commandes de gestion**, exécutées directement par l'analyseur

- **des affirmations**, qui ne sont pas traitées par le robot

Le robot considère d'abord la formule du groupe nominal qui est rendue par l'analyse et en déduit, grâce au sous programme "formule_gn", les numéros des objets concernés par l'ordre ou la question. Le robot transforme alors l'ordre qui lui a été donné en un ou plusieurs ordres élémentaires :

```
informer sur l'état des objets visibles
prendre un objet dans la main
ouvrir une boîte
fermer une boîte
trouver une place libre sur la table
poser sur la table l'objet qui est dans la main
mettre sur une boîte l'objet qui est dans la main
mettre dans une boîte l'objet qui est dans la main
créer un nouvel objet dans la main
retirer du jeu l'objet qui est dans la main
donner un nom à un objet
donner un objet fermé à l'opérateur
demander à l'opérateur de rendre un objet qui lui a été donné
```

Par exemple, si on lui demande de placer une sphère dans une boîte rouge elle-même surmontée d'une boîte verte , alors qu'il a une pyramide en main, après avoir déterminé de quelle boîte et de quelle sphère il s'agit, il va :

```
trouver une place libre sur la table
poser la pyramide sur cette place libre
prendre la boîte verte
poser la boîte verte sur la table
ouvrir la boîte rouge
prendre la sphère
poser la sphère dans la boîte rouge
```

initialisation ->

```
clear_module ,
dessine_init ,
init_relations ,
pronom := 0 ,
pro := 0 ,
pronb := 0 ,
dedans := 0 ,
enregistrer := non .
```

init_relations -> /* mise des objets en position de départ */

```
main := 0 ,
nb_objet := 2 ,
assert( objet( 0 , 'l' , " , opérateur , " , " ) , [] ) ,
assert( objet( 1 , la , " , table , " , " ) , [] ) ,
assert( objet( 2 , la , " , main , " , " ) , [] ) ,
assert( état( 1 , " , " , " , pos(ligtable,40)) , [] ) ,
assert( état( 2 , " , " , " , pos(ligmain,colmain)) , [] ) .
```

/* ne pas marquer les pronoms qui, quel,.. pour la réponse */

marq(_x).

```
executer( _formule , ordre) -> / , writeln(_formule) , _formule .
executer( _formule , question) -> / , writeln(_formule) , _formule .
executer( _formule , affirmation) -> enregistre(_formule , [] ) .
executer( [_conc , _cond] , règle) -> enregistre(_conc , _cond).
```

```
enregistre(_conclusion , _conditions) ->
    enregistrer == oui ,
    putln("compris , je rajoute ça à la base") ,
    assert(_conclusion , _conditions) .
```

/* **execution des ordres** */

```
hsh_coded(execute_ordre( , _ )) .
execute_ordre(générer , _formule) ->
    génère(_numéro , _formule) .
execute_ordre(générer , _nom , ' ' , _formule) ->
    génère(_numéro , _formule) ,
    nomme(_nom , _numéro) .
```

```
génère(_nb_objet , _formule) ->
    _formule=[_taille,_couleur,rel(de_nombre(poids,nombre(_poids)), ' '),
    _forme , _def , _genre , singulier],
    novar(_taille),novar(_couleur),novar(_forme),
    taille(_taille , _ta) ,
    forme(_forme) ,
    couleur(_couleur) ,
    nb_objet := nb_objet + 1 ,
    _nb_objet := nb_objet ,
```

```

    pronom := _nb_objet ,
    pro := _forme ,
    poser ,
    assert(objet(_nb_objet, la, _taille, _forme, _couleur, _poids), []),
    assert(état(_nb_objet, fermée, dans, 2, pos(ligmain-1, colmain)), []),
    main := _nb_objet ,
    dessine_objet(_nb_objet) ,
    afficheLn(* j'ajoute', _nb_objet) , / .

nomme(_nom , _numéro) -> détermine(_nom , _x) , / , fail.
nomme(_nom , _numéro) -> pronom(_nom) , / , fail.
nomme(_nom , _numéro) -> vtaille(_nom, _x) , / , fail.
nomme(_nom , _numéro) -> vforme(_nom) , / , fail.
nomme(_nom , _numéro) -> vcouleur(_nom) , / , fail.
nomme(_nom , _numéro) ->
    assert(surnom(_nom , _numéro) , []).

execute_ordre(donner , _formule , à , _opérateur) -> / ,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
        _état, _prép, _objet),

    prenable(_forme) ,
    pronom := _numéro ,
    dégage(_numéro) ,
    ferme(_numéro) ,
    prendre(_numéro) ,
    nouvelle_pos(_numéro , avec , 0) ,
    main := 0 .

execute_ordre(reprendre , _formule) -> / ,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
        _état, avec, 0),

    pronom := _numéro ,
    poser ,
    nouvelle_pos(_numéro , dans , 2) .

execute_ordre(vider , _formule) -> / ,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
        _état, _prép, _objet),

    pronom := _numéro ,
    ouvrable(_forme) ,
    vide(_numéro) .

execute_ordre(désintégrer , _formule) -> / ,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
        _état, _prép, _objet),

    prenable(_forme) ,
    pronom := 0 ,
    dégage(_numéro) ,
    vide(_numéro) ,
    prendre(_numéro) ,
    main := 0 ,
    afficheLn(* je supprime' , _numéro) ,
    efface_objet(_numéro) ,

```

```

    replace(objet(_numéro, _art, _taille, _forme, _couleur, _poids), [],
            objet(" ", " ", " ", " ", " ", " "), []),
    replace(état(_numéro, _état, _préposition, _obj, _pos), [],
            état(" ", " ", " ", " ", " "), []) / ,
    replace(surnom(_nom, _numéro), [], surnom(" ", " "), []).
execute_ordre(ouvrir, _formule) -> / ,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
                _état, _prép, _objet),

    pronom := _numéro ,
    ouvrable(_forme) ,
    ouvre(_numéro) .
execute_ordre(fermer, _formule) -> / ,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
                _état, _prép, _objet),

    pronom := _numéro ,
    ouvrable(_forme) ,
    ferme(_numéro) .
execute_ordre(prendre, _formule) -> / ,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
                _état, _prép, _objet),

    pronom := _numéro ,
    prenable(_forme) ,
    main \== _numéro ,      /* pas déjà dans la main */
    dégage(_numéro) ,
    prendre(_numéro) , / .
execute_ordre(poser,
    [' ', ' ', rel(de(_formule), ' '), pile, défini, féminin, singulier],
    sur, _objet) ->
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
                _état, _prép, _obj),

    prenable(_forme) ,
    formule_gn(_objet, _numéro, _taille, _forme, _couleur, _poids,
                _état, _prép, _objetc),

    pronom := _numéro ,
    socle(_formec) ,
    ferme(_numéro) ,
    pile := [] , fait_pile(_numéro) ,
    dégage(_numéro) ,
    n := _numéro , / ,
    pop(pile, _num) ,
    prendre(_num) ,
    _n := n , nouvelle_pos(_num, sur, _n) , n := _num ,
    fail .
fait_pile(_numéro) ->
    état(_objet, _ouvert, sur, _numéro, _pos) ,
    fait_pile(_objet) ,
    fail .
fait_pile(_numéro) -> pushfirst(pile, _numéro) .

execute_ordre(poser, _formule, dans, _boite) ->

```

```

formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
           _état,_prép,_objet),

prenable(_forme) , pro := _forme ,
formule_gn(_boite,_numéroc,_taillec,_formec,_couleurc,_poidsc,
           _étatc,_prépc,_objetc),

pronom := _numero ,
ouvrable(_formec) ,
verifdiff(_numéro , _numéroc) ,
veriftaille(_taillec , _taille) ,
ouvre(_numéroc) ,
dégage(_numéro) ,
prendre(_numéro) ,
nouvelle_pos(_numéro , dans , _numéroc) , / .

execute_ordre(poser , _formule , sur , _boite ) ->
formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
           _état,_prép,_objet),

prenable(_forme) , pro := _forme ,
formule_gn(_boite,_numéroc,_taillec,_formec,_couleurc,_poidsc,
           _étatc,_prépc,_objetc),

pronom := _numéro ,
socle(_formec) ,
verifdiff(_numéro , _numéroc) ,
veroftaille(_taillec , _taille) ,
ferme(_numéroc) ,
dégage(_numéro) ,
prendre(_numéro) , /* si jamais il etait dans la boite */
ferme(_numéroc) ,
prendre(_numéro) , /* reprendre l objet apres fermeture */
nouvelle_pos(_numéro , sur , _numéroc) , / .

execute_ordre(poser , _formule ) ->
formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
           _état,_prép,_objet),

pronom := _numéro ,
main == _numéro ,
poser , / .

execute_ordre(nommer , _nom , ' ' , _formule ) -> / ,
formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
           _état,_prép,_objet),/ ,

nomme(_nom , _numéro), / .

execute_ordre(pile, _formule) -> / ,
formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
           _état,_prép,_objet),

pronom := _numéro ,
socle(_forme) ,
afficheIn('pile de' , _numéro) ,
état(_obj , _état , sur , _numéro , _pos) ,
writeIn , édite_pile(_obj) ,
fail .

```

```

édite_pile(_numéro) ->
    afficheLn(' ', _numéro),
    état(_obj, _état, sur, _numéro, _pos),
    édite_pile(_obj),
    fail.

execute_ordre(lister) -> /, affiche_tout(tout) .
execute_ordre(regarder) -> /, affiche_tout(visible) .
execute_ordre(regarder, _formule) -> /,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids, _état, _prép, _objet),
    pronom := _numéro,
    informer(_numéro, visible) .
execute_ordre(dessiner) -> /, dessine_tout .
execute_ordre(dessiner, _formule) -> /,
    formule_gn(_formule, _numéro, _taille, _forme, _couleur, _poids,
        _état, _prép, _objet),
    pronom := _numéro,
    dessine_objet(_numéro) .

```

/* fonctions de base pour les actions du robot */

```

dégage(_numéro) ->
    état(_objet, _état, sur, _numéro, _pos),
    dégage(_objet),
    prendre(_objet),
    fail .

dégage(_numéro) ->
    état(_numéro, _état, dans, _objet, _pos),
    _objet \== 2,
    dégage(_objet),
    ouvre(_objet),
    fail .

dégage(_numéro) ->
    état(_numéro, _état, avec, 0, _pos), /,
    put("* donc tu me rends "),
    afficheLn(" ", _numéro),
    poser,
    nouvelle_pos(_numéro, dans, 2),
    main := _numéro, / .

dégage(_numéro) .

```

```

poser ->
    _main := main,
    _main \== 0,
    nouvelle_pos(_main, sur, 1), / .

poser .

```

```

prendre(_numéro) ->
    main \== _numéro,
    poser,

```

```

nouvelle_pos(_numéro , dans , 2) ,
main := _numéro , / .
prendre(_numéro) .

ouvre(_numéro) ->
    état(_numéro , fermée , _prép , _objet , _pos) , / ,
    dégage(_numéro) ,
    dedans := _numéro ,
    poser ,
    dessine_couv(_numéro , ouverte) ,
    replace( état(_numéro , fermée , _pré , _obj , _po) , [] ,
            état(_numéro , ouverte , _pré , _obj , _po) , [] ) , / ,
    afficheLn(* j'ouvre', _numéro) , / .
ouvre(_numéro) .

```

```

ferme(_numéro) ->
    état(_numéro , ouverte , _prép , _objet , _pos) , / ,
    dégage(_numéro) ,
    dedans := _numéro ,
    poser ,
    dessine_couv(_numéro , fermée) ,
    replace( état(_numéro , ouverte , _pré , _obj , _po) , [] ,
            état(_numéro , fermée , _pré , _obj , _po) , [] ) , / ,
    afficheLn(* je ferme', _numéro) , / .
ferme(_numéro) .

```

```

vide(_numéro) ->
    état(_objet , _état , dans , _numéro , _pos) ,
    dégage(_objet) ,
    prendre(_objet) ,
    fail .
vide(_numéro) .

```

/* changement d'un objet de place */

```

nouvelle_pos(_numéro , _préposition , _objet) ->
    calcule_pos(_préposition , _objet , pos(_x , _y)) ,
    état(_numéro , _état , _sur , _obj , _oldpos) ,
    efface_pos(_numéro , _sur , _obj) ,
    replace( état(_numéro , _état , _sur , _obj , _oldpos) , [] ,
            état(_numéro , _état , _préposition , _objet , pos(_x , _y)) , [] ) ,
    vide_table(_oldpos) ,
    main := 0 ,
    dessine_pos(_numéro , _préposition , _objet) ,
    affiche(* je met' , _numéro) ,
    afficheLn(_préposition , _objet) , / .

```

/* fonctions d'affichage */

```

infos_tout(_numéro , _valétat , _préposition , _objet) ->
    objet(_numéro, _article, _taille , _forme, _couleur, _poids),
    tradétat(_forme , _valétat , _état) ,
    quel_surnom(_nom , _numéro) ,
    writewords(_nom, _article, _taille , _forme, _couleur, _état, est),
    afficheIn(_préposition , _objet) .

```

```

infos_réduit(_numéro) ->
    objet(_numéro, _article, _taille , _forme, _couleur, _poids),
    quel_surnom(_nom , _numéro) ,
    writewords(_nom, _article, _taille , _forme, _couleur) ,
    writeln .

```

```

tradétat(boite , _ouverte , _ouverte) -> / .
tradétat(_forme , _ouverte , ") .

```

```

informer(_numéro , tout) ->
    état(_numéro , _état , _préposition , _objet , _pos) , / ,
    infos_tout(_numéro , _état , _préposition , _objet) .
informer(_numéro , visible) ->
    état(_numéro , _état , _préposition , _objet , _pos) , / ,
    visible(_préposition , _objet) ,
    infos_tout(_numéro , _état , _préposition , _objet) .

```

```

affiche_tout(_visible) ->
    objet(_numéro , _article , _taille , _forme , _couleur , _poids) ,
    _numéro > 2 ,
    informer(_numéro , _visible) ,
    fail.

```

```

affiche(_prép , _numéro) ->
    quel_surnom(_nom , _numéro) ,
    objet(_numéro , _article , _taille , _forme , _couleur , _poids) ,
    writewords(_prép, _nom, _article, _taille, _forme, _couleur) .
afficheIn(_prép , _numéro) ->
    affiche(_prép , _numéro) , writeln .

```

```

quel_surnom(_nom , _numéro) -> surnom(_nom , _numéro) , / .
quel_surnom(" , _numéro) .

```

```

visible(sur , 1) -> / .          /* déterminer si l'objet est visible */
visible(dans , 2) -> / .
visible(sur , _objet) -> / ,
    état(_objet , _état , _préposition , _obj , _pos) , / ,
    visible(_préposition , _obj) , / .
visible(dans , _objet) ->
    état(_objet , ouverte , _préposition , _obj , _pos) , / ,
    visible(_préposition , _obj) , / .

```

/* fonctions de base des réponses aux questions */

estceque(_formule) ->
 _formule ,
 writeln(oui) , / .

estceque(_formule) -> writeln(non).

compte(_formule) ->
 nb ::= 0 ,
 formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
 _éta,_pré,_obj),
 nb ::= nb + 1 ,
 fail .
compte(_formule) .

combien(_formule) ->
 compte(_formule) ,
 _nb ::= nb , pronb ::= nb ,
 putln("il y en a " , _nb) .

quoi(sous , _formule) ->
 formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
 _éta,sur , _obj),
 infos_réduit(_obj) ,
 pronom ::= _obj ,
 fail .

quoi(autour , _formule) ->
 formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
 _éta,dans , _obj),
 infos_réduit(_obj) ,
 pronom ::= _obj ,
 fail .

quoi(_prép , _formule) ->
 formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
 _éta,_pré,_obj),
 état(_objet , _état , _prép , _numéro , _pos) ,
 infos_réduit(_objet) ,
 pronom ::= _objet ,
 fail .

où(_formule) ->
 formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
 _état,_prép,_objet),
 pronom ::= _numéro ,
 efface_objet(_numéro) , dessine_objet(_numéro) ,
 afficheIn(_prép , _objet) ,
 fail .

hsh_coded(quel(,)) .
quel(taille, _formule) ->

```

        formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
                    _état,_prép,_objet),

        writeln(_taille) ,
        pronom := _numéro ,
        fail .
quel(forme , _formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
                _état,_prép,_objet),

    writeln(_forme) ,
    pronom := _numéro ,
    fail .

quel(couleur , _formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
                _état,_prép,_objet),

    writeln(_couleur) ,
    pronom := _numéro ,
    fail .
quel(poids , _formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
                _état,_prép,_objet),

    _numéro > 2 ,
    writeln(_poids , ' ' , grammes) ,
    pronom := _numéro ,
    fail .
quel(poids , _rapport , _formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
                _état,_prép,_objet),

    _numéro > 2 ,
    block(titi) ,
        unité(poids , _u2 , _rapport) ,
        /(titi) ,
    _poi := _poids / _rapport ,
    writeln(_poi , ' ' , _u2 , '(s)) ,
    pronom := _numéro ,
    fail .
quel(volume , _formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
                _état,_prép,_objet),

    taille(_taille , _volume) ,
    writeln(_volume , ' ' , litre) ,
    pronom := _numéro ,
    fail .
quel(position , _formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
                _état,_prép,_objet),

    afficheln(_prép , _objet) ,
    pronom := _numéro ,
    fail .

```

```

quel(nom , _formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
               _état,_prép,_objet),

    pronom := _numéro ,
    quel_surnom(_nom , _numéro) ,
    writeln(_nom) ,
    fail .

quel(_formule) ->
    formule_gn(_formule,_numéro,_taille,_forme,_couleur,_poids,
               _état,_prép,_objet),

    pronom := _numéro ,
    infos_réduit(_numéro) .

hsh_coded(existe(_, _)) .
existe(comp(_comp , _nombre) , _formule) -> /* plus de Nb nom */
    compte(_formule) ,
    pronb := nb ,
    _nb := nb , verif_de(_comp , _nb , _nombre) ,
    nb > _nombre ,
    writeln(oui) , / .
existe(nombre(_nombre) , _formule) ->
    compte(_formule) ,
    pronb := nb ,
    nb == nombre ,
    writeln(oui) , / .

existe(_comp , _formule) -> /* plus de nom (sous entendu que de ...)*/
    compte(_formule) ,
    nb > pronb ,
    _nb := nb ,
    _pronb := pronb ,
    verif_de(_comp , _nb , _pronb) ,
    writeln(oui) , / .
existe(_specif , _formule) ->
    _nb := nb ,
    putln(non , ' il y en a ' , _nb) .

hsh_coded(existe(_, _, _)) .
existe(_comp , _nom1 , _nom2) -> /* plus, moins, autant de n1 que de n2 */
    compte(_nom1) ,
    pronb := nb ,
    compte(_nom2) ,
    _pronb := pronb ,
    _nb := nb ,
    verif_de(_comp , _pronb , _nb) ,
    writeln(oui) , / .
existe(_specif , _nom1 , _nom2) ->
    _nb := pronb ,

```

```
putln(non, ' il y en a ' , _nb) .
```

dessiner les objets

La fonction de ces sous programmes est de dessiner des objets de forme, de taille et de couleur spécifiés à un endroit précisé par les coordonnées (ligne et colonne) du milieu de la base de l'objet. La couleur "noir" correspond à un effacement.

cela fonctionnait sous turbo pascal, mais ne fonctionne plus sous delphi, les primitives clrscr, cursor, window, bgcolor, ... ayant été supprimées.

```
dessine_init ->
    clrscr ,
    ligtable := 12 , ligmain := 5 , colmain := 60 .
dessine_tout .
dessine_couv( _numéro , _état) .
efface_objet( _numéro) .
dessine_objet( _numéro) .
vide_table( _l , _c) .
calcule_pos( _prép , _obj , pos(ligmain-1 , colmain)) .
dessine_pos( _numéro , _préposition , _objet) .
efface_pos( _numéro , _prép , _objet) .

    /******
    /* calcul de la position d 'un objet */
    /******

calcule_pos(dans , 2 , pos(ligmain-1 , colmain)) -> / .
calcule_pos(sur , 1 , pos(ligtable-1 , _c)) -> / ,
    retract_one(place_libre(_p),[]) ,
    _c := _p * 8 .
calcule_pos(dans , _obj , pos(_l , _c)) -> / ,
    état( _obj , _ouverte , _prép , _ob , pos(_x , _c)) ,
    _l := _x - 1 .
calcule_pos(sur , _obj , pos(_l , _c)) ->
    état( _obj , _ouverte , _prép , _ob , pos(_x , _c)) ,
    objet( _obj , _art , _taille , _forme , _couleur , _poids) ,
    vtaille( _taille , _vl) ,
    _l := _x - _vl - 1 .

vide_table(pos(_l , _c)) ->      /* récupération d'une place sur la table */
    _l == ligtable -1 ,        /* si n.ligne = ligne_table - 1 */
    _p := _c / 8 ,
    assert(place_libre(_p) , [] ) , / .
vide_table(_pos).

    /* places libres au départ sur la table */
place_libre(5). place_libre(3). place_libre(7). place_libre(1).
```

```
place_libre(9). place_libre(4). place_libre(6). place_libre(2).
place_libre(8).
```

le programme robot - robot.pro

```
module(robot).          /* contient la description de l'état du monde */
define( objet(_,_,_,_)). /* les objets existants */
define( état(_,_,_,_)). /* la position des objets */
define( surnom(_,_)).    /* les noms des objets */

multi_hsh_coded( objet(_,_,_,_)). /* hsh code pour accélérer */
multi_hsh_coded( état(_,_,_,_)).
multi_hsh_coded( surnom(_,_)).
end_module.

main.
load('grammair.pro'). /* les règles de la grammaire française */
load('nombre.pro').   /* la grammaire des nombres et des quantités */
load('outil.pro').     /* le dictionnaire des mots outils du français */
load('dico.pro').      /* le dictionnaire du robot */
load('semantik.pro').  /* les contraintes sémantiques */
load('formule.pro').   /* la mise en formule des propositions */
load('action.pro').    /* les règles d'action du robot */
load('robodess.pro').  /* les dessins des objets */
```

exemple de manipulation

```
génère une petite sphère orange de 1 kg
      * je pose la sphère sur la table
génère A une grosse boîte rouge pesant 4 kilos
nomme D la sphère
génère B une petite boîte verte qui pèse 2 kilos 5
      * je pose A sur la table
génère C une minuscule sphère verte dont le poids est 2 kg
      * je pose B sur la table
mets la sur B
      * je met C sur B
génère E une minuscule boîte orange de 1 kilo
mets la petite boîte dans la grande
      * je pose E sur la table
      * j'ouvre A
      * je met C dans la main
      * je met C sur la table
      * je met B dans la main
      * je met B dans A
combien y a t il de boîtes qui sont dans la grande
      I
y a t il autant de pyramides
```

non
 y a t il plus de boites que de pyramides
oui
 mets l'objet pesant la moitié de A sur E
** je met C dans la main*
** je met C sur E*

 mets sur A l'objet dont la taille est petite et qui est sur la table
** je ferme A*
** je met D dans la main*
** je met D sur A*
 quelle est la couleur de l'objet qui est sur la boite de plus de 3 kg
orange
 combien y a t il de sphères qui sont sur une boite
2
 quelle est la couleur de celle qui est sur la boite plus petite que A
verte
 combien de kilos fait C
2 kilos
 combien pèse C
2000 grammes
 où est la boite pesant au moins le triple du poids de la sphère orange
sur la table
 quelle est sa taille
grosse
 quelle est la couleur de la boite qui a la même taille que la sphère
verte
 mets sur A l'objet vert qui pèse plus de deux fois plus que D
** je met D dans la main*
** je met D sur la table*
** j'ouvre A*
** je met B dans la main*
** je met B sur la table*
** je ferme A*
** je met B dans la main*
** je met B sur A*
 que supporte A
B la petite boite verte
 qui supporte B
A la grosse boite rouge
 quel est son poids
4000 grammes
 qui est supporté par A
B la petite boite verte
 combien pèse-t-elle
2500 grammes
 pose E sur B
 pose C sur E

pose la pile de B sur A
* *je prends C la minuscule sphère*
* *je pose C sur la table*
* *je prends E*
* *je pose E sur la table*
* *je prends B*
* *je pose B sur A*
* *je prends E*
* *je pose E sur B*
* *je prends C*
* *je pose C sur E*
donne D à l'opérateur

Analyse du langage naturel et déduction logique

1 La Logique

1.1 L'intelligence

A partir de quand peut-on dire qu'un animal fait preuve d'intelligence ?

Soit un rat placé dans une cage avec 3 portes peintes en rouge, vert et bleu. On dépose tous les matins un morceau de gruyère derrière la porte rouge, qui est la première à droite.

Au bout de quelques jours, le rat finit par trouver son gruyère du premier coup.

On déplace alors les portes et on place encore le gruyère derrière la porte rouge.

Au bout de quelques jours, le rat finit par trouver son gruyère du premier coup.

On déplace alors les portes tous les jours, et le rat a compris, il trouve du premier coup

Le rat a mémorisé la couleur de la porte intéressante : rouge.

On place un singe dans une pièce où il y a une banane pendue au plafond et une caisse contenant un manche à balai. Au bout de quelques minutes, le singe trouve le manche à balai et tape sur la banane, qui tombe.

Le singe a su utiliser quelque chose pour en faire un outil.

Enfin, que dirait-on d'un perroquet qui, après avoir entendu les 4 affirmations suivantes, répondrait correctement aux deux questions :

Tous les chats sont des félidés	
les félidés mangent des souris	
Grosminet est un chat	
Hector ne mange pas de souris	
Que mange Grosminet ?	des souris
Est-ce que Hector est un chat ?	non

On dirait : ce n'est pas croyable, ce perroquet raisonne, il est intelligent !

Et bien ce perroquet existe, il s'appelle Coco et il habite chez moi.

Le programme est une réécriture en *myprolog* puis en *gnu prolog* du programme « moslog » de la première partie, écrit en pascal, et capable de faire des déductions. La syntaxe a été étendue, en particulier pour comprendre les noms composés, des relatives complexes (comme dans le programme du robot) et des règles (si .. alors ...).

1.2 Les propositions

Une proposition peut être vraie ou fausse.

Elle peut ne s'appliquer qu'à un seul individu :

Grosminet est un chat

elle peut avoir un sens universel :

les chats sont des félidés

les chats aiment les souris

Une proposition dite universelle s'applique à plusieurs individus, et on peut la décomposer en deux ou plusieurs propositions non universelles, s'appliquant à des individus particuliers.

si « *X est un chat* » est vrai alors « *X est un félidé* » est vrai aussi

si « *X est un chat* » est vrai ET si « *Y est une souris* » est vrai, alors « *X aime Y* » est vrai

et on peut en déduire que

si « *X n'est pas un félidé* » est vrai alors « *X n'est pas un chat* » est vrai.

si « *X n'aime pas les souris* » est vrai alors « *X n'est pas un chat* » est vrai

Mais attention, cela ne veut pas dire que *si X aime les souris, alors X est un chat*

On n'a pas dit :

seuls les chats aiment les souris

d'ailleurs ce serait faux, certains chiens, dits ratiers, aiment les souris et les chassent, de même que les rapaces.

1.3 Le raisonnement par l'absurde et l'implication réciproque

En mathématiques, on utilise souvent le raisonnement par l'absurde, qui considère deux propositions P et Q. Supposons que :

P implique Q (qui signifie que si P est vrai, alors Q est vrai)

Si Q est faux, P ne peut être vrai sinon Q serait vrai, donc P est faux.

Autrement dit

Non Q implique non P (Si Q est faux, alors P est faux)

Cela ne veut pas dire que

Q implique P

On n'a pas dit

P implique Q et réciproquement

On n'a pas dit non plus

si et seulement si P est vrai alors Q est vrai

La réciprocité doit être explicitement affirmée, par une deuxième affirmation :

les chiens aboient (si X est un chien, alors X aboie)

ceux qui aboient sont des chiens (si X aboie alors X est un chien)

Ou en une seule phrase par :

seuls les chiens aboient

1.4 Les transformations de proposition

Plusieurs propositions peuvent avoir exactement le même sens, et des règles de transformation simples permettent de transformer une forme en l'autre.

Par exemple, la proposition universelle :

les chats aiment les souris

a le même sens que :

tous les chats aiment les souris (il suffit de rajouter *tous* devant)

ou que :

les souris sont aimées par les chats (il suffit d'inverser sujet et complément, de mettre le verbe au passif et d'introduire l'ancien sujet par *par*)

De même :

les vaches n'aiment pas les souris

a le même sens que :

aucune vache n'aime les souris (en remplaçant *les* par *aucune*, en supprimant *pas* et en passant le sujet et le verbe au singulier)

et a le sens contraire de :

les vaches aiment les souris (en retirant *ne ... pas*)

de même, toutes les éléments suivants ont le même sens :

<i>une souris petite</i>	<i>une petite souris</i>
<i>une souris dont la taille est petite</i>	<i>une souris qui est petite</i>
<i>une souris qui a la taille petite</i>	<i>une souris qui a une petite taille</i>
<i>une souris de petite taille</i>	<i>une souris à la taille petite</i>
<i>une souris à taille petite</i>	<i>une souris à petite taille</i>
<i>une souris ayant une petite taille</i>	<i>une souris ayant la taille petite</i>

et se transforment tous par des règles simples en

une souris petite

la *taille*, la *couleur*, sont reconnus comme des attributs ayant une liste de valeurs possibles, ou une valeur numérique comparable à d'autres de même type.

la même transformation, considérant « poil » comme un attribut, transforme

une souris dont les poils sont longs

....

en *une souris qui a les poils longs*

et

une souris mangeant ...

en *une souris qui mange*

1.5 Les syllogismes

Le syllogisme est un raisonnement composé de deux propositions prémisses, dont l'une au moins est universelle, et desquels on déduit une conclusion. Bien avant le maître Lewis Carroll (auteur de *Alice au pays des merveilles*, et de *la logique sans peine*), Aristote avait montré quatre figures de syllogisme :

Tous les chats sont des félidés
tous les félidés ont des griffes
donc tous les chats ont des griffes

Tous les chats sont des félidés
les félidés ne sont pas des canidés
donc les chats ne sont pas des canidés

Hector est un chat
tous les chats mangent des souris
donc Hector mange des souris

Hector est un chat
les chats ne sont pas des canidés
donc Hector n'est pas un canidé

On se propose d'analyser par un programme informatique des propositions simples de ce genre et d'effectuer des raisonnements logiques basés sur ces mécanismes afin de répondre à des questions à ce sujet.

Il s'agit d'une réécriture en *myprolog* puis finalement en *gnu prolog* du programme « moslog », écrit en pascal et présenté dans la première partie. La syntaxe de *gnu prolog* est légèrement différente de celle de *myprolog*.

Différences de syntaxe entre *myprolog* et *gnu prolog*

L'opérateur `->` devient `:-`

Le `cut` / devient `!`

Une liste `[a,b, !c]` devient `[a,b |c]`

Si un fait peut être ajouté par « `assert` », tous les faits de ce prédicat doivent être ajoutés par `assert` et pas par « `fait(...)`. »

Et évidemment, les prédicats prédéfinis n'ont pas toujours les mêmes noms

2 Rappel sur le langage Prolog

2.1 Mode d'enregistrement des affirmations en prolog

Prolog enregistre des affirmations d'implication du genre

$P \Rightarrow Q$ équivalente à Q ou $\text{non}(P)$

Q ou $\text{non}(Q)$ est vrai, et comme $\text{non}(Q)$ implique $\text{non}(P)$, Q ou $\text{non}(P)$ est vrai

$P \text{ et } Q \text{ et } R \Rightarrow S$ équivalente à S ou $\text{non}(P)$ ou $\text{non}(Q)$ ou $\text{non}(R)$

Sous la forme

$Q :- P.$

$S :- P, Q, R.$

Le signe « $:-$ » est à lire comme « *si* » et non comme « implique », c'est l'inverse !

La partie droite de la règle est une condition, qui peut être multiple, et la règle s'applique si l'ensemble des conditions est vraie.

Une règle sans condition, est un fait, réputé toujours vrai, que prolog note :

$P.$

2.2 Propositions spécifiques et propositions universelles

Notre programme va analyser des phrases de plus en plus complexes, va enregistrer ces informations, et va être capable de répondre à des questions faisant appel à des déductions. Les deux parties : analyse, enregistrement et réponse sont facilement séparables, on peut ne s'intéresser qu'à la partie « analyse » et remplacer tout ce qui concerne l'enregistrement et la réponse, pour réaliser une autre application, comme on le fera par la suite.

Considérons d'abord les propositions simples entrant dans la composition des syllogismes, et effectuons quelques contrôles grammaticaux et transformations :

Dans cette étude on enregistrera une proposition non universelle sous la forme d'un fait

Prolog, ainsi : *Hector est un grec*

s'enregistre en : affirme (Hector, être, grec, vrai) .

et *Hector n'est pas un chat*

en : affirme (Hector, être, chat, faux) .

Une proposition universelle est enregistrée ici sous la forme d'une règle Prolog :

les grecs sont des menteurs

est traduit en *si X est grec alors X est menteur*

soit la règle prolog $\text{affirme}(_x, \text{être}, \text{menteur}, \text{vrai}) :- \text{affirme}(_x, \text{être}, \text{grec}, \text{vrai}).$

Ce qui fait apparaître une variable $_x$, la règle s'appliquant pour tout x ; par la suite cette variable x sera remplacée par un individu particulier qui satisfait la condition de la règle.

Grâce au bon vieux mécanisme du raisonnement par l'absurde, on sait que notre affirmation, semblable à $P \Rightarrow Q$, est équivalente à $\text{non}(Q) \Rightarrow \text{non}(P)$, c'est-à-dire à

si X n'est pas menteur alors X n'est pas grec

laquelle s'enregistre sous la forme :

$\text{affirme}(_x, \text{être}, \text{grec}, \text{faux}) :- \text{affirme}(_x, \text{être}, \text{menteur}, \text{faux}).$

Avec le verbe avoir, on obtient de même

un oiseau a des ailes

si X est un oiseau alors X a des ailes

affirme(_x, avoir, aile, vrai) :- affirme(_x, être, oiseau, vrai).

et l'équivalent négatif :

affirme(_x, être, oiseau, faux) :- affirme(_x, avoir, aile, faux).

Avec un autre verbe que les verbes être et avoir, la traduction fait apparaître une deuxième variable :

les chats aiment les souris

se traduit en

si X est un chat et si Y est une souris alors X aime Y

et s'enregistre en

affirme(_x, aimer, _y, vrai) :-

affirme(_x, être, chat, vrai), affirme(_y, être, souris, vrai).

avec donc deux conditions concernant les deux entités mises en relation par le verbe.

Cette affirmation a deux conditions se schématise en $P \text{ et } Q \Rightarrow R$

et est équivalente à $R \text{ ou } \text{non}(P \text{ et } Q)$

Soit $R \text{ ou } \text{non}(P) \text{ ou } \text{non}(Q)$ - où l'on remarque qu'un seul élément n'est pas nié, et c'est R ce que l'on peut permuter en

$\text{non}(P) \text{ ou } \text{non}(Q) \text{ ou } R$

et, si on note np pour non(P) et nr pour non(R)

$\text{np ou non}(Q) \text{ ou non}(nr)$

où l'on remarque, en penchant un peu la tête, qu'un seul élément, np, n'est pas nié, et donc on y reconnaît l'équivalent de :

$Q \text{ et } nr \Rightarrow np$

Soit $Q \text{ et } \text{non}(R) \Rightarrow \text{non}(P)$, le 1^{er} équivalent négatif

Et, par la dernière permutation possible, on obtient le deuxième équivalent négatif :

$P \text{ et } \text{non}(R) \Rightarrow \text{non}(Q)$

Ce qui, revenant à nos chats et à nos souris, nous donne le premier équivalent négatif :

celui qui n'aime pas les souris n'est pas un chat

et donc la deuxième règle :

affirme(_x, être, chat, faux) :-

affirme(_y, être, souris, vrai), affirme(_x, aimer, _y, faux).

Et le deuxième équivalent négatif :

celui qui n'est pas aimé par les chats n'est pas une souris

affirme(_y, être, souris, faux) :-

affirme(_x, être, chat, vrai), affirme(_x, aimer, _y, faux).

Enregistrer les équivalents négatifs permettra à notre programme d'effectuer facilement des raisonnements en chainage arrière, alors que le mécanisme interne de prolog réalise déjà le chainage avant.

L'équivalent négatif n'est pas la réciproque, qui serait :

celui qui aime les souris est un chat

La réciproque n'est pas toujours vraie, par contre l'équivalent négatif est toujours vrai.

2.3 Mécanisme interne du raisonnement de Prolog

A la question « *qui est un chat* », le mécanisme interne de Prolog recherche tous les faits et règles du genre :

affirme(_x, être, chat, vrai)

et à la question « *qui n'est pas chat* », le mécanisme recherche les faits et règles du type

affirme(_x, être, chat, faux)

Au cours de ces opérations, Prolog est amené à comparer des conditions à toutes les têtes de règles et faits. Les unes et/ou les autres contiennent ou non des variables, et la comparaison réussit si tout ce qui n'est pas variable est identique. Si une variable se compare à une variable, elle reste variable, et si une variable se compare à un mot, elle prend la valeur de ce mot, afin que la condition soit vérifiée. Si la comparaison réussit, le fait est vérifié, et si c'est une règle, il faut encore en vérifier toutes les conditions avec les nouvelles valeurs des variables.

Le programme trouve ou non des faits conformes ou des règles conformes démontrables, et il répond respectivement *oui* ou *non*, sinon il répond *je ne sais pas*, comme quoi en informatique, il n'y a pas que *oui* et *non* !

Supposons que l'on ait seulement affirmé :

affirme(Grosminet, être, chat, vrai).

affirme(Hector, être, chat, faux).

affirme(Titine, être, souris, vrai).

affirme(_x, aimer, _y, vrai) :- (les chats aiment les souris)

affirme(_x, être, chat, vrai),

affirme(_y, être, souris, vrai).

La question *qui est un chat* se traduit en

affirme(_qui, être, chat, vrai) ?

Prolog compare cela aux 4 têtes de règles, dont les 3 faits.

Au 1^{er} fait, _qui prend la valeur Grosminet et la comparaison réussit, une solution a été trouvée, Prolog affiche les valeurs de toutes les variables de la question.

Au 2^{ème} fait, _qui prend la valeur Hector, mais la comparaison de faux et vrai échoue.

Au 3^{ème} fait, _qui prend la valeur Titine, mais la comparaison de chat à souris échoue.

A la règle, _qui se compare avec succès à _x, mais la comparaison de être à aimer échoue.

Et donc Prolog répond simplement :

Grosminet

La question *qui aime Titine* se traduit en

affirme(_qui, aimer, Titine, vrai) ?

seule la règle 4 se compare avec succès, _x se comparant à _qui, et _y à Titine, et prolog cherche alors à voir si la 1^{ère} condition affirme(_qui, être, chat, vrai) est vrai, puis si la deuxième : affirme(_y, être, souris, vrai), avec _y = Titine, l'est aussi, ce qui est le cas.

Pour que affirme(_qui, aimer, souris, vrai) ? réponde non seulement *Grosminet* mais aussi *chat*, il faut ajouter au programme le fait : affirme(_x, être, _x, vrai).

Si une affirmation réciproque a été enregistrée, un problème technique se pose :

seuls les chiens aboient

s'enregistre comme une affirmation non réciproque en :

1 affirme(_x, aboyer, vrai) :- affirme(_x, être, chien, vrai).

2 affirme(_x, être, chien, faux) :- affirme(_x, aboyer, faux).

auquel on ajoute la réciproque :

3 affirme(_x, être, chien, vrai) :- affirme(_x, aboie, vrai).

4 affirme(_x, aboyer, faux) :- affirme(_x, être, chien, faux).

Lorsque Prolog cherche à savoir si *X aboie (1)*, il est amené à vérifier si *X est un chien (3)*

Or pour vérifier si *X est un chien (3)*, Prolog doit chercher à vérifier si *X aboie (1)*

Donc, cela va boucler indéfiniment, à moins qu'il y ait un mécanisme anti boucle efficace dans l'interpréteur Prolog.

3 1^{ère} méthode d'analyse des phrases

3.1 Analyse grammaticale

3.1.1 Séparation des mots

Déclarons qu'une phrase est soit une affirmation, soit une question. Pour cela, nous définissons deux règles :

```
phrase(_phrase) :- affirmation (_phrase).  
phrase(_phrase) :- question(_phrase).
```

On note « _phrase » une variable contenant la phrase à analyser, cette variable contient donc une liste de mots, par exemple : [les, chats, mangent, les, souris]. _phrase est une phrase si la condition *affirmation* est remplie OU si la condition *question* est remplie.

Déclarons qu'une phrase de type proposition universelle est une affirmation. Une telle phrase est composée d'un article suivi d'un nom, d'un verbe, d'un autre article et d'un autre nom, chacun de ces mots de la variable _phrase étant lui-même rangé dans une variable :

```
affirmation ([_article1, _nom1, _verbe, _article2, _nom2]) :-  
    article(_article1),  
    nom(_nom1),  
    verbe(_verbe),  
    article(_article2),  
    nom(_nom2),  
    writeln('Règle à enregistrer : ', _nom1, _verbe, _nom2).
```

Nous avons ici une règle à plusieurs conditions, qui doivent toutes être vraies pour que la règle soit vraie : notre phrase est une affirmation si son 1^{er} mot _article1 remplit la condition article(_article1), ET son second mot _nom1 la condition nom(_nom1), ET ...

Et maintenant déclarons quelques faits représentant notre vocabulaire :

article(le).	nom(chat).	verbe(mange).
article(les).	nom(chats).	verbe(mangent).
article(la).	nom(souris).	verbe(est).

Avec ces simples déclarations, notre programme est déjà capable d'analyser les phrases

[le,chat,mange,la,souris]

[les,chats,mangent,les,souris]

Mais aussi

[les,chat,mange,le,souris]

contenant des fautes d'accords

[la,souris,mange,le,chat]

dont la forme est correcte, mais le sens notoirement faux

Nous utiliserons le petit programme suivant, qui lit des mots, et les traduit en liste de mots puis déclenche l'analyse :

```
read_phrase(['.' | Phrase_in], Phrase_out) :- reverse(['.' | Phrase_in], Phrase_out), !.
read_phrase(Phrase_in, Phrase_out) :-
    read_token(Atom),
    read_phrase([Atom | Phrase_in], Phrase_out).

phrase :-
    repeat,
        read_phrase([], _phrase),
        phrase(_phrase).
```

Read_phrase est un peu complexe, son rôle est de lire un mot (un token), de l'ajouter en tête de la liste des mots, de recommencer jusqu'au point final de la phrase, puis de retourner la liste. Contentons nous de constater qu'il rend une liste de mots terminés par un point.

Nous lancerons donc notre programme par la commande
phrase.

qui se met à lire indéfiniment des phrases, grâce à la procédure magique read_phrase, et à les analyser.

3.1.2 Contrôle des accords grammaticaux

Nous allons à présent vérifier le genre et le nombre de chaque élément de la phrase

Les chats aiment les souris.

```
affirmation([_article1, _nom1, _verbe, _article2, _nom2]) :-
    article(_article1, _genre1, _nombre1),
    nom(_nom1, _genre1, _nombre1, _racine_nom1),
    verbe(_verbe, _nombre1, _racine_verbe),
    article(_article2, _genre2, _nombre2),
    nom(_nom2, _genre2, _nombre2, _racine_nom2),
    ajoute_règle(_racine_nom1, _racine_verbe, _racine_nom2, vrai).
```

article(le, masculin, singulier).	nom(chat, masculin, singulier, chat).	verbe(a, singulier, avoir).
article(les, _any, pluriel).	nom(chats, masculin, pluriel, chat).	verbe(ont, pluriel, avoir).
article(la, féminin, singulier).	nom(souris, féminin, _any, souris).	verbe(est, singulier, être).
verbe(mange, singulier, manger).	verbe(mangent, pluriel, manger).	verbe(sont, pluriel, être).

On remarque que :

_article1 et _nom1 doivent avoir même genre et nombre : _genre1 et _nombre1

La règle article(_article1,... recherche tous les articles et rend le genre et le nombre correspondant à l'_article1 reconnu : « les ».

_nom1 et _verbe doivent avoir le même nombre, _nombre1

La règle nom(_nom1,... recherche tous les noms de même genre et nombre que _article1 et correspondant à _nom1 (« chats ») et rend la racine de ce nom (« chat »).

3.2 Enregistrement des faits et règles

3.2.1 Les règles pour les phrases universelles

le but est d'enregistrer la règle correspondant à :

racine_nom1 verbe_infinitif racine_nom2

Soit donc, pour la phrase « *un chat est un félin* »

affirme(_x, être, félin, vrai) :- affirme(_x, être, chat, vrai).

ainsi que l'équivalent négatif :

affirme(_x, être, chat, faux) :-affirme(_x, être, félin, faux).

ajoute_règle(_racine_nom1, être, _racine_nom2, vrai) :-

assertz((affirme(_x, être, _racine_nom2,vrai) :-affirme(_x, être, _racine_nom1,vrai))),

assertz((affirme(_x, être, _racine_nom1,faux) :-affirme(affirme(_x, être, _racine_nom2,faux))).

Remarquons que cette règle « ajoute_règle » ne gère que le cas du verbe être à la forme affirmative. Il faut la décliner plusieurs fois pour gérer le verbe avoir et les autres verbes et la forme négative. Dans le cas d'un verbe différent de *être* et *avoir*, il apparaît une deuxième variable _y représentant le complément du verbe, et deux équivalents négatifs.

Pour la phrase « *les chats mangent les souris* » on obtient :

affirme(_x, manger, _y, vrai) :-

affirme(_x, être, chat, vrai) ,

affirme(_y, être, souris, vrai).

affirme(_x, être, chat, faux) :-

affirme(_y, être, souris, vrai),

affirme(_x, manger, _y, faux).

affirme(_x, être, souris, faux) :-

affirme(_y, être, chat, vrai),

affirme(_x, manger, _y, faux).

ajoute_règle(_racnom1, _racverbe, _racnom2, vrai) :-

nonêtreavoir(_racverbe),

assertz((affirme(_qqun, _racverbe, _y, vrai) :-

affirme(_qqun, être, _racnom1, vrai), affirme(_y, être, _racnom2, vrai))),

assertz((affirme(_qqun, être, _racnom1, faux) :-

affirme(_qqun, _racverbe, _y, faux), affirme(_y, être, _racnom2, vrai))),

assertz((affirme(_qqun, être, _racnom2, faux) :-

affirme(_qqun, _racverbe, _y, faux), affirme(_y, être, _racnom1, vrai))).

nonêtreavoir(être) :- !, fail.

nonêtreavoir(avoir) :- !, fail.

nonêtreavoir(_x).

3.2.2 Les faits pour les phrases spécifiques

Une phrase de type « simple fait » s'applique à un individu précis, représenté par son prénom

Grosminet est un chat

et s'analyse de la même manière :

```
affirmation(_prénom, _verbe, _article, _nom) :-  
    prénom(_prénom, _genre),  
    verbe(_verbe, singulier, _racine_verbe),  
    article(_article, _genre, _nombre),  
    nom(_nom, _genre, _nombre, _racine_nom),  
    ajoute_fait(_prénom, _racine_verbe, _racine_nom, vrai).
```

```
prénom(Grosminet, masculin).
```

et pour notre phrase, le programme enregistre le fait :

```
affirme(Grosminet, être, chat, vrai).
```

Pour éviter d'enregistrer des doublons ou des incohérences, il est intéressant de contrôler si on savait ce fait, ou même si on savait son contraire.

```
ajoute_fait(_prénom, _racverbe, _racnom, vrai) :-  
    affirme(_prénom, _racverbe, _racnom, vrai),  
    writeln('Je le savais déjà'), !, fail.  
ajoute_fait(_prénom, _racverbe, _racnom, vrai) :-  
    retourne(_vrai, _faux),  
    affirme(_prénom, _racverbe, _racnom, faux),  
    writeln('C'' est incohérent avec ce que tu m''as déjà dit'), !, fail.  
ajoute_fait(_prénom, _racverbe, _racnom, vrai) :-  
    assertz(affirme(_prénom, _racverbe, _racnom, vrai)),  
    writeln(Enregistré).  
  
retourne(vrai, faux).  
retourne(faux, vrai).
```

Le fonctionnement de « ajoute_fait » nécessite un gris-gris particulier : les 3 règles « ajoute_fait » sont applicables, et des que l'une va à son terme, il ne faut plus essayer les autres, surtout pas la dernière, sinon on enregistrerait toujours et on dirait « Enregistré », même après avoir constaté que c'était inutile par l'une des 2 premières règles. C'est le sens du « ! » placé en fin de ces règles, qui signale à Prolog la fin des règles « ajoute_fait » en cours. « fail » porte bien son nom, signifiant *échec* : la condition n'est pas remplie.

Le même genre de contrôle doit être fait pour les règles comme pour les faits, pour empêcher de déclarer deux fois la même règle, ou une règle incohérente avec les précédentes.

Dans la suite des opérations nous ne distinguerons pas les faits des règles et

Grominet est un chat

s'enregistrera sous forme de la règle :

affirme(_x,être,chat,vrai) :- affirme(_x,être,Grominet,vrai).

Il est alors nécessaire d'ajouter au programme la règle :

affirme(_x,être,_x,vrai).

3.3 Fonctionnement de la réponse aux questions

Les questions s'analysent grammaticalement comme les affirmations. Attention : « *que mangent les chats* » doit s'analyser, en inversant sujet et complément, c'est-à-dire en :

affirme(_racnom, _racverbe, _x, vrai).

Les questions sont de deux types :

- 1- Question de type « *qui verbe gn* », « *que verbe gn* », avec plusieurs réponses possibles.

La question s'analyse sous la forme

affirme(_racnom, _racverbe, _x, vrai) que verbe gn (gn verbe quoi)

affirme(_x, _racverbe, _racnom, vrai) Qui verbe gn

ou affirme(_racnom, _x, _y, vrai) Gn fait quoi

et Prolog répond par la liste de tous les _x déclarés dans les faits ou règles enregistrés

question([qui, _verbe, _article, _nom]) :-

verbe(_verbe, singulier, _racverbe),

article(_article, _genre, _nombre),

nom(_nom, _genre, _nombre, _racnom),

writeln('Je recherche les réponses :'),

affirme(_x, _racverbe, _racnom, vrai),

writeln(_x).

- 2- Question de type « *est ce que ...* », avec une réponse en *oui* ou *non*

La question s'analyse sous la forme

répondre(_prénom, _racverbe, _racnom, vrai)

Il est important de voir que ce n'est pas parcequ'une affirmation n'est pas déclarée qu'elle est fausse. On recherche les faits ou règles amenant à répondre « *oui* », puis ceux amenant à répondre « *non* », et si on ne trouve ni les uns ni les autres, on répond « *Je ne sais pas* », comme quoi il n'y a pas que « *oui* » et « *non* » en informatique !

répondre(_prénom, _racverbe, _racnom, vrai) :-

affirme(_prénom, _racverbe, _racnom, vrai),

writeln(oui), !.

répondre(_prénom, _racverbe, _racnom, vrai) :-

retourne(_vrai, _faux),

affirme(_prénom, _racverbe, _racnom, faux),

writeln(non), !.

répondre(_prénom, _racverbe, _racnom, vrai) :-

writeln('Je ne sais pas').

3.4 Premiers résultats

3.4.1 Grammaire analysée

3.4.1.1 Affirmations comprises par le système

Gns Verbe Gnc	Tous Article Nom Verbe Gnc
Gns ne Verbe pas Gnc	Aucun Nom ne Verbe Gnc
Gns Verbe	Seuls Gns Verbe Gnc
Gns ne Verbe pas	Seuls Gns ne Verbe pas Gnc
Gnc est Participe-passé par Gns	Gnc n est pas Participe-passé par Gns
Ceux qui Verbe1 Gnc1 Verbe2 gnc2	Ceux qui ne Verbe1 pas Gnc1 Verbe2 gnc2

un « *groupe nominal* », peut prendre la place d'un sujet (Gns) ou d'un complément d'objet (Gnc). La seule différence entre un Gns et un Gnc est qu'un Gnc peut être vide, en particulier dans le cas d'un verbe intransitif.

Gn= prénom/article nom/adjectif/

3.4.1.2 Questions comprises par le système

Qui Verbe Gnc Que Verbe Gns

Verbe=Forme-Verbe/ne forme-Verbe pas
Forme-Verbe = est/sont/a/ont/mange/mangent

3.4.1.3 Vocabulaire initial

Participe-passé=mangé/mangée/mangées/mangés	Prénom=Pierre/Grosminet
Nom=chat/souris/griffe/félicé/ ...	Article=le/la/les/un/une/des

A tout moment, on peut faire apparaître la liste de tous les faits et règles enregistrés :
listing(affirme) .

un module permet d'ajouter des noms et des verbes au dictionnaire.

3.4.2 Exemples

On arrive ainsi à faire savoir au système que :

Les félidés mangent les souris	affirme(_x, manger, _y, vrai) :- affirme(_x, être, félidé, vrai), affirme(_y, être, souris, vrai).
Les félidés ont des griffes	affirme(_x, avoir, griffe, vrai) :- affirme(_x, être, félidé, vrai).
Grosminet est un chat	affirme(_x, être, chat, vrai) :-affirme(_x, être, Grosminet).
Les chats sont des félidés	affirme(_x, être, félidé, vrai):- affirme(_x, être, chat, vrai).
Hector ne mange pas les souris	affirme(_x, manger, _y, faux) :- affirme(_x, être, Hector, vrai), affirme(_y, être, souris, vrai).
Aucun chien ne mange des souris	affirme(_x, manger, _y, faux) :- affirme(_x, être, chien, vrai), affirme(_y, être, souris, vrai),
Seuls les chiens aboient	affirme(_x, aboyer, vrai):-affirme(_x, être, chien, vrai). affirme(_x, être, chien, vrai):-affirme(_x, aboyer, vrai).
Celui qui mange des souris est carnivore	affirme(_x, être, carnivore, vrai):- affirme(_x, manger, _y, vrai), affirme(_y, être, souris, vrai).

On peut répondre aux questions suivantes :

Que mangent les chats	affirme(chat, manger, _x, vrai) ?	souris
Qui mange les souris	affirme(_x, manger, souris, vrai) ?	félidé, chat, Grosminet
Est ce que Hector est un chat	affirme(Hector, être, chat, vrai) ?	non
Que mange Grosminet	affirme(Grosminet, manger, _x, vrai) ?	souris
Qu est Grosminet	affirme(Grosminet, être, _x, vrai) ?	chat, félidé
Qu a Grosminet	affirme(Grosminet, avoir, _x, vrai) ?	griffe
Que ne mange pas Hector	affirme(Hector, manger, _x, faux)	souris

Quelle va être la réponse à :

Est-ce que Hector a des griffes	affirme(Hector, avoir, griffe, vrai) ?
---------------------------------	--

Hector ne mange pas les souris, donc il n'est pas un félidé, donc il n'est pas un chat, mais il a peut être des griffes, les félidés ne sont pas les seuls à avoir des griffes, les chiens et les aigles en ont aussi, donc on ne peut pas répondre avec certitude par oui ou non. Et la proposition disant que les chiens ne mangent pas les souris n'apporte rien concernant les éventuelles griffes d'Hector, la bonne réponse est « *Je ne sais pas* » .

Hector est sans doute un chien, mais rien ne le dit, ce peut aussi bien être ce bon vieux roi Hector.

4 2^{ème} méthode d'analyse : le groupe nominal

4.1 Amélioration de l'analyse des phrases

4.1.1 Le groupe nominal

La méthode décrite jusqu'ici fonctionne parfaitement, mais elle a quelques défauts :

- Elle amène à écrire une règle « affirmation » pour chaque type de phrase

article nom verbe article nom

article nom verbe prénom

prénom verbe article nom

prénom verbe prénom

et les mêmes avec la forme négative : ne verbe pas

et les mêmes sans complément au verbe

Toutes ces règles diffèreraient peu les unes des autres, car on serait amené à y réaliser les mêmes analyses d'un « article nom », d'un prénom, d'un verbe et les mêmes contrôles de genre et nombre associés.

Le programme en résultant est donc assez verbeux, et en grande partie répétitif. Pour éviter cela, on introduit la notion de « groupe nominal », objet grammatical pouvant prendre la place d'un sujet ou d'un complément : *le chat* ou *Grosminet*.

On utilise le fait qu'en Prolog on peut représenter une liste par son ou ses premiers mots et le reste de la liste des mots, séparés par un « | » : [_mot1, _mot2 | _reste]

On utilise également les subtilités du « ! » et du « fail ». Le programme devient étonnamment court, et il reste très compréhensible, quoique son écriture demande une certaine habitude de la programmation en Prolog.

Le groupe nominal et la forme verbale s'analysent par des règles à trois paramètres :

- La bribe restante de phrase à analyser, commençant par ce qui est supposé être un groupe nominal
- La phrase restante, lorsqu'on en a retiré ce groupe nominal
- Le 3^{ème} paramètre est le résultat de l'analyse, réduit pour l'instant à une structure contenant la racine du nom, son genre et son nombre

```
groupe_nominal_sujet([_article, _nom | _reste], _reste, gn(_racine, _genre, _nombre)):-
```

```
    article(_article, _genre, _nombre),
```

```
    nom(_nom, _genre, _nombre, _racine).
```

```
groupe_nominal_sujet([_prénom | _reste], _reste, gn(_prénom, _genre, singulier)):-
```

```
    prénom(_prénom, _genre).
```

Profitons en pour gérer les adjectifs compléments du verbe être

un collier est cher

un chien est domestique

et aussi le cas de *non* dans :

un chien de couleur grise est non rare

```
groupe_nominal_sujet([_adj |_reste], _reste, gn(_rac_adj, _genre, _nombre)) :-  
    adjectif(_adj, _genre, _nombre, _rac_adj).
```

```
groupe_nominal_sujet([non, _adj |_reste], _reste, gn(non(_rac_adj), _genre, _nombre)) :-  
    adjectif(_adj, _genre, _nombre, _rac_adj).
```

```
adjectif(rare, _genre, singulier, rare).
```

```
adjectif(carnivore, _genre, singulier, carnivore).
```

Pour les verbes intransitifs, le complément doit pouvoir être absent, donc :

```
groupe_nominal_compl(_phrase, _reste, gn) :-  
    groupe_nominal_sujet(_phrase, _reste, gn).  
groupe_nominal_compl(_reste, _reste, gn(_vide, _genre, _nombre)).
```

4.1.2 Les formes verbales

On fait le même genre d'analyse pour reconnaître les formes active, affirmative et négative ainsi que les formes passives :

```
forme_verbale_active([_verb |_reste], _reste, verbe(_racverbe, vrai, _nombre)) :-  
    verbe(_verb, _nombre, _racverbe).  
forme_verbale_active([_ne, _verb, pas |_res], _res, verbe(_racverb, faux, _nombre)) :-  
    ne(_ne),  
    verbe(_verb, _nombre, _racverb).  
forme_verbale_passive([_est, _ppassé, par |_reste], _reste,  
    verbe(_racverbe, vrai, _genre, _nombre)) :-  
    verbe(_est, _nombre, être),  
    ppassé(_ppassé, _genre, _nombre, _racverbe).  
forme_verbale_passive([_ne, _est, pas, _ppassé, par |_reste], _reste,  
    verbe(_racverbe, faux, _genre, _nombre)) :-  
    ne(_ne),  
    verbe(_est, _nombre, être),  
    ppassé(_ppassé, _genre, _nombre, _racverbe).
```

```
ne(ne).
```

```
ne(n).
```

```
ppassé(mangé, masculin, singulier, manger).
```

4.2 Analyse de la phrase à l'aide du groupe nominal

Toutes les règles « *affirmation* » se résument alors à deux règles :

`affirmation(_phrase) :-`

```
    groupe_nominal_sujet(_phrase, _reste, gn(_racnom, _genre, _nombre)),
    forme_verbale_active(_reste, _rest, verbe(_racverbe, _vrai, _nombre)),
    groupe_nominal_compl(_rest, ['.'], gn(_racnom2, _genre2, _nombre2)),
    ajoute_règle(_racnom, _racverbe, _racnom2, _vrai).
```

et une règle analogue pour la forme passive

l'appel du deuxième groupe nominal vérifie que la phrase est complètement analysée, et donc que le reste est la liste constituée du seul point final : ['.']

4.3 Les questions

Les questions sont traitées par des analyses dérivées de celle de la phrase affirmative :

`phrase([est, ce, que | _phrase]) :-`

```
    groupe_nominal_sujet(_phrase, _reste, gn(_type, _racnom, _genre, _nombre)),
    forme_verbale_active(_reste, _rest, verbe(_typev, _racverbe, _vrai, _nombre)),
    groupe_nominal_compl(_rest, ['.'], gn(_type2, _racnom2, _genre2, _nombre2)),
    répondre(_racnom, _racverbe, _racnom2, _vrai).
```

`phrase([que | _reste]) :-`

```
    forme_verbale_active(_reste, _rest, verbe(_typev, _racverbe, _vrai, singulier)),
    groupe_nominal_sujet(_rest, ['.'], gn(_type2, _racnom2, _genre2, _nombre2)),
    répondre(_racnom2, _racverbe, _que, _vrai).
```

`phrase([qui | _reste]) :-`

```
    forme_verbale_active(_reste, _rest, verbe(_typev, _racverbe, _vrai, singulier)),
    groupe_nominal_sujet(_rest, ['.'], gn(_type2, _racnom2, _genre2, _nombre2)),
    répondre(_qui, _racverbe, _racnom2, _vrai).
```

`répondre(_x, _verbe, _y, _vrai) :-`

```
    affirme(_x, _verbe, _y, _vrai),
    writeln(_x, _verbe, _y, _vrai).
```

Il peut y avoir plusieurs réponses identiques. En effet, on peut affirmer un fait, qui s'enregistre, puis une règle qui déduit ce fait :

Grosminet mange les souris

puis *les chats mangent les souris*

Prolog trouvera que *Grosminet mange* par deux raisons différentes. Pour éliminer ces réponses doubles il faut, dans « répondre », gérer une liste des réponses déjà trouvées.

4.4 Génération de texte

Dans un langage classique, les sous programmes ont des paramètres et certains de ces paramètres sont des paramètres fournis au sous programme, lequel calcule un résultat dans les paramètres de sortie. En prolog, la notion de paramètre d'entrée ou de paramètre de sortie n'existe pas. Ainsi

`groupe_nominal_sujet(_x,_y,_z) .`

va fonctionner non pas en analyse de texte mais en génération de texte et va produire tous les groupes nominaux possibles avec notre dictionnaire, et leur traduction.

Et même, on peut donner la traduction, le programme va retrouver toutes les expressions ayant cette traduction :

`groupe_nominal_sujet(_x,_y, gn( chien, noir, masculin, singulier) ?`

alors qu'on l'avait écrit dans notre programme pour uniquement répondre à :

`groupe_nominal_sujet([le,chien,qui,a,la,couleur,noire],_reste,_traduction) ?`

4.5 Une règle élémentaire

On étend facilement le programme pour comprendre :

celui qui vole un œuf vole un bœuf
celui qui a des griffes est un prédateur

Qui s'enregistre en

```
affirme(_x, vole, bœuf, vrai) :- affirme(_x, vole, œuf, vrai).  
affirme(_x, vole, œuf, faux) :- affirme(_x, vole, bœuf, faux).
```

phrase([_ce, qui | _reste]) :-

```
ce(_ce, _nombre),  
forme_verbale_active(_reste, _rest, verbe(_typev, _racverbe1, _vrai1, _nombre)),  
groupe_nominal_compl(_rest, _res, gn(_type1, _racnom1, _genre1, _nombre1)),  
forme_verbale_active(_res, _re, verbe(_typev2, _racverbe2, _vrai2, _nombre)),  
groupe_nominal_compl(_re, [], gn(_type2, _racnom2, _genre2, _nombre2)),  
ajoute_règle(_racverbe1, _racnom1, _vrai1, _racverbe2, _racnom2, _vrai2).
```

ce(ceux, pluriel). ce(celui, singulier).

On remarque que la phrase universelle

un chat est un félin

s'analyse comme

celui qui est un chat est un félin

et il suffit donc de modifier l'appel de *ajoute_règle* en considérant que dans la phrase standard il y a un verbe « être » sous-entendu:

```
ajoute_règle(_racverbe1, _racnom1, _vrai1, _racverbe2, _racnom2, _vrai2) :-  
    assertz((affirme(_x, _racverbe2, _racnom2, _vrai2) :-  
                affirme(_x, _racverbe1, _racnom1, _vrai1)  
                affirme(_x, être, _racnom1, vrai),  
                affirme(_y, être, _racnom2, vrai)))).
```

5 Traitement des épithètes

Etendons maintenant le programme pour analyser les adjectifs épithètes des noms. Ces adjectifs sont associés à une propriété et permettent de spécifier la valeur de cette propriété.

Ainsi les adjectifs *lourd* et *léger* sont des valeurs pour la propriété *poids*, les adjectifs *mâle* et *féfelle*, pour la propriété *sexe*, les adjectifs *blanc*, *noir*, *rouge*, ... pour la propriété *couleur*, ...

5.1 Cas du sujet

un mouton mâle est un béliér

Si X est mouton ET si X a le sexe mâle alors X est béliér, doit s'enregistrer en :

affirme(_x, être, béliér, vrai) :-

affirme(_x, être, mouton, vrai),

affirme(_x, avoir, sexe, mâle, vrai).

et son équivalent négatif, un peu plus complexe du fait du ET devenant OU dans la négation

si X est béliér alors X n'est pas mouton OU X n'a pas le sexe mâle

et s'enregistrant sous forme des deux règles suivantes :

affirme(_x, être, mouton, faux) :-

affirme(_x, être, béliér, faux),

affirme(_x, avoir, sexe, mâle, vrai).

affirme(_x, avoir, sexe, mâle, faux) :-

affirme(_x, être, béliér, faux),

affirme(_x, être, mouton, vrai).

5.2 Cas du complément avec le verbe être

un béliér est un mouton mâle

se comprend comme deux affirmations distinctes:

un béliér est un mouton

ET *un béliér a le sexe mâle*

et s'enregistre donc en deux règles :

affirme(_x, être, mouton, vrai) :- affirme(_x, être, béliér, vrai).

affirme(_x, avoir, sexe, mâle, vrai) :- affirme(_x, être, béliér, vrai).

et leurs équivalents négatifs :

affirme(_x, être, béliér, faux) :- affirme(_x, être, mouton, faux).

affirme(_x, être, béliér, faux) :- affirme(_x, être, mâle, faux).

5.3 Cas du complément avec le verbe avoir

un chat a des griffes rétractiles

donne

affirme(_x, avoir, griffe, rétractile, vrai) :- affirme(_x, être, chat, vrai).

avec son équivalent négatif.

affirme(_x, être, chat, faux) :- affirme(_x, avoir, griffe, rétractile, faux).

Et on rajoutera au programme la règle générale suivante

affirme(_x, avoir, _attribut, vrai) :- affirme(_x, avoir, _attribut, _valeur, vrai).

signifiant entre autres que si quelqu'un a des griffes rétractiles, il a des griffes.

5.4 Cas du complément avec les autres verbes

les loups mangent les moutons femelles

se traduit en une règle à trois conditions :

Si X est mouton ET X a sexe femelle ET Y est loup alors Y mange X

affirme(_y, manger, _x, vrai) :-

affirme(_x, être, mouton, vrai),

affirme(_x, avoir, sexe, femelle, vrai),

affirme(_y, être, loup, vrai).

et un des 3 équivalents négatifs (chacun correspondant à la négation d'une des 3 conditions)

affirme(_y, être, loup, faux) :-

affirme(_x, être, mouton, vrai),

affirme(_x, avoir, sexe, femelle, vrai),

affirme(_y, manger, _x, faux).

Je vous laisse trouver les deux autres.

5.5 Cas de la négation

une brebis n'est pas un mouton mâle

ne suppose pas que la brebis soit un mouton, et n'a pas le même sens que

une brebis est un mouton non mâle (ou *une brebis est un mouton femelle*)

qui le suppose, est plus précis.

D'ailleurs, à la question

est-ce que un chat est un mouton mâle

Nous répondons

non

ce qui montre bien que

un chat n'est pas un mouton mâle

ne suppose pas que le chat soit un mouton. En fait, il faut mieux dire :

une brebis est un mouton non mâle

ou *une brebis est un mouton femelle*

et *celui qui est femelle n'est pas mâle*

et *celui qui est mâle n'est pas femelle*

Même raisonnement pour :

un chien n a pas des griffes rétractiles

un vautour ne mange pas la viande fraîche

un lion mange la viande fraîche

qui ne suppose pas que le chien a des griffes, et se traduit en

affirme(_x, avoir, griffe, rétractile, faux) :- affirme(_x, être, chien, vrai).

Alors que

un chien a des griffes non rétractiles

suppose que le chien a des griffes, et se traduit en :

affirme(_x, avoir, griffe, non(rétractile), vrai) :- affirme(_x, être, chien, vrai).

En résumé, le fait de dire

une brebis n est pas un mouton mâle

un chien n a pas les griffes rétractiles

ou *un vautour ne mange pas la viande fraîche*

ne suppose pas que la brebis soit un mouton, que le chien ait des griffes et que le vautour mange de la viande. Il faut dans ce cas faire porter la négation sur l'adjectif plutôt que sur le verbe, et dire :

une brebis est un mouton femelle (ou est un mouton non mâle)

un chien a des griffes non rétractiles (il n'y a pas de mot synonyme de *non rétractile*)

un vautour mange la viande pourrie (ou mange la viande non fraîche)

5.6 La transitivité du verbe être et des autres verbes

De

un lion est un mammifère

et *un mammifère est un animal*

on peut déduire que

un lion est un animal

de même :

un lion mange les mammifères

un chat est un mammifère

donc *un lion mange les chats*

mais de :

un lion mange les mammifères

un mammifère est un animal

on ne peut pas déduire que

un lion mange les animaux

En effet, il ne mange pas les huitres, on pourrait simplement déduire que

un lion mange quelques animaux

la transitivité est le mécanisme de base de prolog, et grâce aux règles d'équivalent négatif nous avons ajouté le chainage arrière, notre programme est donc capable de faire des déductions intéressantes.

5.7 Les limites du programme

Notre programme ne traite que des propositions universelles, s'appliquant à toutes les entités, et n'admet pas les exceptions. Donc, pas de *quelques*, *la plupart*, *presque tout*, *il existe un x* *qui*, c'est toujours *pour tout x* ...

les félidés ont des griffes rétractiles

se comprend comme

tous les félidés ont des griffes rétractiles

et *un félidé a les griffes rétractiles* aussi

tant pis pour le jaguar, il ne lui reste plus qu'à se créer sa propre famille !

plus gênant :

le petit chat noir mange du kitkat

est compris comme

tous les petits chats noirs mangent du kitkat

ce qui pose problème s'il y a plusieurs petits chats noirs dans l'espace considéré.

Il faut donc donner un nom à tous les individus, et les utiliser pour éviter de prendre une affirmation spécifique pour une affirmation universelle : Minou mange du kitkat

5.8 Première version de l'analyse du groupe nominal avec un adjectif épithète

L'adjectif peut être placé avant ou après le nom, d'où deux règles d'analyse :

```
groupe_nominal_sujet([_article, _nom, _adj | _reste], _reste,
                      gn(_racine_nom, _racine_adj, _genre, _nombre)) :-
    article(_article, _genre, _nombre),
    nom(_nom, _genre, _nombre, _racine_nom),
    adjectif(_adj, _genre, _nombre, _racine_adj).
groupe_nominal_sujet([_article, _adj, _nom | _reste], _reste,
                      gn(_racine_nom, _racine_adj, _genre, _nombre)) :-
    article(_article, _genre, _nombre),
    adjectif(_adj, _genre, _nombre, _racine_adj),
    nom(_nom, _genre, _nombre, _racine_nom).
```

5.9 Les règles à enregistrer

5.9.1 Découpe de la phrase universelle en deux propositions

La phrase universelle est du genre :

celui qui verbe1 un nom1 adjectif1 verbe2 un nom2 adjectif2

l'adjectif pouvant être une relative, et pouvant aussi être absent. L'analyse finit par l'appel :

```
ajoute_règle(_racverbe1, _racnom1, _racadj1, vrai, _racverbe2, _racnom2,
             _racadj2, _vrai2).
```

La phrase universelle se décompose donc en deux propositions : une condition et une conclusion :

si X verbe1 nom1 adjectif1 vrai1 alors X verbe2 nom2 adjectif2 vrai2

à quoi on ajoute l'équivalent négatif :

si X verbe2 nom2 adjectif2 non vrai2 alors X verbe1 nom1 adjectif1 non vrai1

5.9.2 Conversion des propositions en propositions élémentaires

Convertissons séparément chacune de ces deux propositions en propositions élémentaires avec seulement un nom **ou** un adjectif, mais pas les deux :

- *X être nom adjectif*
devient X être nom **et** X avoir attribut adjectif
- *X manger souris*
fait apparaître Y, que X mange, et qui est une souris
*X manger Y **et** Y être souris*
- ces propositions élémentaires se combinent et sont reliées par *et* , mais lorsqu'on en calculera la négation, les *et* deviendront *ou*, suivant en cela les bonnes lois de la logique.

On recense ainsi les cas suivants de propositions élémentaires et leur forme enregistrée :

Médor est un chien
affirme(_x, être, nom_d_entité, vrai).
Médor a une queue
affirme(_x, avoir, _attribut, _vrai).
Médor a une petite taille (a une longue queue, a des griffes non rétractiles, des doigts en corne....)
affirme(_x, avoir, _attribut, _valeur, _vrai).
Le cas particulier de la valeur numérique : Médor a 4 pattes
affirme(_x, avoir, patte, 4, vrai).
Le cas particulier d'une valeur mesurable avec une unité : Médor a un poids de 2 kg
Que l'on pourrait enregistrer
affirme(_x,avoir,poids, 2, kg, vrai)
ou affirme(_x,avoir,poids, mesure(2, kg), vrai)
Mais on utilisera la notation en nuplet que permet prolog :
affirme(_x, avoir, poids, (2, kg), _vrai)
Médor aboie
affirme(_x, _verbe, _vrai)
Médor mange une souris
affirme(_x, _verbe, _y, _vrai).
Coco construit des nids
affirme(_x, _verbe, _attribut, vrai).
beezbeez construit des nids en cire
affirme(_x, _verbe, _attribut, _valeur , _vrai).

Le sujet est toujours une entité.

Les noms communs sont divisés en 2 classes :

- Les noms d'entité (par exemple les noms des classes, familles, ordres, espèces d'animaux ou alors les noms des régions, départements, fleuves, villes, ...).
- Les noms d'attributs servant à décrire ces entités (la taille, le poids, la forme, ... la surface, le débit, la population, ...)

Ainsi que tous les mots annexes servant à les décrire :

Les parties de ces entités : patte, aile, bec, museau, poumon, plume,

Les noms représentant les choses sur lesquels ces entités agissent : nid, terre, eau, avoine, blé, électricité, industrie, tourisme,

5.9.3 Enregistrement des règles

Suite à la conversion des deux propositions condition et conclusion en propositions élémentaires, il s'agit de créer les règles prolog représentant l'affirmation et celles représentant ses équivalents négatifs. Si la conclusion ne contient qu'une seule proposition élémentaire et la condition, une ou plusieurs, reliées par un *et*, une seule règle suffit :

un mouton mâle est un bélier (si X est mouton **et** X a sexe mâle alors X est bélier)
affirme(_x, être, bélier, vrai) :-
 affirme(_x, être, mouton, vrai),
 affirme(_x, avoir, sexe, mâle, vrai).

Si la conclusion est un *et*, il faut créer une règle pour chaque élément du *et*, avec la même condition

un chien est un animal carnivore (si X est chien alors X est animal **et** X est carnivore)
soit donc la même chose que : *un chien est un animal et un chien est carnivore*
affirme(_x, être, animal, vrai) :- affirme(_x, être, chien, vrai).
affirme(_x, avoir, nourriture, carnivore, vrai) :- affirme(_x, être, chien, vrai).

Et l'équivalent négatif :

affirme(_x, être, chien, faux) :- affirme(_x, être, animal, faux).
affirme(_x, être, chien, faux) :- affirme(_x, avoir, nourriture, carnivore, faux).

Si la conclusion est un *ou*, il faut créer une règle pour chaque élément du *ou*, en ajoutant en condition la négation de l'autre. Cette conclusion en *ou* apparaît lorsque l'on calcule l'équivalent négatif de

un mouton mâle est un bélier

dont l'équivalent négatif est : si X n'est pas bélier alors X n'est pas mouton **ou** X n'a pas sexe mâle

affirme(_x, avoir, sexe, mâle, faux) :- affirme(_x, être, bélier, faux), affirme(_x, être, mouton, vrai).
affirme(_x, être, mouton, faux) :- affirme(_x, être, bélier, faux), affirme(_x, avoir, sexe, mâle, vrai).

Si la condition est un *et*, il suffit de mettre tous les éléments en condition de la règle

un mouton mâle est un bélier (si X est mouton **et** X est mâle alors X est bélier)
celui qui mange la petite souris est un rapace

si X mange Y et Y est souris et Y a taille petit alors X est rapace
affirme(_x, être, rapace, vrai) :-
 affirme(x, manger, _y, vrai), affirme(_y, être, souris, vrai), affirme(_y, avoir, taille, petit, vrai).

Comme tous ces calculs de négation de proposition vont se faire automatiquement, on va pouvoir calculer sans se casser la tête tous les équivalents négatifs. Ici par exemple, le *et* à trois éléments va devenir un *ou* à 3 éléments d'où trois équivalents négatifs.

affirme(_x, manger, _y, faux) :-
 affirme(_x, être, rapace, faux), affirme(_y, être, souris, vrai), affirme(_y, avoir, taille, petit, vrai).
affirme(_y, être, souris, faux) :-
 affirme(_x, être, rapace, faux), affirme(_y, avoir, taille, petit, vrai), affirme(_x, manger, _y, vrai).
affirme(_y, avoir, taille, petit, faux) :-
 affirme(_x, être, rapace, faux), affirme(_y, être, souris, vrai), affirme(_x, manger, _y, vrai).

6 Les formules logiques

A ce stade, il apparaît que la meilleure chose à faire remonter de l'analyse du groupe nominal est non pas la racine du nom mais une formule logique faisant référence à une variable `_x` représentant l'entité « nom ». cette formule pour un nom seul est :

```
affirme(_x,être,_racine_nom,vrai).
```

Ajouter un épithète ou une relative à un nom, c'est simplement ajouter à l'aide de l'opérateur ET un élément à cette formule, ainsi *un chat noir* a comme formule logique :

```
et(affirme(_x,être,chat,vrai), affirme(_x,avoir,couleur,noir,vrai))
```

l'analyse du groupe nominal avec un épithète devient alors :

```
groupe_nominal_sujet([_article, _nom, _adj | _reste], _reste,
                      _x, gn(et(affirme(_x,être,_racine_nom,vrai),
                               affirme(_x,avoir,_atr,_racine_adj,vrai)), _genre, _nombre)) :-
    article(_article, _genre, _nombre),
    nom(_nom, _genre, _nombre, _racine_nom),
    adjectif(_adj, _genre, _nombre, _racine_adj),
    atr(_atr,_racine_adj).
```

```
atr(couleur, blanc).    atr(couleur, noir).    atr(couleur, gris).    atr(taille, petit).    atr(taille,
grand).
```

6.1 Analyse des relatives

L'épithète n'est qu'un attribut du nom parmi d'autres, que sont les relatives.

Expression	transformation
le chat <i>petit</i>	<code>affirme(_x,avoir,taille,petit,vrai)</code>
l animal <i>dont le cou est grand</i>	<code>affirme(_x,avoir,cou,grand,vrai)</code>
le chat <i>dont la taille est petite</i>	<code>affirme(_x,avoir,taille,petit,vrai)</code>
le chat <i>de petite taille</i>	<code>affirme(_x,avoir,taille,petit,vrai)</code>
le chat <i>qui mange une souris</i>	<code>et(affirme(_x,manger,_y,vrai),</code> <code>affirme(_y,être,souris,vrai))</code>
le fils <i>d Hector</i>	<code>et(affirme(_y,être,hector,vrai),</code> <code>affirme(_y,avoir,fils,_x,vrai))</code>

```
groupe_nominal_sujet([_article, _nom | _rest], _reste,
                      _x, gn(et(affirme(_x,être,_racnom,vrai),_rel),_genre,_nombre)) :-
    article(_article, _genre, _nombre),
    nom(_nom, _genre, _nombre, _racnom),
    relative(_rest, _reste, _x, _rel).
```

```
relative([dont, _article, _attribut, _est, _adj | _reste], _reste, _x,affirme(_x,avoir,_atr,_racadj,vrai)) :-
    article(_article, _genre, _nombre),
    nom(_attribut, _genre, _nombre, _atr),
    verbe(_est, _nombre, être),
    adjectif(_adj, _genre, _nombre, _racadj).
```

```

relative_dont([_article, _attribut, _est | _rest], _reste, _x, _rel) :-
    article(_article, _genre, _nombre),
    nom(_attribut, _genre, _nombre, _atr), nomde(_atr, _),
    verbe(_est, _nombre, être),
    quantité_relative(_rest, _reste, _atr, _x, _rel),!.
relative([_de, _adj, _attribut | _reste], _reste, _x, affirme(_x, avoir, _atr, _racadj, vrai)) :-
    de_à(_de),
    nom(_attribut, _genre, _nombre, _atr),
    adjectif(_adj, _genre, _nombre, _racadj).
relative_de([_attribut | _rest], _reste, _x, _rel) :-
    nom(_attribut, _genre, _nombre, _atr), nomde(_atr, _),
    quantité_relative(_rest, _reste, _atr, _x, _rel).
relative([_qui | _rest], _reste, _x, _rel) :-
    forme_verbale_active(_rest, _res, verbe(_racverbe, _vrai, _nombre)),
    relative_qui(_res, _reste, _racverbe, _vrai, _nombre, _x, _rel).

relative_qui(_phrase, _reste, _verbe, _vrai, _nomb, _x, et_cond(_rel, affirme(_x, _verbe, _y, _vrai))):-
    nonêtreavoir(_verbe),
    groupe_nominal_complément(_verbe, _phrase, _reste, _y, gn(_rel, _), _rel\=true,!.
relative_qui([_article, _attribut, _adj, | _reste], _reste, avoir, _vrai, _nombre1, _x,
    affirme(_x, avoir, _atr, _racadj, _vrai)) :-
    article(_article, _genre, _nombre),
    nom(_attribut, _genre, _nombre, _atr),
    adjectif(_adj, _genre, _nombre, _racadj).
relative_qui([_article, _attribut, _adj, | _reste], _reste, avoir, _vrai, _nombre1, _x, _rel) :-
    article(_article, _genre, _nombre),
    nom(_attribut, _genre, _nombre, _atr),
    quantité_relative(_rest, _reste, _atr, _x, _rel)..

```

6.2 Analyse des quantités mesurables

```

% 2 kg
quantité_relative(_in, _reste, _atr, _x, _rel):-
    quantité(_in, _reste, _atr, _x, _rel),!.
% supérieur à celui du chat
quantité_relative([_comp, _à | _in], _reste, _atr, _x, et(affirme(_x, avoir, _atr, _valu, vrai),
    et(_rel, et_calc(affirme(_y, avoir, _atr, _z, vrai), être(_cmp, _valu, _z))))):-
    compare([_comp, _à], _cmp), qr(_in, _rest),
    groupe_nominal_sujet_joker(_rest, _reste, _y, gn(_rel, _)).
% celui du chat
quantité_relative(_in, _reste, _atr, _x, et(affirme(_x, avoir, _atr, _valu, vrai),
    et(_rel, et_calc(affirme(_y, avoir, _atr, _z, vrai), être(égal, _valu, _z))))):-
    qr(_in, _rest),
    groupe_nominal_sujet_joker(_rest, _reste, _y, gn(_rel, _)).
% le même poids que le chat
quantité_relative([_le, même, _atrs, _que | _rest], _reste, _atr, _x, et(affirme(_x, avoir, _atr, _valu, vrai),
    et(_rel, et_calc(affirme(_y, avoir, _atr, _z, vrai), être(égal, _valu, _z))))):-
    article(_le, _genre, _nombre), que(_que),
    nom(_atrs, _genre, _nombre, _atr), nomde(_atr, _),
    groupe_nominal_sujet(_rest, _reste, _y, gn(_rel, _)).

```

```

qr([_pronom,_de|_rest],_rest):-ce(_pronom,_),de(_de).
qr([_le,même,que,_pronom,_de|_rest],_rest):-article(_le,_),ce(_pronom,_),de(_de).

quantité(_in,_reste,_at,_x,affirme(_x,avoir,_atri,_valu,vrai)):-
    valquantité(_in,_reste,_atr,_valu),complete(_at,_atr,_atri).
% au plus 2 kg
quantité([_au,_plus|_rest],_reste,_at,_x,
    et_calc(affirme(_x,avoir,_atri,_val,vrai),être(_comp,_val,_valu))):-
    compare([_au,_plus],_comp),
    valquantité(_rest,_reste,_atr,_valu),complete(_at,_atr,_atri).
% au plus de 2 kg
quantité([_au,_plus,de|_rest],_reste,_at,_x,
    et_calc(affirme(_x,avoir,_atri,_val,vrai),être(_comp,_val,_valu))):-
    compare([_au,_plus],_comp),
    valquantité(_rest,_reste,_atr,_valu),complete(_at,_atr,_atri).
% au plus la moyenne du poids des chats
quantité([_au,_plus|_rest],_reste,_at,_x,
    et_calc(affirme(_x,avoir,_atri,_val,vrai),et(_calcule,être(_comp,_val,_valu)))):-
    compare([_au,_plus],_comp),
    groupe_nominal_valeur_joker(_rest,_reste,_atr,_valu,_calcule),complete(_at,_atr,_atri).
quantité([de|_in],_reste,_at,_x,_rel):-quantité(_in,_reste,_at,_x,_rel).
quantité([d|_in],_reste,_at,_x,_rel):-quantité(_in,_reste,_at,_x,_rel).

complete(_at,_atr,_atr):-var(_at),!.
complete(_at,_atr,_atr):-nonvar(_at).

compare([supérieur_égal,_à],supérieurégal):-à(_à).
compare([supérieur,_à],supérieur):-à(_à).
compare([supérieure,_à],supérieur):-à(_à).
compare([supérieurs,_à],supérieur):-à(_à).
compare([supérieures,_à],supérieur):-à(_à).
compare([identique,_à],égal):-à(_à).
compare([égal,_à],égal):-à(_à).
compare([égaux,_à],égal):-à(_à).

compare([inférieur_égal,_à],inférieurégal):-à(_à).
compare([inférieur,_à],inférieur):-à(_à).
compare([inférieure,_à],inférieur):-à(_à).
compare([inférieurs,_à],inférieur):-à(_à).
compare([inférieures,_à],inférieur):-à(_à).
compare([identiques,_à],égal):-à(_à).
compare([égale,_à],égal):-à(_à).
compare([égales,_à],égal):-à(_à).

compare([_le,même],égal):-article(_le,_).
compare([plus,de],supérieur).
compare([au,plus],inférieurégal).
compare([au,moins],supérieurégal).

compare([autant,de],égal).
compare([au,maximum],inférieurégal).
compare([moins,de],inférieur).
compare([au,minimum],supérieurégal).

% 3 m de hauteur
valquantité([_nomb,_unité,de,_adj|_reste],_reste,_adj,(_valu,_abréviation)):-
    nombre(_nomb,_valu),
    unité(_unité,_abréviation),
    nomde(_adj,_),!.
% 3 m de haut
valquantité([_nomb,_unité,de,_adj|_reste],_reste,_atr,(_valu,_abréviation)):-
    nombre(_nomb,_valu),
    unité(_unité,_abréviation),
    ladjectif(_adj,_,_,_atr),!.
% 3 m
valquantité([_nomb,_unité|_reste],_reste,_atr,(_valu,_abréviation)):-
    nombre(_nomb,_valu),
    unité(_unité,_abréviation),

```

```

dimension(_atr,_abréviation),!.
% 3 doigts en corne
valquantité([_nomb,_nom,_prep,_nom2 | _reste], _reste,(_atr,_atr2),_valu):-
    nombre(_nomb,_valu),
    nom(_nom,_,_,_atr),
    prep(_prep),
    nom(_nom2,_,_,_atr2),!.
% 2 ailes colorées
valquantité([_nomb,_nom,_adj | _reste], _reste,(_atr,_atr1),_valu):-
    nombre(_nomb,_valu),
    nom(_nom,_genre,_nombre,_atr),
    adjectif(_adj,_genre,_nombre,_atr1,_),!.
% 1 petite nageoire dorsale
valquantité([_nomb,_adj,_nom,_adj2 | _reste], _reste,(_atr,_atr1,_atr2),_valu):-
    nombre(_nomb,_valu),
    adjectif(_adj,_genre,_nombre,_atr1,_),
    nom(_nom,_genre,_nombre,_atr),
    adjectif(_adj2,_genre,_nombre,_atr2,_),!.
% 2 pattes
valquantité([_nomb,_nom | _reste], _reste,_atr,_valu):-
    nombre(_nomb,_valu),
    nom(_nom,_,_,_atr),!.

```

dimension(taille,m).	dimension(poids,kg).	dimension(hauteur,m).
dimension(longueur,m).	dimension(largeur,m).	dimension(périmètre,m).
dimension(circonférence,m).	dimension(profondeur,m).	dimension(épaisseur,m).
dimension(surface,m2).	dimension(superficie,km2).	dimension(étendue,m2).
dimension(volume,m3).	dimension(vitesse,kmh).	dimension(durée,s).
dimension(densité,habm2).	dimension(population,hab).	dimension(niveau,m).
dimension(production,kg).	dimension(débit,m3heure).	dimension(température,d).
dimension(pression,bar).	dimension(gestation,sem).	dimension(longévité,an).
dimension(incubation,j).	dimension(course,kmh).	dimension(nage,kmh).
dimension(vol,kmh).		

```

dimension(_atr,_x) :- var(_atr),novar(_x),conversion(_x,_,_y),dimension(_atr,_y),!.
dimension(_atr,_x) :- var(_atr),novar(_x),conversion(_y,_,_x),dimension(_atr,_y),!.

```

```

unité(_x,_y) :-
    abréviation(_x,_,_y),!.
unité(_x,_y) :-
    abréviation(,_x,_y),!.
unité(_x,_x) :- unité(_x).

```

abréviation(kilo,kilos,kg).	abréviation(tonne,tonnes,t).	abréviation(quintal,quintaux,q).
abréviation(gramme,grammes,g).	abréviation(heure,heures,h).	abréviation(minute,minutes,mn).
abréviation(seconde,secondes,s).	abréviation(jour,jours,j).	abréviation(semaine,semaines,sem).
abréviation(an,ans,an).	abréviation(mètre,mètres,m).	abréviation(kilomètre,kilomètres,km).
abréviation(centimètre,centimètres,cm).	abréviation(millimètre,millimètres,mm).	

conversion(kg,1000,g).	conversion(t,1000,kg).	conversion(q,100,kg).
conversion(km,1000,m).	conversion(m,100,cm).	conversion(m,1000,mm).
conversion(km2,10^6,m2).	conversion(litre,1,l).	conversion(m3,10^4,l).
conversion(h,3600,s).	conversion(h,60,mn).	conversion(mn,60,s).
conversion(j,24,h).	conversion(sem,7,j).	conversion(an,12,mois).
conversion(an,365,j).	conversion(mois,30.5,j).	

6.3 Les prédicats numériques globaux

Il s'agit de calculer une valeur dépendant de plusieurs affirmations : la moyenne, la somme, le maximum, le minimum, ...

```
maximum([_le, plus, grand, des, _attrs | _reste], _reste, _le, plus, grand, _attrs).
maximum([_le, plus, grand, _attrs | _reste], _reste, _le, plus, grand, _attrs).
maximum([_le, _attrs, le, plus, grand | _reste], _reste, _le, plus, grand, _attrs).
maximum([_le, _attrs, du, plus, grand | _reste], _reste, _le, plus, grand, _attrs).
maximum([_le, _attrs, du, plus, grand | _reste], [joker | _reste], _le, plus, grand, _attrs).
maximum([_le, _attrs, de, la, plus, grand | _reste], _reste, _le, plus, grand, _attrs).
maximum([_le, _attrs, de, la, plus, grand | _reste], [joker | _reste], _le, plus, grand, _attrs).
```

```
plus_adj(plus, _adj, max_list) :- comparatif(_adj, _, _), !.
plus_adj(plus, _adj, min_list) :- comparatif(_, _adj, _), !.
plus_adj(moins, _adj, min_list) :- comparatif(_adj, _, _), !.
plus_adj(moins, _adj, max_list) :- comparatif(_, _adj, _), !.
```

moyen(moyenne, féminin, avr_list).	moyen(moyen, féminin, avr_list).
moyen(maximum, _, max_list).	moyen(minimum, _, min_list).
moyen(somme, féminin, sum_list).	moyen(cumul, masculin, sum_list).
moyen(addition, féminin, sum_list).	

% la moyenne des poids des chats

```
groupe_nominal_valeur([_la, _moyenne, des, _attrs | _reste], _reste, _atr, _resu,
                       calcule(_minmax_list, x, _racnom, _atr, _resu)) :-
    article(_la, _genre, singulier),
    moyen(_moyenne, _genre, _minmax_list),
    de(_des),
    dim(_attrs, _atr),
    groupe_nominal_sujet(_reste, _reste, _x, gn(_racnom, _genre2, _nombre2)).
groupe_nominal_valeur([_la, _moyenne, de, _la2, _attrs | _reste], _reste, _atr, _resu,
                       calcule(_minmax_list, x, _racnom, _atr, _resu)) :-
    article(_la, _genre, singulier),
    moyen(_moyenne, _genre, _minmax_list),
    de(_de),
    article(_la2, _, _),
    dim(_attrs, _atr),
    groupe_nominal_sujet(_reste, _reste, _x, gn(_racnom, _genre2, _nombre2)).
groupe_nominal_valeur([_la, _attrs, _moyenne | _reste], _reste, _atr, _resu,
                       calcule(_minmax_list, x, _racnom, _atr, _resu)) :-
    article(_la, _genre, singulier),
    dim(_attrs, _atr),
    moyen(_moyenne, _genre, _minmax_list), !,
    groupe_nominal_sujet(_reste, _reste, _x, gn(_racnom, _genre2, _nombre2)).
groupe_nominal_valeur(_phrase, _reste, _atr, _resu, calcule(_minmax_list, x, _racnom, _atr, _resu)) :-
    maximum(_phrase, _reste, _le, plus, _adj, _attrs),
    article(_le, _, _),
    adjectif(_adj, _, _, _racadj, _),
    plus_adj(_plus, _racadj, _minmax_list),
    dim(_attrs, _atr),
    groupe_nominal_sujet(_reste, _reste, _x, gn(_racnom, _genre2, _nombre2)).
```

```

groupe_nominal_valeur([_le, nombre, _de | _rest], _reste, _ , _resu, compte(_x, _racnom, _resu)) :-
    de(_de),
    groupe_nominal_sujet([joker | _rest] , _reste, _x, gn( _racnom, _genre2, _nombre2)).

groupe_nominal_valeur_joker([_le|_in], _reste, _atr, _y, _gn):-
    article(_le, _ , _),!, groupe_nominal_valeur([_le|_in], _reste, _atr, _y, _gn ).
groupe_nominal_valeur_joker(_in, _reste, _atr, _y, _gn):-
    groupe_nominal_valeur([joker|_in], _reste, _atr, _y, _gn ).

% predicats de calculs globaux
calcule(_pred, _x, _racnom, _attribut, _resu):-
    findall(_x, _racnom, _l1),
    sort(_l1, _l2),
    member(_z, _l2), affirme(_z, avoir, _attribut, (_zz, _v), vrai),!,
    findall(_tv,
        (member(_y, _l2), affirme(_y, avoir, _attribut, (_tu, _u), vrai), conver(_tu, _u, _tv, _v)),
        _l),
    =..(_fon, [_pred, _l, _resu]),
    call(_fon),!.
calcule(_pred, _x, _racnom, _attribut, 0).

conver(_tu, _u, _tu, _u):-!.
conver(_tu, _u, _tv, _v) :-
    conversion(_u, _n, _v),!
    is(_tv, _n*_tu).
conver(_tu, _u, _tv, _v) :-
    conversion(_v, _n, _u),!
    is(_tv, _tu/_n).

% certains ne sont pas prévus directement dans prolog :
% calcul de la moyenne
avr_list([], 0):-!.
avr_list(_l, _resu):-
    sum_list(_l, _sum),
    length(_l, _nb),
    is(_resu, _sum/_nb),!.
avr_list(_l, 0).

% calcul du nombre d'éléments
compte(_x, _racnom, _resu):-
    findall(_x, _racnom, _l1),
    sort(_l1, _l),
    length(_l, _resu),!.
compte(_x, _racnom, 0).

```

6.4 La formule logique de la phrase

De même que le groupe nominal et la relative rendent une formule logique, l'analyse de la phrase rend une formule logique constituée de la formule logique du sujet, celle du verbe et celle du complément, dans le but de pouvoir déclarer facilement la règle résultante, quelque chose du genre :

```
affirme(_x,_verbe,_y,_vrai) :-  
    formule_sujet,  
    formule_complément.
```

si le complément est absent, sa formule se simplifie en « true », condition toujours réalisée qu'on ne mentionnera même pas.

la nature du complément dépend du verbe.

Avec tous les verbes, y compris être et avoir, on peut avoir un complément qui a une structure de groupe_nominal_sujet, décrivant une entité.

Avec le verbe avoir, on peut avoir aussi une définition d'attribut : *la couleur noire*, ou *un poids de 2 kg*, ou *un long cou recourbé* ou *un poids supérieur à la moyenne du poids des chats* ou *des cornes en os*, toutes choses qui ne peuvent pas apparaître, dans notre modèle, en tant que sujet.

Avec le verbe être, on peut avoir aussi une définition d'attribut, par une structure du genre *de petite taille* ou *de taille supérieure à celle de Médor* ou une structure du genre *le fils de Médor*

Avec les verbes qui ne sont ni être ni avoir, on peut trouver un simple adverbe ou des structures regroupant préposition, nom, adjectif (*sur la terre*, *des nids sur terre*, ...)

affirmation(_phrase,_resu) :-

groupe_nominal_sujet(_phrase, _reste, _x, gn(_racnom, _genre, _nombre)),

forme_verbale_active(_reste, _rest, verbe(_racverbe, _vrai, _nombre)),

complément_active(_rest,['.'],_x,_racnom,_nombre,_racverbe,_vrai,_resu).

complément_active([_adv | _reste], _reste, _x, _racnom, _, _racverbe, _vrai,

phrase(_racnom, affirme(_x, _racverbe, _adv, _vrai), true)):-

adverbe(_adv),!.

complément_active(_in, _rest, _x, _racnom, _nombre, _racverbe, _vrai, phrase(_si, _alorsv, _alors)):-

groupe_nominal_complément(_racverbe, _in, _rest, _y, gn(_racnom2, _genre2, _nombre2)),!

ctrl_participe_est(_genre, _nombre, _racverbe, _genre2, _nombre2),

sialors(_racnom, _racverbe, _vrai, _racnom2, _x, _y, _si, _alorsv, _alors).

complément_active(_in, _in, _x, _racnom, _, _racverbe, _vrai,

phrase(_racnom, affirme(_x, _racverbe, _vrai), true)).

adverbe(vite).

Il semble y avoir toujours deux entités, nommées _x et _y, correspondant l'une au sujet et l'autre au complément or, il n'y en a souvent qu'une en cause (le sujet) le complément ne précisant que des attributs du sujet (avec le verbe être c'est toujours le cas) Il faut alors faire en sorte que _x=_y. La propriété est dans ce cas complètement décrite dans la formule du complément, qui peut prendre la place de la formule du verbe, en n'y conservant que la valeur de _vrai. _y et le complément disparaissant complètement, remplacés par une condition toujours vrai : « true ».

sialors(_racnom, être, vrai, et_cond(_v1, _v2), _x, _x, _racnom, _v2, _v1):-!.

sialors(_racnom, être, vrai, _racnom2, _x, _x, _racnom, _racnom2, true):-!.

et dans le cas où le verbe être est à la forme négative, il faut propager cette négation dans la formule du complément, toujours affirmatif.

sialors(_racnom, être, faux, et(_v1, _v2), _x, _x, _racnom, _nega, true) :- neg(et(_v1, _v2), _nega),!.

sialors(_racnom, être, faux, et_cond(_v1, _v2), _x, _x, _racnom, _nv2, _v1):-neg(_v2, _nv2),!.

sialors(_racnom, être, faux, affirme(_x, être, _y, vrai), _x, _x, _racnom, affirme(_x, être, _y, faux), true) :-!.

sialors(_racnom, être, faux, affirme(_x, avoir, _y, _z, vrai), _x, _x, _racnom, affirme(_x, avoir, _y, _z, faux), true) :-!.

Avec le verbe avoir, comme avec le verbe être, la propriété `affirme(_x,avoir,...)` est complètement décrite dans le complément, et c'est cette formule du complément qui doit là encore prendre la place de la formule du verbe, en n'y conservant que la valeur de `_vrai`. `_y` et le complément disparaissant.

```
sialors(_racnom,avoir,_vrai,affirme(_z,avoir,_p,_vrai1),_x,_y,
      _racnom,affirme(_x,avoir,_p,_vrai),true):-!.
sialors(_racnom,avoir,_vrai,affirme(_z,avoir,_p,_v,_vrai1),_x,_y,
      _racnom,affirme(_x,avoir,_p,_v,_vrai),true):-!.
sialors(_racnom,avoir,vrai,et(_r,_q),_x,_x,_racnom,et(_r,_q),true):-!.
sialors(_racnom,avoir,faux,et(_r,_q),_x,_x,_racnom,ou(_nr,_nq),true):-!,neg(r,nr),neg(_q,nq).
sialors(_racnom,avoir,vrai,et_cond(_v1,_v2),_x,_x,_racnom,_v2,_v1):-!.
sialors(_racnom,avoir,faux,et_cond(_v1,_v2),_x,_x,_racnom,_nv2,_v1):-neg(_v2,_nv2),!.
```

6.5 Enregistrement des règles

L'analyse de la phrase rend une formule logique : `phrase(_si,_alorsv,_alors)`
`_si` représente les conditions liées au sujet, basées sur la variable `_x`
`_alors` représente les conditions liées au complément, basées sur la variable `_y`
`_alorsv` représente la conclusion : `_x racverbe _y _vrai`

On serait tenté de déclarer simplement :
`assertz((_alorsv :-et(_si,_alors))).`

Ou plutôt
`assertz((_alorsv :- (_si,_alors))).`

Car en prolog une liste de condition dans une règle réalise un « et ».

Soit donc :
`_alorsv :-`
`_si,`
`_alors.`

Et, pour l'équivalent négatif :
`neg(_si,_negsi),`
`neg(_alors,_negalors),`
`neg(_alorsv,_negalorsv),`
`assertz((ou(_negsi,_negalors):-_negalorsv)).`

Mais ce serait trop simple. La conclusion `_alorsv` n'est pas toujours une proposition simple, elle peut comporter un *et*. On a déjà fait intervenir des négations de formule logique dans « `sialors` » et voici encore des négations pour l'équivalent négatif. Les conditions ne sont donc pas toujours regroupées par des « et », qui sont devenus des « ou » dans ces négations.

6.5.1 Négation d'une formule logique

Voici comment se déclare la négation d'une formule logique, contenant ou non des « et » et des « ou » :

Tout d'abord les propositions élémentaires :

```
neg(affirme(_x,_verbe,vrai) , affirme(_x,_verbe,faux)).
neg(affirme(_x,_verbe,faux) , affirme(_x,_verbe,vrai)).
neg(affirme(_x,_verbe,_y,vrai) , affirme(_x,_verbe,_y,faux)).
neg(affirme(_x,_verbe,_y,faux) , affirme(_x,_verbe,_y,vrai)).
neg(affirme(_x,_verbe,_y,_z,vrai) , affirme(_x,_verbe,_y,_z,faux)).
neg(affirme(_x,_verbe,_y,_z,faux) , affirme(_x,_verbe,_y,_z,vrai)).
```

Puis les propositions contenant des « et » et des « ou » :

```
neg(et(_a,_b), ou(_nega,_negb)):- neg(_a,_nega),neg(_b,_negb).
neg(ou(_a,_b), et(_nega,_negb)):- neg(_a,_nega),neg(_b,_negb).
neg(true,false).
neg(false,true).
neg(et_cond(_rel,_aff), et_cond(_rel,_naff)) :- neg(_aff,_naff).
neg(et_calc(_aff,_rel), et_calc(_aff,_nrel)) :- neg(_rel,_nrel).
```

```
neg(être(_cmp,_a,_b), être(_ncmp,_a,_b)) :- nega(_cmp,_ncmp).
```

nega(inférieur , supérieurégal).	nega(supérieur , inférieurégal).
nega(supérieurégal , inférieur).	nega(inférieurégal , supérieur).
nega(égal , différent).	nega(différent , égal).

et_cond est un *et* dont l'un des deux éléments (que l'on place alors le premier) est en fait une condition liée à une entité apparue en complément d'un verbe dans une relative.

et_calc est un *et* dans lequel le second élément n'est pas une des formes de *affirme(...)* mais un prédicat introduit pour les calculs et comparaisons de valeurs numériques.

être(supérieur, _a, _b)

Dans une affirmations du genre :

le chat a un poids supérieur à 5 kg

et_calc sert à représenter la contrainte (> 5 kg) que l'on pourrait vérifier si la valeur de l'attribut est connu au moment de l'affirmation, mais si elle ne l'est pas encore, la contrainte serait à vérifier plus tard.

Dans une question du genre :

Quels sont les animaux de plus de 5 kg

et_calc n'est qu'un filtre à prendre en compte, comme toute relative.

6.5.2 Les comparaisons de valeurs numériques

Les mesures (3 kg) sont représentées par un nombre et une unité de mesure : (nombre, unité)
et les nombres par un nombre (4 pattes)

```
être(_cmp,(_a,_u),(_b,_u)):-!,
    être(_cmp,_a,_b).
être(_cmp,(_a,_u),(_b,_v)):-
    conversion(_u,_n,_v),!,
    is(_va,n*_a),
    être(_cmp,_va,_b).
être(_cmp,(_a,_u),(_b,_v)):-
    conversion(_v,_n,_u),!,
    is(_vb,n*_b),
    être(_cmp,_a,_vb).

% on peut aussi comparer des valeurs imprécises : une taille haute est plus haute qu'une taille basse
être(_cmp,_a,_b):-
    atom(_a),
    atom(_b),!,
    conv(_a,_numa),
    conv(_b,_numb),
    être(_cmp,_numa,_numb).

être(égal,_a,_a):-!.
être(différent,_a,_a):-!,fail.
être(différent,_a,_b).

être(_cmp,_a,_b):-var(_a),fail,!.
être(_cmp,_a,_b):-var(_b),fail,!.
être(inférieur,_a,_b):-number(_a),number(_b),_a<_b.
être(supérieur,_a,_b):-number(_a),number(_b),_a>_b.
être(inférieurégal,_a,_b):-number(_a),number(_b),_a<=_b.
être(supérieurégal,_a,_b):-number(_a),number(_b),_a>=_b.

conv(minuscule,0):-!.
conv(_adj,1):-comparatif(_,_adj,_),!.
conv(moyen,2).
conv(standard,2).
conv(_adj,3):-comparatif(_adj,_,_),!.
conv(énorme,4).

comparatif(haut,bas,niveau).      comparatif(haut,bas,hauteur).
comparatif(long,court,longueur). comparatif(large,étroit,largeur).
comparatif(profond,bas,profondeur). comparatif(épais,mince,épaisseur).
comparatif(rapide,lent,vitesse).  comparatif(grand,petit,_).
comparatif(important,bas,_).      comparatif(étendu,petit,_any).
comparatif(volumineux,petit,volume). comparatif(gros,petit,_any).
comparatif(lourd,léger,poids).    comparatif(long,court,durée).
comparatif(obtu,aigu,angle).       comparatif(chaud,froid,température).
comparatif(énorme,minuscule,_).   comparatif(vieux,jeune,âge).
```

6.5.3 enregistrement

l'analyse de la phrase se termine par la séquence d'enregistrement :

pour la phrase elle-même :

```
enregistrer(_alorsv,et(_si,_alors),
```

et pour les équivalents négatifs :

```
enregistrer(_negsi,et(_alors,_negalorsv)),
```

```
enregistrer(_negalors,et(_si,_negalorsv)).
```

```
enregistrer(_x,ou(_a,_b)):- enregistrer(_x,_a), enregistrer(_x,_b),!.
```

```
enregistrer(false,_cond):-!.
```

```
enregistrer(true,_cond):-!.
```

```
enregistrer(ou(_a,_b),_cond):-
```

```
neg(_a,_nega), neg(_b,_negb),
```

```
enregistrer(_a,et(_negb,_cond)), enregistrer(_b,et(_nega,_cond)),!.
```

```
enregistrer(et(_a,_b),_cond):-
```

```
enregistrer(_a,_cond), enregistrer(_b,_cond),!.
```

```
enregistrer(et_cond(_a,_b),_cond):-
```

```
enregistrer(_b,et(_a,_cond)),!.
```

```
enregistrer(et_calc(_a,_b),_cond):-
```

```
enregistrer(_b,et(_a,_cond)),!.
```

```
enregistrer(_a,_cond):-
```

```
flat(_cond,[],_liste),
```

```
reverse(_liste,_list),
```

```
assertz(_a,_list).
```

6.6 Dernière amélioration de l'analyse du groupe nominal sujet

Cette dernière version permet la présence éventuelle, en plus du nom, de 3 adjectifs, d'un nombre quelconque de relatives, et la gestion de noms composés.

On pourra donc analyser par exemple :

un grand oiseau marin piscivore qui a un long bec crochu et dont les pattes sont grandes.

```
groupe_nominal_sujet([_article | _res], _reste, _x, gn(_rel, _genre, _nombre)):-
```

```
article(_article, _genre, _nombre),
```

```
adjectif(_res, _rest, _genre, _nombre, _x, _reladj1),
```

```
nom(_rest, _resta, _genre, _nombre, _racine_nom), add(affirme(_x,être,_racine_nom,vrai), _reladj1, _rel1),
```

```
adjectif(_resta, _restb, _genre, _nombre, _x, _reladj2), add(_rel1, _reladj2, _rel2),
```

```
adjectif(_restb, _restc, _genre, _nombre, _x, _reladj3), add(_rel2, _reladj3, _rel3),
```

```
nrelative(_restc, _reste, _x, _racine_nom, _rel4), add(_rel3, _rel4, _rel).
```

```
groupe_nominal_sujet(_in, _reste, _x, gn(affirme(_x,être,_prénom,vrai), _genre, _nombre)) :-
```

```
prénom(_in, _reste, _prénom, _genre, _nombre).
```

```
groupe_nominal_sujet([_le | _in], _reste, _x, gn(affirme(_x,être,_nom,vrai), _genre, _nombre)) :-
```

```
article(_le, _genre, _nombre),
```

```
nom_propre(_in, _reste, _nom, _genre, _nombre).
```

```
groupe_nominal_sujet([_variable | _reste], _reste, _y, gn(affirme(_y,être,_variable,vrai), _genre, singulier)):-
```

```
variable(_variable, _variable).
```

```

adjectif([_adj|_reste],_reste,_genre,_nombre,_x,affirme(_x,avoir,_atr,_racadj,vrai)):-
    adjectif(_adj,_genre,_nombre,_racadj,_atr),!.
adjectif([non,_adj|_reste],_reste,_genre,_nombre,_x,affirme(_x,avoir,_atr,non(_racadj)),vrai)):-
    adjectif(_adj,_genre,_nombre,_racadj,_atr),!.
adjectif(_in,_in,_,_,_x,true).

nom(_in,_reste,_genre,_nombre,_racine_nom):-
    nom_composé(_in,_reste,_genre,_nombre,_racine_nom),!.
nom([_nom|_reste],_reste,_genre,_nombre,_racine_nom):-
    nom(_nom,_genre,_nombre,_racine_nom),nonnomde(_racine_nom).

add(_rel,true,_rel):-!.
add(_rel1,_rel2,et(_rel1,_rel2)).

nrelative(_rest,_reste,_x,_racine_nom,_rel):-
    relative(_rest,_res,_x,_racine_nom,_rel1),
    suite_relative(_res,_reste,_x,_racine_nom,_rel1,_rel).
nrelative(_reste,_reste,_x,_racine_nom,true).

suite_relative([et|_rest],_reste,_x,_racine_nom,_rel1,_rel):-
    relative(_rest,_res,_x,_racine_nom,_rel2),
    suite_relative(_res,_reste,_x,_racine_nom,et(_rel1,_rel2),_rel),!.
suite_relative(_reste,_reste,_x,_racine_nom,_rel,_rel).

nom_composé([_a,_b|_reste],_reste,_genre,singulier,_nom):-nom_composé([_a,_b],_,_genre,_nom).
nom_composé([_a,_b,_c|_reste],_reste,_genre,singulier,_nom):-nom_composé([_a,_b,_c],_,_genre,_nom).
nom_composé([_a,_b|_reste],_reste,_genre,pluriel,_nom):-nom_composé(,[_a,_b],_genre,_nom).
nom_composé([_a,_b,_c|_reste],_reste,_genre,pluriel,_nom):-nom_composé(,[_a,_b,_c],_genre,_nom).

nom_propre([_nom|_reste],_reste,_nom,_genre,singulier):-nom_propre_declaré(_nom,_genre).
nom_propre([_a,_b|_reste],_reste,_nom,_genre,_nombre):-nom_propre_declaré([_a,_b],_genre,_nombre,_nom).
nom_propre([_a,_b,_c|_reste],_reste,_nom,_genre,_nombre):-nom_propre_declaré([_a,_b,_c],_genre,_nombre,
_nom).

prénom([_nom|_reste],_reste,_nom,_,singulier):-prénom(_nom,_genre).
prénom([_a,_b|_reste],_reste,_nom,_,singulier):-prénom_declaré([_a,_b],_genre,_nom).
prénom([_a,_b,_c|_reste],_reste,_nom,_,singulier):-prénom_declaré([_a,_b,_c],_genre,_nom).

nom_composé([porc,épique],[porcs,épiques],masculin).

nom_propre(bretagne,féminin).

nom_propre_declaré([loire,atlantique],féminin,singulier,loire_atlantique))

```

6.7 Les substantifs construits avec de

le père d'Hector est Médor

père, se construit avec *de* et définit une relation précise entre Médor et Hector, qui s'enregistre en une seule règle :

```
affirme(_x,avoir,père,_y,vrai):-  
    affirme(_x,être,Hector,vrai),  
    affirme(_y,être,Médor,vrai).
```

mais en fait tous les attributs fonctionnent de cette manière, ainsi :

la taille d'Hector est petite

Hector a une petite taille

se traduisent en :

```
affirme(_x,avoir,taille,petit,vrai):-affirme(_x,être,Hector,vrai).
```

Et donc, c'est le moment de faire sauter le verrou qui disait que le sujet est toujours un nom d'entité en ajoutant simplement cette règle :

```
phrase([_le,_attribut,_de|_rest]):-  
    article(_le,_genre,_nombre),  
    nom(_attribut,_genre,_nombre,_atr),nomde(_atr,_,_),  
    de(_de),  
    groupe_nominal_sujet_joker(_rest,_res,_x,g(_racnom,_g,_n)),  
    forme_verbale_active(_res,_re,verbe(être,_vrai,_nombre)),  
    phrase_est(_re,_racadj),  
    traiter(phrase(_racnom,affirme(_x,avoir,_atr,_racadj,_vrai),true)).  
phrase_est([_adj,|.],_racadj):-  
    adjectif(_adj,_,_nombre,_racadj,_,!).  
phrase_est(_re,_prénom):-  
    prénom(_re,[|.],_prénom,_,_).
```

Pour les groupes nominaux suivants certains équivalents de « de » (du, des, au, aux), il n'y a pas d'article et il faut donc en rajouter un par :

```
groupe_nominal_sujet_joker(_in,_reste,_y,_gn):-groupe_nominal_sujet(_in,_reste,_y,_gn).  
groupe_nominal_sujet_joker(_in,_reste,_y,_gn):-groupe_nominal_sujet([joker|_in],_reste,_y,_gn).
```

```
article(joker,_any1,_any2).
```

« Joker » est un article fictif qui s'accordera donc en genre et nombre avec le nom suivant, quel qu'il soit.

6.8 Nouvelles phrases comprises par le système

Les nouvelles affirmations, et les nouvelles questions :

une autruche est un oiseau à grand cou	
une autruche ne vole pas	
un perroquet est un oiseau rare	
un perroquet vole	
quels sont les oiseaux qui ne volent pas	autruche
une girafe est un animal qui a quatre pattes	
une autruche est un animal à deux pattes	
une girafe est un mammifère dont le cou est grand	
quels sont les animaux ayant un grand cou	girafe, autruche
un bélier est un mouton de sexe mâle	
Fifine est une brebis	
une brebis est un mouton femelle	
est-ce que Fifine est un bélier	non
est ce que Fifine est un mouton non mâle	oui
un vautour mange la viande non fraîche	
est ce que le vautour mange la viande pourrie	oui
un mouton n a pas de griffes	
un félin est un animal qui a des griffes rétractiles	
un canidé a des griffes non rétractiles	
qui n a pas des griffes rétractiles	mouton, chien
qui n a pas des griffes non rétractiles	mouton, félin
qui a des griffes non rétractiles	chien
qui a des griffes rétractiles	félin
Médor est un grand chien	
Médor est gris	
Arthur est un petit chien	
Arthur est blanc	
quelle est la couleur du petit chien	blanc
un canidé a des grandes canines	
un chien est un canidé	
quelle est la taille des canines de Médor	grande
Médor a les canines noires	
comment sont les canines de Médor	grande, noir
Hector est un chien de couleur noire	
Hector est de grande taille	
Médor aime les grandes souris	
Titine est une souris qui a une petite taille	
Hector aime Titine	
quel est le chien qui aime une petite souris	Hector

saint brieuc est la capitale des côtes du nord
la loire atlantique a nantes pour capitale
la loire atlantique est un département de bretagne
la loire atlantique a une surface de 10000 km²
le morbihan est un département de bretagne
la surface du morbihan est de 15000 km²
quelle est la surface moyenne des départements de bretagne

les canidés sont des mammifères terrestres carnivores qui ont des griffes non rétractiles et qui ont des oreilles triangulaires.

le rhinocéros est un ongulé qui a 1 défense et qui a 3 doigts par pied.

Le porc a une longévité de 10 ans.

La longévité de la vache est de 9 ans.

Quel est l'animal qui a la plus grande longévité

Quels sont les animaux qui ont une longévité supérieure à celle de la vache

7 Analyse des règles en si ... alors

L'affirmation, déjà traitée :

celui qui Verbe1 GN1 verbe2 GN2

a le sens de

si X Verbe1 GN1 alors X Verbe2 GN2

Cela peut se généraliser en

si GNS1 Verbe1 GNC1 et/ou GNS2 Verbe2 GNC2 alors GNS3 Verbe3 GNC3

et son équivalent négatif, pour simplifier, dans le cas d'une condition unique :

si GS3 ne verbe3 pas GNC3 alors GNS1 ne verbe1 pas GNC1

les variables x, y, z et t pouvant prendre la place de n'importe quel groupes nominal.

7.1 Les relations de parenté

Cela permet alors d'apprendre au système les relations de parenté :

si x est le père de y alors x est le parent de y.

si x est la mère de y alors x est le parent de y.

si x est le parent de y alors y est l'enfant de x.

si x est le parent de y et si y est masculin alors y est le fils de x.

si x est le parent de y et si y est féminin alors y est la fille de x.

si x est le parent de y et si y est le parent de z alors x est le grandparent de z.

si x est le grandparent de y et si x est masculin alors x est le grandpère de y.

si x est le grandparent de y et si x est féminin alors x est la grandmère de y.

si x est le grandparent de y alors y est le petit-enfant de x.

si x est le grandparent de y et si y est masculin alors y est le petit-fils de x.

si x est le grandparent de y et si y est féminin alors y est la petite-fille de x.

si x est le parent de y et si x est le parent de z et si y n'est pas z alors y est le frère/soeur de z.

si x est le frère/soeur de y et si x est masculin alors x est le frère de y.

si x est le frère/soeur de y et si x est féminin alors x est la soeur de y.

si x est le parent de y et si z est le frère/soeur de x alors z est l'oncle/tante de y.

si x est l'époux de y ou si x est l'épouse de y alors x est l'époux/épouse de y.

si x est l'époux de y alors y est l'épouse de x.

% attention la règle si x est l'épouse de y alors y est l'époux de x donne une boucle infinie

% donc il faut déclarer les époux et pas les épouses

si x est le parent de y et si z est le frère/soeur de x et si t est l'époux/épouse de z alors t est l'oncle/tante de y.

si x est l'oncle/tante de y et si x est masculin alors x est l'oncle de y.

si x est l'oncle/tante de y et si x est féminin alors x est la tante de y.

si x est l'oncle/tante de y alors y est le neveu/niece de x.

si x est l'oncle/tante de y et si y est masculin alors y est le neveu de x.

si x est l'oncle/tante de y et si y est féminin alors y est la niece de x.

si x est le parent de y et si z est l'époux de y alors x est le beau-parent de z.

si x est le parent de y et si z est l'épouse de y alors x est le beau-parent de z.

si x est le beau-parent de y et si x est masculin alors x est le beau-père de y.
 si x est le beau-parent de y et si x est féminin alors x est la belle-mère de y.
 si x est le beau-parent de y alors y est le gendre/brue de x.
 si x est le beau-parent de y et si y est masculin alors y est le gendre de x.
 si x est le beau-parent de y et si y est féminin alors y est la brue de x.
 si y est le parent de x et si z est l'onclet/ante de x et si z est le parent de t alors x est le cousin/cousine de t.
 si x est le cousin/cousine de y et si x est masculin alors x est le cousin de y.
 si x est le cousin/cousine de y et si x est féminin alors x est la cousine de y.

7.2 Transformation des noms des variables en variables prolog

L'analyse étant terminée, le résultat contient des x, y, z ou t en place de certains groupes nominaux et il suffit de remplacer tous les x par un même `_a`, tous les y par un même `_b`, ...
 Cela se fait en parcourant récursivement la structure résultat (chaque élément des *et* et des *ou* des formules logiques). Passons sur cette tambouille.

8 Les hyper règles

Ce sont des règles générales, que l'on déclare dans le programme et non sous forme d'affirmations interactives. Elles donnent leur sens à la copule *non*, et à quelques synonymes et contraires.

8.1 La copule non et les verbes être et avoir

si

une brebis a le sexe femelle est vrai

alors

une brebis a le sexe mâle est faux

il faut donc ajouter au programme les règles générales suivantes :

affirme (_x, avoir, _attribut, _yy, faux) :- contraire(_yy, _y), affirme (_x, avoir, _attribut, _y, vrai).

affirme (_x, avoir, _attribut, _yy, vrai) :- contraire(_yy, _y), affirme (_x, avoir, _attribut, _y, faux).

De même si

un chat a les griffes rétractiles est vrai

alors

un chat a les griffes non rétractiles est faux

par contre avec un autre verbe que *avoir* , si

si *un chat mange les souris femelles* est vrai

un chat mange les souris non femelles peut être également vrai

on ne peut donc rien en déduire d'autre.

8.2 Les contraires et le verbe être

Il est nécessaire de formaliser dans le dictionnaire la relation entre *mâle* et *femelle*, *pourri* et *frais*, ... sans avoir pour chaque couple à déclarer :

celui qui est mâle est non femelle

ni *celui qui est non mâle est femelle*

Pour cela, il suffit de déclarer une fois pour toutes dans le programme :

affirme(_x, être, non(_w), vrai) :-

 contraire(_z, _w),

 affirme(_x, être, _z, vrai).

affirme(_x, être, _w, vrai) :-

 contraire(_z, _w),

 affirme(_x, être, non(_z), vrai).

8.3 Les contraires et les autres verbes

De même on évite d'avoir à déclarer pour chaque verbe :

celui qui mange la viande pourrie mange la viande non fraîche

en ajoutant :

affirme (_x, _verbe, _nom, non(_w), vrai) :-

 contraire(_z, _w),

 affirme (_x, _verbe, _nom, _z, vrai).

affirme(_x, _verbe, _nom, _w, vrai) :-

 contraire(_z, _w),

 affirme (_x, _verbe, _nom, non(_z), vrai).

contraire(mâle, femelle).

contraire(femelle, mâle).

contraire(frais, pourri).

contraire(pourri, frais).

Ces 4 types d'hyper règles, du fait de la réciprocité qu'elles entraînent, renforcent la nécessité d'un contrôle anti boucle efficace.

9 Les paradoxes logiques

On l'a vu, notre programme utilise abondamment le raisonnement par l'absurde, qui dit que

Si P implique Q (si P est vrai alors Q est vrai)

Alors non Q implique non P (si Q est faux alors P est faux)

Voyons comment il réagit à ces deux affirmations du bon sens commun :

ce qui est rare est cher

ce qui n'est pas cher est rare

De la première affirmation (si X est rare, alors X est cher), on peut conclure, par le fameux raisonnement par l'absurde, que *ce qui n'est pas cher n'est pas rare (si X n'est pas cher, alors X n'est pas rare)*, ce qui est incompatible avec la seconde affirmation, qui dit exactement le contraire !

C'est tout simplement que l'une des deux affirmations est fausse, ou n'a pas de sens, ou n'a un certain sens que dans un certain contexte, à un certain moment. En fait quelque chose n'est pas simplement *cher* mais est *plus cher que*, tout est relatif disait Einstein !

Le programme ne détecte que la cohérence des faits, pas des règles, et donc il ne détecte pas cette incohérence et enregistre les deux affirmations.

Soit à présent un barbier de village qui rase tous ceux qui ne se rasent pas eux-mêmes. Se rase-t-il ? Notre programme comprend :

si X ne rase pas X alors Barbier rase X

Ce qui s'enregistre en :

affirme(Barbier, rase, _x, vrai) :- affirme(_x, rase, _x, faux).

affirme(_x, rase, _x, vrai) :- affirme(Barbier, rase, _x, faux).

fort justement, si on pose la question

est ce que Barbier rase Barbier

il répond : *je ne sais pas*

Et là encore le programme est pris en défaut car on peut affirmer indifféremment

Barbier rase Barbier

ou *Barbier ne rase pas Barbier*

Pour ces deux paradoxes, l'incohérence n'apparaît qu'en réponse à *qui fait quoi*, à quoi on obtient :

cher est rare

cher n'est pas rare

et

Barbier rase Barbier

Barbier ne rase pas Barbier

Le programme final, (celui ci est en gnu prolog), résultat du chapitre précédent est capable de lire un fichier texte décrivant les animaux, un autre décrivant les départements et régions de France, et un dernier décrivant les relations familiales.

Pour lire ces 3 fichiers, en particulier celui sur la classification des animaux, le programme est amené à comprendre des affirmations du type :

groupe-nominal-sujet verbe ex: un poisson nage
groupe-nominal-sujet verbe groupe-nominal-complément ex :
un félin a des griffes rétractiles
groupe-nominal-sujet est verbe par groupe-nominal-sujet ex :
la souris est mangée par le chat
seul groupe-nominal-sujet verbe groupe-nominal-complément ex:
seul un vertébré a une colonne vertébrale
le attribut de groupe-nominal-sujet est quantité-relative
une règle ex :
si x est le père de y et si y est de sexe masculin alors y est le fils de x
celui qui verbe groupe-nominal-complément verbe groupe-nominal-complément

Concernant le groupe nominal, sujet ou complément, on distingue

nom-d-entité, servant à nommer les entités à étudier :
chat, poisson, merlan, porc épig
nom-d-attribut, servant à décrire les caractéristiques des entités :
température, taille, poids, couleur

les parties des entités et tout ce qui sert à les décrire sont également des attributs, que les entités ont, ou pas, et de telle et telle façon :

patte, antenne, queue, nid

le système est capable de répondre à des questions

quel est le attribut des groupe-nominal-sujet ex :
quelle est la couleur du cheval blanc d henri IV
est ce que phrase
qui verbe groupe-nominal-complément
que verbe groupe-nominal-sujet
quels sont les groupe-nominal-sujet
quels sont les attributs respectifs des groupe-nominal-sujet
quels sont classés par attribut les groupe-nominal-sujet
il y a combien de groupe-nominal-sujet

un groupe-nominal-sujet (gns) c'est :

(adjectif) nom-d-entité (non) (adjectif) (adjectif) (relatives) ex :
un petit animal noir qui mange une souris et qui a des griffes non rétractiles
article plus/moins adjectif des groupe-nominal-sujet ex :
le plus lourd des chats qui ont la couleur noire
prénom ex: hector/médor/titine
article nom-propre ex : la loire atlantique
x/y/z, servant à définir des règles ex : si x est le père de y alors y est l enfant de x

un verbe (être, avoir ou autre) prend la forme :

verbe

ne verbe pas

est verbé par

n est pas verbé par

groupe-nominal-sujet (gns) ex:

un petit animal insectivore qui a des plumes et dont le bec est long

groupe-nominal-complément

adverbe (si verbe non être/avoir) ex : nage vite

(préposition) article (adjectif) nom-d-entité (adjectif) (si verbe non être/avoir) ex:

vit sur la terre

article nom-d-attribut préposition non-d-attribut (si verbe non être) ex :

a des ongles en corne

construit des nids dans les arbres

compléments spécifiques au verbe être

(non) adjectif ex : est petit

de attribut quantité-relative ex :

est de poids supérieur/égal à 2 kg

de adjectif attribut ex :

de taille petite

de petite taille

le attribut de groupe-nominal-sujet ex :

le fils d hector

compléments spécifiques au verbe avoir :

(adjectif) nom-d-attribut (adjectif) (adjectif) ex : a un long bec crochu

nombre nom-d-attribut ex: a 4 pattes

de article nom-d-attribut ex : a de la fourrure

de adjectif nom-d-attribut ex : a de longues pattes

pour attribut gns ex : a pour fils hector

gns pour attribut ex : a quimper pour capitale

quantité : ex a 2 kg

article attribut supérieur/inférieur/égale à quantité-relative ex :

a un poids supérieur/égal à 2 kg

plus/moins/autant de attribut que gns ex :

a plus de poids que médor

le groupe nominal peut avoir une relative complexe :

dont article attribut est/n est pas adjectif ex:

dont la taille est petite

dont article attribut est/n est pas quantité-relative ex:

dont la taille est supérieur à celle du chien

dont article attribut est groupe-nominal-sujet ex

dont le fils est hector

dont groupe-nominal-sujet est/n est pas article attribut ex :

dont quimper est la capitale

qui verbe groupe-nominal-complément ex : qui mange de l herbe

participe-présent groupe-nominal-complément ex : mangeant la souris
à attribut ex : à poils
à nombre attribut ex : à 4 pattes
de adjectif attribut ex : de petite taille
de groupe-nominal-sujet ex : le chat de hector
de quantité ex : de 2 kg
de attribut quantité-relative ex : de poids supérieur à celui du chien
que verbe groupe-nominal-sujet ex : que mange le chat

une valeur d'attribut (gn-valeur) :

article moyenne/somme/minimum/maximum des attributs de groupe-nominal-sujet ex
la somme du poids des chats
article attribut moyenne/somme/minimum/maximum des groupe-nominal-sujet ex :
le poids moyen des chats
article attribut du plus/moins adjectif des groupe-nominal-sujet ex :
le poids du plus lourd des chats
le nombre de groupes-nominal-sujet ex :
le nombre de chats rouges qui volent et qui nagent
la somme des surfaces des départements de bretagne
comparatif à celle de groupe-nominal-sujet ex :
supérieur/égal à celui du chat

une valeur numérique, avec ou sans unité:

nombre unités (de attribut) ex : 2 m de haut
valeur : 4 pattes
comparatif (de) valeur ex :
plus de 3 kg
supérieur à 3 kg

on lance le programme par :

`gprolog`
`consult('coco.pl').`

il attend alors des phrases comme celles-ci-dessus, terminées par un .

il y a de base un dictionnaire contenant quelques noms propres d'où :

`grosminet est un chat.`
`les chats sont des félidés.`
`les félidés ont des griffes rétractiles.`
`Les félidés sont des mammifères.`
`Les mammifères produisent du lait.`
`les félidés ont 4 pattes.`
`hector n a pas de griffes.`
`qu a grosminet.`
`griffes,pattes`
`qu est grosminet.`
`chat,félidé,mammifère,grosminet`
`est ce que hector est un chat.`
`non`
`est ce que hector est un félidé.`
`Non`

que fait grosminet.

être féliné, être chat, être mammifère, avoir griffe, avoir patte, produire lait
est ce que hector produit du lait

coco ne sait pas

hector est un mammifère

qui produit du lait

chat, féliné, mammifère, grosminet, hector

Le fichier animal.txt contient des définitions d'animaux ou d'espèces, introduits par « seuls », sous entendant comme on l'a vu la réciproque : tel animal a telle propriété et si un animal a cette propriété, c'est donc tel animal. Puis des propriétés annexes, sans réciproque. Chaque affirmation se termine par un point.

On le lit par la commande :

fichier animal.

Le résultat de l'analyse et les règles enregistrées se trouvent dans animal.log.

seuls les animaux qui mangent de la viande et qui mangent des plantes et qui mangent des poissons sont omnivores.

seuls les animaux qui mangent de la viande sont carnivores.

seuls les animaux qui mangent des plantes sont herbivores.

seuls les animaux qui mangent des détritiques sont détritiques.

seuls les animaux qui mangent la charogne sont nécrophages.

seuls les animaux qui mangent des graines sont granivores.

seuls les animaux qui mangent des poissons sont piscivores.

seuls les animaux qui mangent des insectes sont insectivores.

seul un animal qui vit sur terre est terrestre.

seul un animal qui vit dans la mer est marin.

un animal qui respire dans l'air a des poumons.

un animal qui respire dans l'eau a des branchies.

un animal qui a des nageoires vit dans l'eau.

un animal qui a des ailes vit dans l'air.

un animal qui a un poumon et qui a des pattes vit sur terre.

%un animal qui a des branchies vit dans l'eau.

les animaux qui ont des pattes non symétriques sautent.

seul un animal qui pond des oeufs est ovipare.

seul un animal qui ne pond pas des oeufs est vivipare.

un animal qui ne pond pas des oeufs n'est pas un poisson.

un animal qui ne pond pas des oeufs n'est pas un oiseau.

celui qui a des poils n'a pas d'écailles.

celui qui a des poils n'a pas des plumes.

celui qui a des écailles n'a pas de poils.

celui qui a des écailles n'a pas des plumes.

celui qui a des plumes n'a pas des poils.

celui qui a des plumes n'a pas d'écailles.

seul un animal qui a une colonne vertébrale est un vertébré.
seul un animal qui n'a pas de colonne vertébrale est un invertébré.
seul un vertébré qui a des poumons est un tétrapode.

seul un tétrapode vivipare est un mammifère.
seuls les tétrapodes ovipares qui ont 2 ailes et qui ont un bec en corne et qui ont des plumes et qui ont 2 pattes sont des oiseaux.
seuls les tétrapodes ovipares qui ont des écailles soudées sont des reptiles.
seuls les tétrapodes ovipares qui ont la peau nue sont des amphibiens.

seul un vertébré ovipare qui a des branchies et qui a des nageoires est un poisson.
seuls les vertébrés ovipares qui ont des branchies et qui n'ont pas de nageoires sont des agnathes.

un poisson a des écailles libres.
un poisson a une température variable.
un poisson pond dans l'eau.

seuls les poissons cartilagineux sont des chondrichthyens.
le requin est un chondrichthye.
la raie est un chondrichthye.
la raie a un dard venimeux.
la roussette est un chondrichthye.

seuls les poissons osseux sont des ostéichyens.
les ostéichyens ont une vessie natatoire.
l'esturgeon est un ostéichyen.
l'esturgeon se reproduit en rivière.
l'esturgeon migre.
la morue est un ostéichyen.
la lotte est un ostéichyen.
le merlan est un ostéichyen.
le lieu est un ostéichyen.
le haddock est un ostéichyen.
le gardon est un ostéichyen.

seuls les chondrichthyens qui ont un long corps cylindrique sont des élopomorphes.
l'anguille est un élopomorphe.
l'anguille se reproduit en mer.
l'anguille migre.
le congre est un élopomorphe.
la murène est un élopomorphe.
la murène a une morsure venimeuse.

seuls les ostéichyens qui ont 1 grande nageoire dorsale et qui ont 1 petite nageoire dorsale sont des salmoniformes.
le saumon est un salmoniforme.
l'éperlan est un salmoniforme.
le brochet est un salmoniforme.

la truite est un salmoniforme.
le salmoniforme se reproduit en rivière.
le salmoniforme migre.
le salmoniforme saute.

seuls les ostéichiens qui ont 1 grande nageoire dorsale et qui n ont pas 1 petite nageoire dorsale sont des ostariophysaires.
la carpe est un ostariophysaire.
le pinranha est un ostariophysaire.

seuls les ostéichiens qui ont 2 grandes nageoires dorsales sont des percomorphes.
la dorade est un percomorphe.
la rascasse est un percomorphe.
la rascasse a des épines venimeuses.
le grondin est un percomorphe.
la perche est un percomorphe.
le rouget est un percomorphe.
le mulot est un percomorphe.
le capitaine est un percomorphe.
la gobie est un percomorphe.
la gobie a des cellules urticantes.
la vive est un percomorphe.
la vive a des épines venimeuses.
le thon est un percomorphe.
le maquereau est un percomorphe.
le turbot est un percomorphe.
le flétan est un percomorphe.
la limande est un percomorphe.
la sole est un percomorphe.
l hippocampe est un ostéichien.

%seul un vertébré qui a des poumons est un tétrapode.
un tétrapode a 4 membres.

%seul un tétrapode vivipare est un mammifère.
celui qui allaite est un mammifère.
un mammifère allaite.
un mammifère a des mamelles.
un mammifère a une température constante.
celui qui produit du lait est un mammifère.
un mammifère produit du lait.

un mammifère terrestre a des poils.
un mammifère terrestre a des oreilles.
un mammifère terrestre a 4 pattes.
les mammifères terrestres ont une queue.
les mammifères qui ont des longues pattes courent vite.

seuls les mammifères terrestres carnivores qui ont des griffes non rétractiles et qui ont des oreilles triangulaires sont des canidés.

les canidés ont des canines longues.
les canidés ont un museau long.
les canidés ont une truffe sur le museau.
les canidés sautent.

le loup est un canidé.
le chacal est un canidé.
le renard est un canidé.
les chiens sont des canidés.

le lévrier est un chien.

seuls les mammifères terrestres carnivores qui ont des griffes non rétractiles et qui ont des oreilles rondes sont des ursidés.
les ours sont des ursidés.
l ours blanc vit dans les pays froids.
l ours brun ne vit pas dans les pays froids.
l ours hiberne.
le panda est un ursidé.
les ursidés marchent l amble.

seuls les mammifères terrestres carnivores qui ont des griffes rétractiles sont des félidés.
%le critère griffes rétractiles n est pas bon, le guépard n en a pas ?
%en quoi un félifforme se distingue des autres carnivores ?

les félidés sont des félifformes.
%en quoi les félidés se distinguent des autres félifformes ?
les félidés sautent.

le tigre est un félidé.
le chat est un félidé.
le lion est un félidé.
le léopard est un félidé.
le jaguar est un félidé.
le lynx est un félidé.
la panthère est un félidé.
le guépard est un félidé.
un guépard a des griffes non rétractiles.

la hyène est un félidé.
% en quoi la hyène se distingue des autres félifformes ?
la hyène femelle est dominante.

la mangouste est un félidé.
% en quoi la mangouste se distingue des autres félifformes ?

seuls les petits mammifères terrestres carnivores qui ont un long corps et qui ont des pattes courtes sont des mustélidés.
les mustélidés ont de la fourrure.
les mustélidés dégagent une odeur musquée.

la belette est un mustélidé.
la martre est un mustélidé.
le blaireau est un mustélidé.
la loutre est un mustélidé.
l hermine est un mustélidé blanc.
le furet est un mustélidé.
le vison est un mustélidé.
la fouine est un mustélidé.
le putois est un mustélidé.

seuls les mammifères terrestres herbivores qui ont une poche ventrale et qui ont des pattes non symétriques et qui ont une grosse queue sont des marsupiaux.
l opossum est un marsupial.
le kangourou est un marsupial.
le koala est un marsupial.
les wallabies sont des marsupiaux.

seuls les mammifères terrestres herbivores qui ont des sabots sont des ongulés.
les ongulés sont herbivores.

seuls les ongulés qui ont un groin et qui ont 2 doigts par pied sont des suinés.
le porc est un suiné.
le cochon est un porc.
le sanglier est un suiné.
le pécari est un suiné.
les porcs ont une queue en spirale.
le phacochère est un suiné.
le phacochère a 4 défenses.

seul un ongulé qui a 1 défense et qui a 3 doigts par pied est un rhinocéros.
la défense du rhinocéros est en ivoire.

seul un ongulé qui a une courte trompe et qui a 3 doigts par pied est un tapir.

seuls les ongulés qui ont deux estomacs et qui ont 2 doigts par pied sont des ruminants.

seuls les ruminants qui ont des cornes en os sont des bovidés.
les vaches sont des bovidés.
les zébus sont des bovidés.
les bisons sont des bovidés.
les antilopes sont des bovidés.
l antilope saute.
les gazelles sont des bovidés.
les gazelles courent vite.
les gazelles sautent.
les buffles sont des bovidés.
les gnous sont des bovidés.

seuls les bovidés qui vivent en montagne sont des caprinés.

les moutons sont des caprinés.
la chèvre est un capriné.
le bouquetin est un capriné.
le chamois est un capriné.
le mouflon est un capriné.
le chamois saute.
le bouquetin saute.

seuls les ruminants qui ont des bois sont des cervidés.
un animal qui a des cornes ramifiées a des bois.
les chevreuils sont des cervidés.
les cerfs sont des cervidés.
les daims sont des cervidés.
les élans sont des cervidés.
les rennes sont des cervidés.
les rennes vivent dans les pays froids.
les élans vivent dans les pays froids.

seuls les ruminants qui ont un long cou et qui ont des cornes sont des girafidés.
les girafes sont des girafidés.
seul un girafidé qui a une langue préhensible est un okapi.
les girafidés marchent l amble.

seuls les ongulés qui ont une grande machoire et qui ont un gros corps et qui ont 4 doigts par pied sont des hippopotames.

seuls les ongulés qui ont un long cou recourbé sont des camélidés.
seuls les camélidés qui ont 2 bosses dorsales sont des chameaux.
seuls les camélidés qui ont 1 bosse dorsale sont des dromadaires.
un camélidé qui n a pas de bosse est un lama.
le lama crache.
les camélidés marchent l amble.

seuls les ongulés qui ont une trompe préhensible et qui ont 2 défenses et qui ont de grandes oreilles sont des éléphants.
les défenses de l éléphant sont en ivoire.
les éléphants ont 5 doigts par pied.
les éléphants marchent l amble.

seuls les ongulés qui ont 1 doigt par pied sont des équidés.
l âne est un équidé.
le cheval est un équidé.
le poney est un équidé.
les chevaux ont une longue queue.
le cheval saute.
le zèbre est un équidé.

seuls les petits mammifères terrestres insectivores qui ne volent pas sont des eulypotyplas.
la musaraigne est un eulypotypla.
la taupe est un eulypotypla.

le hérisson est un eulypotypla.
le hérisson a des épines.

seuls les petits mammifères terrestres insectivores qui volent sont des chiroptères.
les mammifères qui ne sont pas des chiroptères ne volent pas.
le chiroptère a des membres palmés.
le chiroptère se suspend par les pieds.
la chauve souris hiberne.

seuls les mammifères terrestres herbivores qui ont des pattes non symétriques et qui ont 4 incisives sont des lagomorphes.
les lagomorphes ont une petite queue.
les lagomorphes ont des longues oreilles.
le lapin est un lagomorphe.
le lièvre est un lagomorphe.
les lagomorphes ont de la fourrure.

seuls les petits mammifères terrestres herbivores qui ont 2 incisives plates et qui n ont pas de canines sont des rongeurs.
les rongeurs ont de la fourrure.
les écureuils ont une longue queue.
l écureuil hiberne.
les castors ont une longue queue.
les castors ont une queue plate.
la marmotte est un rongeur.
la marmotte hiberne.
le loir est un rongeur.
le loir hiberne.
le rat est un rongeur.
la souris est un rongeur.
le mulot est un rongeur.
le ragondin est un rongeur.
le hamster est un rongeur.
le porc épic est un rongeur.
le cochon d inde est un rongeur.
le cobaye est un rongeur.
le chinchilla est un rongeur.

seuls les mammifères terrestres qui ont des mains préhensibles et qui ont des ongles plats sont des primates.
les primates vivent en groupe.
les primates sautent.

seuls les primates qui ont les pieds préhensibles sont des singes.
l homme est un primate.

le chimpanzé est un singe.
le babouin est un singe.
le gorille est un grand singe.
le lémurien est un singe.

l orang outan est un singe.
le paresseux est un singe.

un mammifère marin a des nageoires.
un mammifère marin nage.
seuls les mammifères marins piscivores qui ne rampent pas sur terre sont des cétacés.
un cétacé qui a 1 défense est un narval.
les cétacés qui n ont pas de dents et qui ont des fanons sont des baleines.
le rorqual est une baleine.
les dauphins sont des cétacés qui ont des dents.
le dauphin saute.
les cachalots sont des cétacés qui ont des dents.
les marsouins sont des cétacés qui ont des dents.

seuls les mammifères marins piscivores qui rampent sur terre sont des phocidés.
les phocidés vivent dans les pays froids.
les phoques sont des phocidés.
les morses sont des phocidés.
le morse a de longues canines.
les otaries sont des phocidés.

seuls les mammifères marins herbivores sont des siréniens.
le dugong est un sirénien.
le lamentin est un sirénien.

%seuls les tétrapodes ovipares qui ont 2 ailes et qui ont un bec en corne et qui ont des plumes
et qui ont 2 pattes sont des oiseaux.
les oiseaux ont des doigts en corne.
les oiseaux ont une température constante.
%les oiseaux mangent les insectes.
les oiseaux mangent des graines.
les oiseaux construisent des nids.

seuls les oiseaux terrestres qui ne volent pas sont des ratites.
l autruche est un ratite.
le kiwi est un ratite insectivore nocturne.
le kiwi a un long bec.
l émeu est un ratite.
le casoar est un ratite.

seuls les oiseaux marins qui ne volent pas et qui nagent sont des sphénisciformes.
les manchots sont des sphénisciformes.
les manchots vivent dans les pays froids.

seuls les oiseaux marins qui vivent dans les pays froids et qui volent et qui nagent sont des alcidés.
le pingouin est un alcidé.
seul un alcidé qui a un bec triangulaire rouge et qui a les pattes palmées est un macareux.

les oiseaux terrestres qui ne sont pas des ratites volent.

les oiseaux marins qui ne sont pas des sphénisciformes volent.

seuls les oiseaux marins qui ont de grandes ailes et qui n'ont pas les pattes palmées sont des neognathaes.

les neognathaes ont le bec jaune.

les albatros sont des neognathaes.

les pétrels sont des neognathaes.

seuls les oiseaux marins omnivores qui ont de grandes ailes et qui ont des pattes palmées sont des laridés.

le goéland est un laridé blanc qui a un bec jaune.

la mouette est un laridé blanc qui a un bec jaune.

seuls les oiseaux marins noirs qui ont un bec crochu et qui ont des longues ailes sont des cormorans.

seuls les oiseaux terrestres qui ont des longues pattes et qui ont un long cou et qui ont un long bec sont des échassiers.

les échassiers migrent.

les échassiers vivent dans des lieux humides.

seuls les échassiers qui ont un bec conique sont des ardéidés.

le héron est un ardéidé.

l'aigrette est un ardéidé blanc.

le butor est un ardéidé.

seul un échassier qui a un bec recourbé est un ibis.

seuls les échassiers qui ont le bec droit sont des ciconiidés.

la cigogne est un ciconiidé.

seuls les échassiers qui ont un bec recourbé et qui ont un bec lamellé sont des flamants.

seuls les oiseaux terrestres qui ont des longues pattes et qui ont un long cou et qui ont un bec court sont des gruiformes.

la grue est un gruiforme.

seuls les oiseaux terrestres qui ont des pattes palmées et qui ont un bec lamellé sont des anatidés.

% c'est quoi un bec lamellé ?

les anatidés ont de petites pattes.

les anatidés vivent dans les lieux humides.

les anatidés nagent en surface.

le cygne est un anatidé blanc qui a un long cou.

l'oie est un anatidé blanc.

le canard est un anatidé.

la sarcelle est un anatidé.

seul un oiseau terrestre qui a un bec crochu et qui a des serres est un rapace.

seuls les rapaces diurnes nécrophages sont des falconidés.

le faucon est un falconidé.
les vautours sont des falconidés.
le condor est un falconidé.

seuls les rapaces diurnes carnivores sont des accipitridés.
l aigle est un accipitridé.
la buse est un accipitridé.
l épervier est un accipitridé.
le milan est un accipitridé.

seuls les rapaces nocturnes carnivores sont des strigidés.
la chouette est un strigidé.
le hibou est un strigidé qui a des oreilles.
l effraie est un strigidé.

seuls les oiseaux terrestres qui ont des ailes courtes et qui ont un bec court sont des galliformes.
la poule est un galliforme.
la gélinotte est un galliforme.
le faisan est un galliforme.
le coq de bruyère est un galliforme.
la perdrix est un galliforme.
la caille est un galliforme.
le paon est un galliforme.
la pintade est un galliforme.
le dindon est un galliforme.

seul un oiseau marin piscivore qui a un long bec et qui a une poche au bec est un pélican.

l outarde est un oiseau terrestre qui a de longues pattes.
% outarde mal définie ?
la bécasse est un oiseau terrestre qui a un long bec.
%bécasse mal définie ?
le courlis est un oiseau terrestre qui a un long bec recourbé.
% courlis mal défini ?

seuls les oiseaux terrestres qui ont un bec court et qui ont de petites pattes sont des columbidés.
la colombe est un columbidé.
le pigeon est un columbidé.
la tourterelle est un columbidé.

seuls les oiseaux terrestres qui ont une longue queue sont des cuculiformes.
%critère longue queue est insuffisant ?
le coucou est un cuculiforme.

seuls les oiseaux terrestres qui ont le bec crochu et qui sont colorés sont des psittaciformes.
le perroquet est un psittaciforme.
le cacatoès est un perroquet.
l aras est un grand perroquet.

la perruche est un petit psittaciforme.

seuls les oiseaux terrestres omnivores sont des corvidés.

% critère insuffisant pour les corvidés ?

le corbeau est un corvidé noir.

la corneille est un corvidé noir.

le choucas est un corvidé.

la pie est un corvidé.

le geai est un corvidé.

seuls les petits oiseaux terrestres qui ont des pattes courtes sont des micropodiformes.

le martinet est un micropodiforme.

le colibri est un micropodiforme qui a un long bec.

seuls les petits oiseaux terrestres qui chantent et qui ont 4 doigts par pied sont des passereaux.

les turridés sont des passereaux.

% en quoi les turridés se distinguent des autres passereaux ?

le merle est un turridé.

la grive est un turridé.

le rossignol est un turridé.

le rouge gorge est un turridé.

les tanagridés sont des passereaux.

% en quoi les tanagridés se distinguent des autres passereaux ?

le pinson est un tanagridé.

le serein est un tanagridé.

le chardonneret est un tanagridé.

le bouvreuil est un tanagridé.

l ortolan est un tanagridé.

la mésange est un passereau.

la fauvette est un passereau.

le roitelet est un passereau.

la bergeronnette est un passereau.

l étourneau est un passereau.

le moineau est un passereau.

le martin pêcheur est un passereau.

le loriot est un passereau.

l hirondelle est un passereau.

le mainate est un passereau.

l alouette est un passereau.

le canari est un passereau.

%seuls les tétrapodes ovipares qui ont des écailles soudées sont des reptiles.

% est ce qu'ils ont un poumon ?

les reptiles ont une température variable.

les reptiles ne volent pas.

seuls les reptiles qui ont 4 pattes et qui n'ont pas de carapace sont des crocodiliens.

% crocodilien = saurien ?

les crocodiliens ont des pattes courtes.
les crocodiliens ont une longue gueule.
les crocodiliens ont une énorme machoire.
les crocodiliens ont un corps long.
les crocodiles sont des crocodiliens.
les alligators sont des crocodiliens.
les caïmans sont des crocodiliens.

seuls les petits crocodiliens sont des lacertiliens.
le lézard est un lacertilien.
le lézard régénère sa queue.
l iguane est un lacertilien.
le caméléon est un lacertilien.
le caméléon a une longue langue.
le varan est un lacertilien.

seuls les reptiles qui n ont pas de pattes sont des serpents.
la vipère est un serpent.
la vipère a une morsure venimeuse.
la couleuvre est un serpent.
le boa est un serpent.
le naja est un serpent qui a une morsure venimeuse.
le cobra est un serpent qui a une morsure venimeuse.
l orvet est un serpent.

seul un reptile qui a une carapace est une tortue.
la tortue a 4 pattes.
la tortue est lente.

%seuls les tétrapodes ovipares qui ont la peau nue sont des amphibiens.
les amphibiens vivent dans des lieux humides.
les amphibiens ont la peau humide.
les amphibiens ont une température variable.
les amphibiens pondent dans l eau.
les amphibiens ne couvent pas.
les amphibiens ont des têtards.
les têtards ont des branchies.
les têtards se métamorphosent.

seuls les amphibiens qui n ont pas de pattes sont des apodes.

seuls les amphibiens qui ont des pattes longues et qui ont 4 pattes sont des anoures.
les anoures ont des pattes non symétriques.
les grenouilles sont des anoures.
le crapaud est un anoure.
la rainette est un anoure.

seuls les amphibiens qui ont des pattes courtes et qui ont un long corps et qui ont 4 pattes sont des urodèles.
le triton est un urodèle.

la salamandre est un urodèle.

%seuls les vertébrés ovipares qui ont des branchies et qui n'ont pas de nageoires sont des agnathes.

un agnathe a des écailles libres.

un agnathe a une température variable.

un agnathe pond dans l'eau.

la lamproie est un agnathe.

%seul un animal qui n'a pas de colonne vertébrale est un invertébré.

les invertébrés sont ovipares.

seuls les invertébrés qui ont des pattes sont des arthropodes.

seuls les arthropodes qui n'ont pas d'ailes et qui ont 8 pattes et qui n'ont pas d'antennes sont des arachnides.

les arachnides ont deux parties.

les arachnides ont des crochets venimeux.

seuls les arachnides qui produisent du fil sont des araignées.

les araignées construisent des toiles.

les araignées ont une morsure venimeuse.

la tarentule est une araignée.

les scorpions sont des arachnides.

les scorpions ont un dard venimeux.

les acariens sont des arachnides.

les tiques sont des arachnides.

seuls les arthropodes qui ont 6 pattes et qui ont 2 antennes sont des insectes.

%les insectes ont 4 ailes ou 2 et 2 atrophiées ou 0

les insectes ont la peau dure.

les insectes ont 3 parties.

%tête thorax et abdomen

les insectes qui ne sont pas les dictyoptères volent.

%quels sont les insectes qui ne volent pas ?

les termites sont des insectes.

%termite mal défini ?

les termites vivent en groupe.

le termite construit des termitières.

seuls les insectes détritivores sont des dictyoptères.

les cafards sont des dictyoptères.

les blattes sont des dictyoptères.

les mantes sont des dictyoptères.

les cancrelats sont des dictyoptères.

seuls les insectes qui se métamorphosent complètement sont des holométaboles.

seuls les holométaboles qui ont 2 ailes et qui n ont pas d élytre et qui sucent la sève sont des diptères.

la mouche est un diptère.

le taon est un diptère.

le moustique est un diptère.

seuls les holométaboles qui ont 4 ailes et qui ont les ailes colorées et qui sucent la sève sont des lépidoptères.

les papillons sont des lépidoptères.

les papillons ont des larves.

les papillons ont des chenilles.

les chenilles se métamorphosent.

les mites sont des lépidoptères.

les teignes sont des lépidoptères.

seuls les holométaboles qui ont 4 ailes et qui ont les ailes jumelées et qui ont les ailes longitudinales et qui broient la nourriture sont des hyménoptères.

les hyménoptères ont un cou mince.

les hyménoptères ont la taille mince.

les hyménoptères construisent des nids en cire.

les hyménoptères vivent en groupe.

les hyménoptères produisent de la cire.

les hyménoptères ont un dard venimeux.

les abeilles sont des hyménoptères.

les abeilles produisent du miel.

les frelons sont des hyménoptères.

les guêpes sont des hyménoptères.

les bourdons sont des hyménoptères.

seuls les hyménoptères qui construisent des fourmilières sont des fourmis.

les fourmis vivent en groupe.

la fourmi a un dard venimeux.

seuls les holométaboles qui ont 2 ailes et qui ont 2 élytres et qui n ont pas de pinces à l abdomen et qui broient la nourriture sont des coléoptères.

% les coléoptères ont 2 ailes atrophiées en carapace

le hanneton est un coléoptère.

le doryphore est un coléoptère.

la coccinelle est un coléoptère.

le scarabée est un coléoptère.

seuls les holométaboles qui n ont pas d ailes et qui sucent le sang des animaux sont des siphonactères.

les siphonactères sont des parasites.

le poux est un siphonactère.

la puce est un siphonactère.

le morpion est un siphonactère.

seul un holométabole qui a 4 ailes et qui a les ailes à poils et qui a les ailes en toit et qui a les pattes non symétriques et qui broie la nourriture est un trichoptère.

% les larves construisent un fourreau

seul un insecte qui se métamorphose incomplètement est un hétérométabole.

seuls les hétérométaboles qui ont 4 ailes et qui broient la nourriture sont des odonates.

%les odonates peuvent faire du vol stationnaire.

les libellules sont des odonates qui ont les ailes horizontales.

la demoiselle est un odonate qui a les ailes verticales.

seul un hétérométabole qui a 2 ailes et qui a 2 élytres et qui suce la sève est un hétéroptère.

la punaise est un hétéroptère.

seul un hétérométabole qui a 2 ailes et qui a 2 élytres et qui a des pinces à l'abdomen et qui broie la nourriture est un dermaptère.

le perce oreille est un dermaptère.

seul un hétérométabole qui a 4 ailes et qui a les ailes verticales et qui a des pinces à l'abdomen et qui broie la nourriture est un éphéméroptère.

l'éphémère est un éphéméroptère.

seul un hétérométabole qui a 4 ailes et qui a les ailes en toit et qui chante avec ses ailes et qui suce la sève est un homoptère.

la cigale est un homoptère.

le puceron est un homoptère.

la cochenille est un insecte qui suce la sève.

la cochenille est un parasite.

%la cochenille est mal définie

seuls les hétérométaboles qui ont 4 ailes et qui ont les ailes longitudinales et qui ont des pinces à l'abdomen et qui ont un long corps et qui broient la nourriture sont des pléoptères.

seuls les hétérométaboles qui ont des pattes non symétriques et qui ont les ailes longitudinales sont des orthoptères.

les sauterelles sont des orthoptères carnivores qui chantent avec les ailes.

les grillons sont des orthoptères herbivores qui chantent avec les ailes.

les criquets sont des orthoptères herbivores qui chantent avec les pattes.

seuls les arthropodes qui ont des pattes nombreuses et qui ont 2 antennes sont des myriapodes.

le scolopendre est un myriapode.

le scolopendre a une morsure venimeuse.

seuls les arthropodes qui ont 4 antennes et qui ont des branchies et qui ont des pattes sont des crustacés.

%les crustacés ont au moins 8 pattes.

les crustacés ont la peau dure.

les crustacés ont 2 parties.

un animal qui a des branchies vit dans l'eau.

le crabe est un crustacé.

le crabe régénère ses pattes.

le homard est un crustacé qui a un long abdomen.

la langouste est un crustacé qui a un long abdomen.

l'écrevisse est un crustacé qui a un long abdomen.

la crevette est un crustacé qui a un long abdomen.

le bernard l'ermite vit dans une coquille vide.

le bernard l'ermite a un corps mou.

seuls les invertébrés qui ont le corps mou et qui n'ont pas de pattes sont des mollusques.

les mollusques n'ont pas d'anneaux.

les mollusques vivent dans les lieux humides.

les mollusques ont la peau humide.

seuls les mollusques qui ont une coquille en spirale et qui ont 2 antennes sont des gastéropodes.

les gastéropodes ont un pied à ventouse.

l'escargot est un gastéropode.

le bigorneau est un gastéropode.

seul un mollusque qui n'a pas de coquille et qui a deux antennes est une limace.

seuls les mollusques qui ont une coquille conique sont des scaphopodes.

seuls les mollusques qui ont 2 coquilles sont des bivalves.

les bivalves se fixent au sol.

les bivalves ont des branchies.

l'huître est un bivalve.

la moule est un bivalve.

la palourde est un bivalve.

la coquille saint-jacques est un bivalve.

seuls les mollusques qui n'ont pas de coquille et qui ont des tentacules à ventouse sont des céphalopodes.

le calmar est un céphalopode.

l'encornet est un calmar.

la seiche est un céphalopode qui a un os.

la pieuvre est un céphalopode qui a 8 tentacules.

le poulpe est une pieuvre.

la pieuvre régénère ses tentacules.

la pieuvre crache de l'encre.

seuls les invertébrés qui n'ont pas de pattes et qui ont un long corps et qui ont un corps à anneaux sont des vers.

les vers n'ont pas d'antennes.

les vers n'ont pas de coquille.

les vers produisent du fil.

le lombric est un ver.
la sangsue est un ver.
le ténia est un ver.
le ténia est un parasite.

seuls les invertébrés qui n'ont pas de pattes et qui ne sont pas symétriques sont des spongiaires.
les spongiaires vivent dans l'eau.
les spongiaires produisent du calcaire.
les éponges sont des spongiaires.
l'éponge se régénère.

seuls les invertébrés qui n'ont pas de pattes et qui ont des cellules urticantes sont des cnidaires.
le corail est un cnidaire.
le corail se fixe au sol.
l'hydre est un cnidaire.
l'hydre a 8 tentacules.
l'hydre se régénère.
la méduse est un cnidaire.
l'anémone de mer est un cnidaire.
l'anémone de mer se fixe au sol.

seuls les invertébrés qui n'ont pas de pattes et qui ont 5 symétries sont des échinodermes.
les échinodermes n'ont pas de tentacules.
les échinodermes ont un pied à ventouse.

les oursins sont des échinodermes qui ont des longues épines.
les étoiles de mer sont des échinodermes.
l'étoile de mer régénère ses membres.
% l'étoile de mer a des membres ou des pattes ou des tentacules ? les échinodermes n'ont ni pattes ni tentacule
% c'est quoi une tentacule
% c'est quoi un membre par rapport à une patte, pour l'étoile de mer en particulier

la chatte est un chat de sexe féminin.
le chaton est un jeune chat.
la chienne est un chien de sexe féminin.
un chiot est un jeune chien.
une louve est un loup de sexe féminin.
un louveteau est un jeune loup.
un bélier est un mouton de sexe masculin.
une brebis est un mouton de sexe féminin.
un agneau est un jeune mouton.
un bouc est une chèvre de sexe masculin.
la jument est un cheval de sexe féminin.
le poulain est un jeune cheval.
la biche est un cerf de sexe féminin.
le faon est un jeune cerf.
la tigresse est un tigre de sexe féminin.

la lionne est un lion de sexe féminin.
le lionceau est un jeune lion.
le taureau est une vache de sexe masculin.
le taurillon est un jeune taureau.
la génisse est une jeune vache.
le veau est une jeune vache.
la renarde est un renard de sexe féminin.
un poussin est un jeune oiseau.
un poulet est une jeune poule.
un jars est une oie de sexe masculin.
un dindon est une dinde de sexe masculin.
un dindonneau est une jeune dinde.
une cane est un canard de sexe féminin.
un caneton est un jeune canard.
un éléphanteau est un jeune éléphant.
un girafon est une jeune girafe.
une chamelle est un chameau de sexe féminin.
une lapine est un lapin de sexe féminin.
un lapereau est un jeune lapin.
une truie est un porc de sexe féminin.
un verrat est un porc de sexe masculin.
un porcelet est un jeune porc.
une ânesse est un âne de sexe féminin.
un ourson est un jeune ours.
un baleineau est une jeune baleine.
une femme est un homme de sexe féminin.

équivalent non.

l opossum a une gestation de 12 j.
le hamster a une gestation de 16 j.
la souris a une gestation de 21 j.
le rat a une gestation de 22 j.
le lapin a une gestation de 30 j.
la taupe a une gestation de 40 j.
le lièvre a une gestation de 42 j.
le renard a une gestation de 54 j.
le hérisson a une gestation de 58 j.
le chat a une gestation de 60 j.
le chien a une gestation de 63 j.
le loup a une gestation de 68 j.
le cobaye a une gestation de 68 j.
le lynx a une gestation de 70 j.
la panthère a une gestation de 93 j.
le lion a une gestation de 106 j.
le porc a une gestation de 115 j.
le castor a une gestation de 128 j.
le mouton a une gestation de 150 j.
la chèvre a une gestation de 150 j.
la gazelle a une gestation de 160 j.

le chamois a une gestation de 165 j.
l hippopotame a une gestation de 200 j.
l ours brun a une gestation de 220 j.
le daim a une gestation de 240 j.
le renne a une gestation de 246 j.
le morse a une gestation de 260 j.
le chevreuil a une gestation de 270 j.
l homme a une gestation de 270 j.
le phoque a une gestation de 276 j.
la vache a une gestation de 280 j.
le chameau a une gestation de 320 j.
le cheval a une gestation de 335 j.
la baleine a une gestation de 335 j.
le lama a une gestation de 360 j.
le marsouin a une gestation de 360 j.
l âne a une gestation de 365 j.
le zèbre a une gestation de 375 j.
le tapir a une gestation de 400 j.
la girafe a une gestation de 440 j.
le cachalot a une gestation de 480 j.
le morse a une gestation de 480 j.
le rhinocéros a une gestation de 490 j.
l éléphant a une gestation de 600 j.

l abeille a une longévité de 1 an.
l aigle a une longévité de 47 an.
l alligator a une longévité de 56 an.
l âne a une longévité de 18 an.
l anguille a une longévité de 5 an.
l antilope a une longévité de 25 an.
l araignée a une longévité de 26 an.
l autruche a une longévité de 68 an.
la baleine a une longévité de 80 an.
la belette a une longévité de 8 an.
le boa a une longévité de 40 an.
le bigorneau a une longévité de 30 an.
la vache a une longévité de 30 an.
le cacatoès a une longévité de 80 an.
le cafard a une longévité de 4 an.
le calmar a une longévité de 4 an.
le canard a une longévité de 15 an.
le canari a une longévité de 12 an.
la carpe a une longévité de 30 an.
le castor a une longévité de 20 an.
le cerf a une longévité de 28 an.
le chameau a une longévité de 29 an.
le chamois a une longévité de 20 an.
le chat a une longévité de 13 an.
la chauve souris a une longévité de 32 an.
le cheval a une longévité de 20 an.

la chèvre a une longévité de 2 an.
le chien a une longévité de 10 an.
le chimpanzé a une longévité de 20 an.
le cobra a une longévité de 12 an.
le cochon d inde a une longévité de 3 an.
la coquille saint jacques a une longévité de 2 an.
le corbeau a une longévité de 20 an.
la corneille a une longévité de 15 an.
le crapaud a une longévité de 30 an.
le crocodile a une longévité de 70 an.
le cygne a une longévité de 29 an.
le daim a une longévité de 10 an.
le dauphin a une longévité de 32 an.
la dinde a une longévité de 12 an.
l écureuil a une longévité de 8 an.
l éléphant a une longévité de 30 an.
l éponge a une longévité de 50 an.
l escargot a une longévité de 1 an.
l esturgeon a une longévité de 30 an.
l étourneau a une longévité de 16 an.
le flétan a une longévité de 40 an.
la girafe a une longévité de 33 an.
le goéland a une longévité de 32 an.
le gorille a une longévité de 39 an.
la grenouille a une longévité de 5 an.
le hamster a une longévité de 2 an.
le hérisson a une longévité de 4 an.
le hibou a une longévité de 68 an.
l hippocampe a une longévité de 4 an.
l hippopotame a une longévité de 30 an.
l hirondelle a une longévité de 8 an.
le homard a une longévité de 30 an.
l huitre a une longévité de 12 an.
le jaguar a une longévité de 23 an.
le jars a une longévité de 49 an.
le kangourou a une longévité de 10 an.
le lapin a une longévité de 6 an.
le lézard a une longévité de 54 an.
la limace a une longévité de 1.5 an.
le lion a une longévité de 10 an.
le loup a une longévité de 10 an.
le mainate a une longévité de 17 an.
le maquereau a une longévité de 15 an.
le martinet a une longévité de 21 an.
le moineau a une longévité de 10 an.
la mouche a une longévité de 76 j.
la moule a une longévité de 60 an.
le moustique a une longévité de 1.5 j.
le mouton a une longévité de 12 an.
l oie a une longévité de 15 an.

l opossum a une longévité de 7 an.
l orang outan a une longévité de 59 an.
l ours a une longévité de 5 an.
le pélican a une longévité de 52 an.
le perroquet a une longévité de 35 an.
la perruche a une longévité de 10 an.
la pieuvre a une longévité de 4 an.
le pigeon a une longévité de 10 an.
le pinson a une longévité de 10 an.
le poisson rouge a une longévité de 41 an.
le poney a une longévité de 54 an.
le porc a une longévité de 10 an.
la poule a une longévité de 7 an.
la punaise a une longévité de 1.5 an.
le rat a une longévité de 3 an.
le raton laveur a une longévité de 14 an.
le renard a une longévité de 8 an.
le rhinocéros a une longévité de 36 an.
le rossignol a une longévité de 3 an.
la sangsue a une longévité de 27 an.
le saumon a une longévité de 13 an.
le singe a une longévité de 50 an.
la souris a une longévité de 1 an.
le taureau a une longévité de 30 an.
le termite a une longévité de 15 an.
le thon a une longévité de 7 an.
le tique a une longévité de 4 j.
la tortue a une longévité de 20 an.
la tourterelle a une longévité de 15 an.
la truite a une longévité de 41 an.
la vache a une longévité de 9 an.
le vautour a une longévité de 41 an.
le lombric a une longévité de 6 an.
le vison a une longévité de 10 an.

le coucou a une incubation de 10 j.
le passereau a une incubation de 12 j.
la mésange a une incubation de 13 j.
la fauvette a une incubation de 13 j.
le moineau a une incubation de 13 j.
le rouge gorge a une incubation de 14 j.
le pinson a une incubation de 14 j.
la grive a une incubation de 14 j.
l hirondelle a une incubation de 15 j.
le merle a une incubation de 15 j.
le pigeon a une incubation de 17 j.
la poule a une incubation de 21 j.
le faisan a une incubation de 26 j.
le canard a une incubation de 28 j.
le dindon a une incubation de 28 j.

le héron a une incubation de 28 j.
l oie a une incubation de 28 j.
la pintade a une incubation de 28 j.
le faucon a une incubation de 29 j.
la cigogne a une incubation de 30 j.
le flamant a une incubation de 30 j.
le cygne a une incubation de 40 j.
l autruche a une incubation de 50 j.
le vautour a une incubation de 54 j.
le kiwi a une incubation de 80 j.

le paresseux a une course de 0.012 kmh.
l escargot a une course de 0.05 kmh.
le boa a une course de 0.36 kmh.
la tortue a une course de 0.37 kmh.
le cafard a une course de 5 kmh.
le crabe a une course de 12 kmh.
le crocodile a une course de 12 kmh.
le cochon a une course de 17 kmh.
le mouton a une course de 24 kmh.
le chameau a une course de 25 kmh.
l homme a une course de 37 kmh.
le lapin a une course de 38 kmh.
l éléphant a une course de 40 kmh.
le chien a une course de 45 kmh.
le loup a une course de 45 kmh.
le rhinocéros a une course de 45 kmh.
le daim a une course de 48 kmh.
le kangourou a une course de 48 kmh.
l autruche a une course de 50 kmh.
l émeu a une course de 50 kmh.
la girafe a une course de 50 kmh.
le phacochère a une course de 50 kmh.
le chacal a une course de 55 kmh.
le buffle a une course de 55 kmh.
l antilope a une course de 70 kmh.
l hyène a une course de 65 kmh.
le zèbre a une course de 65 kmh.
le cheval a une course de 69 kmh.
le lévrier a une course de 69 kmh.
le lièvre a une course de 70 kmh.
la gazelle a une course de 80 kmh.
le gnou a une course de 80 kmh.
le lion a une course de 80 kmh.
le chevreuil a une course de 98 kmh.
le guépard a une course de 100 kmh.

la chauve souris a un vol de 16 kmh.
la bécasse a un vol de 21 kmh.
le chardonneret a un vol de 30 kmh.

l alouette a un vol de 32 kmh.
le corbeau a un vol de 38 kmh.
le faucon a un vol de 40 kmh.
le pélican a un vol de 48 kmh.
la grue a un vol de 50 kmh.
le faisan a un vol de 59 kmh.
le pigeon a un vol de 63 kmh.
la corneille a un vol de 76 kmh.
l épervier a un vol de 80 kmh.
la perdrix a un vol de 84 kmh.
le cygne a un vol de 88 kmh.
le canard a un vol de 90 kmh.
la buse a un vol de 120 kmh.
l oie a un vol de 142 kmh.
le vautour a un vol de 150 kmh.
l aigle a un vol de 161 kmh.
le poisson volant a un vol de 72 kmh.

l homme a une nage de 6.5 kmh.
l otarie a une nage de 40 kmh.
le rorqual a une nage de 37 kmh.
le cachalot a une nage de 40 kmh.
la baleine a une nage de 48 kmh.
le marsouin a une nage de 55 kmh.
le dauphin a une nage de 15 kmh.
le homard a une nage de 23 kmh.
la pieuvre a une nage de 6 kmh.
la perche a une nage de 2 kmh.
l anguille a une nage de 12 kmh.
la carpe a une nage de 12 kmh.
le mulot a une nage de 12 kmh.
le gardon a une nage de 16 kmh.
le brochet a une nage de 33 kmh.
la truite a une nage de 37 kmh.
le saumon a une nage de 40 kmh.
le poisson volant a une nage de 56 kmh.
le merlan a une nage de 70 kmh.
l espadon a une nage de 92 kmh.
le cormoran a une nage de 2.5 kmh.
le manchot a une nage de 27 kmh.
le pingouin a une nage de 36 kmh.
la nage du crocodile est 20 kmh.
la nage de la tortue est de 35 kmh.

On peut alors poser les questions suivantes :

quels sont les canidés dont la course est supérieure à 40 kmh.

[chacal, chien, chienne, chiot, loup, louve, louveteau, lévrier]

quels sont les canidés qui ont une course supérieure à 30 kmh.

[chacal, chien, chienne, chiot, loup, louve, louveteau, lévrier]

quelle est la course des canidés

[(45,kmh),(55,kmh),(69,kmh)]

quelles sont les courses respectives des canidés.

canidé inconnu

chacal 55,kmh

chien 45,kmh

chienne 45,kmh

chiot 45,kmh

loup 45,kmh

louve 45,kmh

louveteau 45,kmh

lévrier 45,kmh

lévrier 69,kmh

renard inconnu

renarde inconnu

quelle est la course du lévrier.

[(45,kmh),(69,kmh)]

le lévrier est un chien, donc coco retient 45 pour chien et 69 pour lévrier

quels sont les canidés dont la course est supérieure à celle du chien.

[chacal, lévrier]

qui a le bec jaune.

[albatros, goéland, mouette, neognathae, pétrel]

quels sont les félidés qui n ont pas les griffes rétractiles.

[guépard]

le fichier famille.txt contient des règles, décrivant les relations familiales :

si x est le père de y alors x est le parent de y.
si x est la mère de y alors x est le parent de y.
si x est le parent de y alors y est l'enfant de x.
si x est le parent de y et si y est masculin alors y est le fils de x.
si x est le parent de y et si y est féminin alors y est la fille de x.
si x est le parent de y et si y est le parent de z alors x est le grandparent de z.
si x est le grandparent de y et si x est masculin alors x est le grandpère de y.
si x est le grandparent de y et si x est féminin alors x est la grandmère de y.
si x est le grandparent de y alors y est le petit-enfant de x.
si x est le grandparent de y et si y est masculin alors y est le petit-fils de x.
si x est le grandparent de y et si y est féminin alors y est la petite-fille de x.
si x est le parent de y et si x est le parent de z et si y n'est pas z alors y est le frère ou la sœur de z.
si x est le frère ou la sœur de y et si x est masculin alors x est le frère de y.
si x est le frère ou la sœur de y et si x est féminin alors x est la sœur de y.
si x est le parent de y et si z est le frère ou la sœur de x alors z est l'oncle ou la tante de y.
si x est l'époux de y ou si x est l'épouse de y alors x est l'époux ou l'épouse de y.
si x est l'époux de y alors y est l'épouse de x.
% attention la règle si x est l'épouse de y alors y est l'époux de x donne une boucle infinie
% donc il faut déclarer les époux et pas les épouses
si x est le parent de y et si z est le frère ou la sœur de x et si t est l'époux ou l'épouse de z alors t est l'oncle ou la tante de y.
si x est l'oncle ou la tante de y et si x est masculin alors x est l'oncle de y.
si x est l'oncle ou la tante de y et si x est féminin alors x est la tante de y.
si x est l'oncle ou la tante de y alors y est le neveu ou la nièce de x.
si x est l'oncle ou la tante de y et si y est masculin alors y est le neveu de x.
si x est l'oncle ou la tante de y et si y est féminin alors y est la nièce de x.
si x est le parent de y et si z est l'époux de y alors x est le beau-parent de z.
si x est le parent de y et si z est l'épouse de y alors x est le beau-parent de z.
si x est le beau-parent de y et si x est masculin alors x est le beau-père de y.
si x est le beau-parent de y et si x est féminin alors x est la belle-mère de y.
si x est le beau-parent de y alors y est le gendre ou la bru de x.
si x est le beau-parent de y et si y est masculin alors y est le gendre de x.
si x est le beau-parent de y et si y est féminin alors y est la bru de x.
si x est le beau-parent de y et y est le parent de z alors z est le beau-frère ou la belle-sœur de x.
si x est le beau-frère ou la belle-sœur de y et x est masculin alors x est le beau-frère de y.
si x est le beau-frère ou la belle-sœur de y et x est féminin alors x est la belle-sœur de y.
si y est le parent de x et si z est l'oncle ou la tante de x et si z est le parent de t alors x est le cousin ou la cousine de t.
si x est le cousin ou la cousine de y et si x est masculin alors x est le cousin de y.
si x est le cousin ou la cousine de y et si x est féminin alors x est la cousine de y.

Puis quelques faits précisant pour chaque quidam son père, sa mère et son sexe :

bernard est le père de pierre.
geneviève est la mère de pierre.
pierre est de sexe masculin.
bernard est le père de dominique.

geneviève est la mère de dominique.
dominique est de sexe féminin.
pierre est le père d antoine.
catherine est la mère d antoine.
antoine est de sexe masculin.
pierre est le père de soizic.
catherine est la mère de soizic.
soizic est de sexe féminin.

philippe est l époux de dominique.
philippe est de sexe masculin.
philippe est le père d éric.
dominique est la mère d éric.
éric est de sexe masculin.
philippe est le père de Bénédicte.
dominique est la mère de Bénédicte.
Bénédicte est de sexe féminin.

D'où les questions possibles :

quels sont les cousins d antoine
quels sont les oncles de soizic
qui sont les gendres de bernard
quelles sont les petites filles de bernard

Systeme expert

Pour transformer le programme en système expert, la lecture du fichier animal par la commande :

fichier expert animal.

va produire non pas des règles

affirme(x,verbe,y,vrai) :- affirme(...), affirme (...).

mais des règles :

regle(affirme(x,verbe,y,vrai) , [affirme(), affirme (...)] , _espèce, _sous-espèce).

Comme conséquence, ce n'est plus le moteur d'inférence de prolog qui va résoudre le problème mais un moteur d'inférence programmé qui va parcourir les règles définissant les espèces.

Petit rappel : on a des règles qui définissent des espèces d'animaux :

seuls les mammifères terrestres carnivores qui ont des griffes non rétractiles et qui ont des oreilles triangulaires sont des canidés.

```
affirme(_861,être,canidé,vrai) :-  
    affirme(_861,être,mammifère,vrai),  
    affirme(_861,avoir,habitat,terrestre,vrai),  
    affirme(_861,avoir,nourriture,carnivore,vrai),  
    affirme(_861,avoir,griffe,non(rétractil),vrai),  
    affirme(_861,avoir,oreille,triangulaire,vrai).
```

Qui devient donc :

```
regle(affirme(_861,être,canidé,vrai),  
    [affirme(_861,être,mammifère,vrai),  
    affirme(_861,avoir,habitat,terrestre,vrai),  
    affirme(_861,avoir,nourriture,carnivore,vrai),  
    affirme(_861,avoir,griffe,non(rétractil),vrai),  
    affirme(_861,avoir,oreille,triangulaire,vrai) ],  
    mammifère,  
    canidé )
```

les 3^{ème} et 4^{ème} paramètres de la règle servent à classer les règles, comme ici définissant un canidé à partir de particularités des mammifères.

et d'autres donnant la liste des membres de l'espèce :

un renard est un canidé. un chacal est un canidé. ...

affirme(_x,être,canidé,vrai) :- affirme(_x,être,renard,vrai).

affirme(_x,être,canidé,vrai) :- affirme(_x,être,chacal,vrai).

d'autres donnant des capacités, ne servant pas pour le système expert :

les canidés ont des canines longues.

affirme(_x,avoir,canine longue,vrai) :- affirme(_x,être,canidé,vrai).

Le programme va chercher à prouver les affirmations du genre

affirme(coco, être, _espèce, vrai). (en commençant avec _espèce = vertébré)

dans le but d'en trouver une dont toutes les conditions sont vérifiées.

Pour cela, on recherche les règles qui ont ce type de but (en fait il n'y en a qu'une, puisque c'est une règle de définition « seul ... »)

Supposons qu'on en soit à prouver affirme(coco, être, canidé, vrai).

Le programme va prendre une à une les 5 conditions d'être un canidé et va chercher à les prouver. La première des 5 est celle définissant un membre des mammifères, qu'il faut donc essayer de prouver avec coco comme membre putatif. Cette affirmation n'est pas dans la pile « connu », ni son contraire affirmation(coco, être, mammifère, faux), mais elle est prouvable par une autre règle puisqu'elle est le but d'une règle. Si c'est vérifié, coco est donc un mammifère et on peut stocker cette affirmation dans la pile « connu », puis on peut passer à la 2^{ème} condition d'être un canidé. Si cette deuxième condition, sur l'habitat terrestre, n'est pas prouvable cela va générer une question à l'utilisateur. Si l'habitat de coco n'est pas terrestre, il faut chercher une autre règle pour être un canidé, et il n'y en a pas donc le fait d'être canidé n'est pas prouvable, et la question est posée à l'utilisateur.

Il y a un fort risque, et même certitude, de boucler car la définition exacte des mammifères est :

seuls ceux qui produisent du lait sont des mammifères, soit 1 règle de plus pour la réciproque :

```
règle (affirme(_x, _être, mammifère, vrai ),
      [ affirme(_x, produire, lait, vrai) ]).
règle (affirme(_x, produire, lait, vrai).
      [ affirme(_x, être, mammifère, vrai) ]).
```

Et donc le fait `affirme(_x, produire, lait, vrai)` devient démontrable par cette deuxième règle , bouclant sur « être mammifère » que l'on est justement en train d'essayer de démontrer. Il faut donc gérer une pile « encours » avec les faits en cours de démonstration, et en évitant d'y remettre un fait qui y est déjà. Cette pile est vidée dès que l'on démontre quelque chose.

La règle de l'équivalent logique négatif permet, d'effectuer des déductions intéressantes, réduisant l'espace de recherche, et donc cela ajoute les règles :

```
règle (affirme(_x, _être, mammifère, faux ),
      [ affirme(_x, produire, lait, faux) ]).
règle (affirme(_x, produire, lait, faux).
      [ affirme(_x, être, mammifère, faux) ]).
```

Mais quand les affirmations ont plusieurs conditions, cela produit plusieurs règles qui risquent de lancer le système sur des pistes non productives au final, mais posant des questions inutiles et pouvant paraître illogiques. C'est pourquoi le fichier `animal.txt` exclu ces deux possibilités en mentionnant : équivalent non et réciproque non.

Le moteur est basé sur la règle « classe » dont la fonction est de vérifier l'appartenance à une espèce et d'en rechercher les sous espèce, et « prouve » dont le rôle est de tenter de prouver un fait en testant d'abord s'il est connu ou si sa négation est connue puis en tentant de prouver toutes ses conditions. On le lance par la commande : `expert`.

```

expert :-
    retractall(connu(_)),
    retractall(encours(_)),
    classe(vertébré,_résu).
expert :-
    classe(invertébré,_résu).

%prouver l'appartenance à l'espèce _but et rechercher les sous espèces de _but
classe(_but,[_but|_résu]) :-
    prouve(racine,affirme(coco, être, _but, vrai)),
    espece(_but,_résu).

% recherche des règles définissant l'espèce _but comme dérivant d'une autre _c par des
particularités
espece(_but,_résu):-
    writen('c est une espèce de ',_but),nl,
    regle(_a,_b,_c,_but),
    writen('suivant ',_but,_a,_b,_c,_but),
    classe(_c,_résu),!.
%recherche des membres de l'espèce
espece(_but,[_y]):-
    writen('voici les ',_but,' que je connais sans avoir de règles pour les différencier'),nl,
    regle(affirme(_x,être,_but,vrai),[affirme(_x,être,_y,vrai)],_a,_b),
    writen('espèce finale suivante ',_but,_y),nl,fail.
espece(_but,[]).

prouve(_regle,_Fait) :-
    connu(_Fait), !.
prouve(_regle,_Fait) :-
    nier_fait(_Fait,_ret),
    connu(_ret), !, fail.
prouve(_regle,_Fait) :-
    encours(_Fait),!,fail.
prouve(_regle,_Fait) :-
    writen(prouve_fait,_Fait,parceque,_regle),nl,
    assertz(encours(_Fait)),
    regle(_Fait,_Conditions,_x,_y),
    notsingle(_Fait,_Conditions),
    prouve_liste(regle(_Fait,_Conditions),_Conditions),
    writen(' - enregistre prouvé : ',_Fait),nl,
    retractall(encours(_)),
    assertz(connu(_Fait)),!.
prouve(_regle,_Fait) :-
    poser_question(_Fait).

%poser_question pose la question en éliminant la double négation, stocke la réponse dans
« connu » et réussit ou échoue selon la réponse oui ou non

%éliminer les règles avec une seule condition, définissant les animaux non différenciés
notsingle(affirme(_a,être,_b,vrai),[affirme(_a,être,_y,vrai)]):- !,fail.
notsingle(_a,_b).

```

Certains mots sont des contraires, par exemple : vivipare / ovipare, vertébré / invertébré, rétractile / non rétractile, ...

Cela intervient dans des faits du genre

affirme(_x, être, vivipare, -vrai)

affirme(_x, avoir ,griffe , rétractile, _vrai)

affirme(_x, avoir, patte, non(symétrique), _vrai)

et donc lorsqu'on recherche si un fait est connu ou nié, il faut en tenir compte :

inv(vrai,faux):-!.

inv(faux,vrai).

contr(_x,_y):-contraire(_x,_y).

contr(_x,_y):-contraire(_y,_x).

contr(non(_x),_x) :- !.

contr(_x,non(_x)).

contraire(ovipare,vivipare).

contraire(diurne,nocturne).

contraire(sec,humide).

contraire(vertébré,invertébré).

Et les deux premières règles « prouve » deviennent :

prouve(_regle,_Fait) :-

connu(_Fait), !.

prouve(_regle,affirme(_x,être,_y,_vrai)):-

contr(_y,_z),

inv(_vrai,_faux),

connu(affirme(_x,être,_z,_faux)),!.

prouve(_regle,affirme(_x,avoir,_y,_z,_vrai)):-

contr(_z,_t),

inv(_vrai,_faux),

connu(affirme(_x,avoir,_y,_t,_faux)),!.

prouve(_regle,_Fait) :-

nier_fait(_Fait,_ret),

connu(_ret), !, fail.

prouve(_regle,affirme(_x,être,_y,_vrai)):-

contr(_y,_z),

connu(affirme(_x,être,_z,_vrai)),!,fail.

prouve(_regle,affirme(_x,avoir,_y,_z,_vrai)):-

contr(_z,_t),

connu(affirme(_x,avoir,_y,_t,_vrai)),!,fail.

Le fichier animal.txt est améliorable, ne serait-ce que pour distinguer les membres d'une sous espèce (renard, loup, chacal) mais aussi, dans l'état actuel, le système part encore sur des pistes sans espoir, par exemple cherchant à prouver « poisson », il cherche tout ce qui amène des branchies alors qu'il a déjà déduit que l'animal respire dans l'air.

Voir coco.pl dans jeux/myprol/win64/debug/pro/coco.pl

dictionnaire automatisé

Tous les exemples étudiés jusqu'ici utilisent un vocabulaire réduit, défini dans le programme lui-même. Pour éviter cette phase fastidieuse de définition dans chaque programme, il faut séparer la grammaire et le programme qui l'analyse, enfin il faut créer un dictionnaire automatisé du français contenant pour chaque mot ses caractéristiques syntaxiques et sémantiques. Dans ce dictionnaire, chaque mot est défini par une liste d'attributs prenant leurs valeurs dans des listes partiellement ou complètement prédéfinies.

Le dictionnaire encyclopédique présenté ici, dont la forme dérive du thésaurus présenté en première partie, est une maquette expérimentale, en évolution permanente. Il consiste en une couverture complète du lexique français, et une difficulté est de le tenir cohérent dans son entier lors d'une évolution des attributs de définition des mots.

La grammaire est écrite en Prolog et les outils associés ont tous été écrits en langage pascal delphi puis en perl, avec le module graphique Tkx (activeperl). Puis cela a été traduit en python, pour pouvoir utiliser les modules de statistiques d'IA et passer à swi-prolog.

La racine d'un mot est présentée seule sur une ligne, chaque attribut est indenté d'une tabulation et éventuellement suivi du caractère « = » si des valeurs doivent être spécifiées, sur des nouvelles lignes indentées d'une autre tabulation

```
mot
  catégorie
  attribut=
    valeur1
    valeur2
pour les attributs, on utilise des expressions du genre :
  attribut=
    valeur
ou      attribut=
        sousattribut1:valeur1,sousattribut2:valeur2
```

1.1 les attributs syntaxiques et sémantiques

ces attributs sont :

- **la catégorie grammaticale au sens classique du terme**

Ces catégories sont des attributs sans valeur :

nom masculin (Nm) nom féminin (Nf) nom propre (Npm ou Npf) nom neutre (Nn)

nom mixte (Nmf)	verbe (V)	adjectif (Adj)	adverbe (Adv)
pronom (Pro)	préposition (Prép)	conjonction (Conj)	article (Art)
interjection (Interj)	onomatopée (Onom)	expression nom (Expnm)	phrase (expphrase)
expression verbe (Expv)	expression propre (Expnpm)	expression adj (Expadj)	

pour les mots outils (pronoms, prépositions, conjonctions, démonstratifs, comparatifs, numéraux, ...) il existe d'autres catégories indiquant chacune une construction particulière référencée dans la grammaire du français utilisée par le noyau d'analyse en prolog.

Nm, Nf, Nmf, Nn sert à distinguer les noms toujours du même genre et les mots pouvant être utilisés avec les deux genres, pouvant alors avoir la même forme ou une forme suffixée différemment.

Travailleur c'est nmf (vivent les travailleuses)

Ministre c'est nn (vous êtes plutôt parité et quota ou compétence ?)

Table c'est nf

Loup c'est nmf (il y a des louves)

Duc c'est nmf (n'oublions pas la duchesse)

- les expressions figées ou variables

elles peuvent être accrochées à l'un des mots qui la composent (verbe ou nom)

abominable

expnm=

abominable homme des neiges

ce n'est qu'un nom qui contient un ou plusieurs espaces, mais l'analyseur lexical ne reconnaît que des mots sans espace, il y a une surcouche pour les expressions.

marcher

expv=

marcher sur ses deux jambes

con

expadj=

con comme un balai

dans l'expv « marcher sur ses deux jambes », « marcher » se conjugue et « ses » peut devenir « mes », « tes » ou « leurs »

dans l'expadj « con comme un balai », con peut se dériver en cons, conne, ...

les phrases toutes faites, les proverbes, les citations sont des expphrase ou exptitre

- le numéro de conjugaison des verbes, comme on le trouve dans le dictionnaire Larousse des conjugaisons, avec quelques ajouts pour différencier éler, éger, érer, éter, ...

conj=

30

- les règles de dérivation et les familles de mots

dérivé du nom (DeN=)

dérivé du verbe (DeV=)

dérivé de l'adjectif (DeAdj=)

Ces attributs De... ne précisent pas assez le type de relation

Le fichier derives.txt systématise les attributs suivants pour les verbes :

Agent	réalise l'action
acte	fait de faire l'action
résu	résultat de l'action
ins	outil servant à réaliser l'action (scier : scie)
org_a	artefact physique ou spirituel utilisé pour réaliser l'action on ajoute qqchse (cheviller : cheville, déstaliniser)
org_r	artefact résultant de l'action (décheviller : cheville) on retire qqchse
prod_a	produit utilisé pour réaliser l'action (ensabler : sable)
prod_r	produit résultant de l'action (désaler : sel)
prod_c	produit créé par l'action
lieu	lieu où se déroule l'action, introduit par une préposition de lieu

(empiler, entasser, déradier)

lieu_in ou _out	l'action consiste à sortir ou entrer dans un lieu
couleur_a ou _r	modifier la couleur de l'objet (brunir, débrunir)
forme_a ou _r	modifier la forme de l'objet (débosseler : bosse, bosseler)
long_a ou _r	
poids_a ou _r	
adnable	
adnonable	
nbase_a ou _r	nom de la racine
abase_a ou _r	donner ou enlever cette caractéristique
inverse	inverse non construit sur la même racine
contraire	inverse construit sur la même racine

- langues étrangères

langue=
 anglais
 le mot est anglais
anglais=
 définit l'équivalent anglais du mot

- mots associés par l'usage

V=
N=
Adj=
Exemple :
 clou
 V=enfoncer, arracher
 chanter
 N=chanson, chant
 lit
 Adj=douillet

1.2 les attributs sémantiques

Ces attributs, lorsqu'ils peuvent prendre de nombreuses valeurs, vont souvent par paire:

si un mot X a un attribut "attrib" de valeur Y
alors le mot Y a un attribut "Aattrib", (A comme Anti), de valeur X.

abeille
 produit=miel
 logis=ruche
miel
 Aproduit=abeille
ruche
 Alogis=abeille

Cette notion d'attributs permet d'établir un réseau entre les mots du dictionnaire. Chaque mot étant donc lié par ces attributs à d'autres mots du dictionnaire, l'ensemble des liens d'un mot avec le reste du dictionnaire représente son champ sémantique, et donne son sens au mot.

- Attributs classificateurs :

Isa= indique que l'objet est un ...
 et hérite de ses propriétés et de celles de ses pères
 cidre
 Isa=boisson
Classe= indique que les objets cités sont des ...
 instrument de musique
 Classe=biniou,gaïta,cabrette,violon
Liste= indique que les objets cités sont les seuls ...
 note
 Liste=do,ré,mi,fa,sol,la,si
Partof= indique que l'objet est une partie de ...
 guidon
 Partof=vélo

Contient= indique que l'objet est composé des objets cités
vélo
Contient=roue,cadre,guidon,selle
Ensemble= indique le nom donné à un ensemble d'objet
bateau
Ensemble=flotte
Ant= spécifie des antonymes
Syn= spécifie des synonymes
Ta= donne des termes associés

- Attributs généraux

forme= soleil
forme=sphère
couleur= sang
couleur=rouge
matière= vis
matière=métal
unité= longueur
unité=mètre
pays= paris
pays=france
région= france
région=bretagne,auvergne,...
symbole= pureté
symbole=colombe,blanc
symbolise= colombe
symbolise=pureté
ustensile= donne la liste des objets annexes
tennis
ustensile=filet,raquette,balle
outil= menuisier
outil(Fonction=couper)=scie
métier= vêtement
métier(Fonction=fabriquer)=tailleur
métier(Fonction=laver)=blanchisseur
contenant= indique le ou les objets contenant l'objet
vin
contenant=bouteille,verre

contenu= indique le contenu de l'objet
 bouteille
 contenu=vin,eau,liquide
 utilisateur= donne la liste des utilisateurs préférentiels de l'objet
 crosse
 utilisateur=évêque
 célèbre= permet de donner le nom de représentants connus
 porte avion
 célèbre=clémenceau
 Nb= pour spécifier une valeur numérique
 domaine= prend les valeurs : maritime, militaire, finance, ...
 genre= définit le genre du discours, cet attribut prend les
 valeurs : argot, populaire, littéraire, ...
 valorisation= cet attribut prend les valeurs :
 favorable ou défavorable, selon l'opinion commune.
 text= pour spécifier un texte, éventuellement sur plusieurs lignes

- Attribut définit par grammaire de cas

usage= (pour un outil ou un objet)
 action= (pour un métier)
 règle= (pour un sport ou un jeux)
 but= (pour un sport ou un jeux)
 vie= (pour un humain)
 œuvre= (pour un humain)
 histoire= (pour un pays ou un objet)

ces attributs prennent des valeur constituées d'une liste de paires sous-attribut:valeur, séparées par une virgule. Les sous attributs sont : v comme verbe, o comme objet, i comme instrument, a comme agent (sujet), l comme lieu, c comme cause, et d'autres comme date, pays, début, fin, fonction, métier, ...

napoléon
 num=1
 npm
 vie=
 v:naître,date:1769
 v:mourir,l:prison,date:1821
 v:combattre,o:europe,bataille:austerlitz,date :1805
 v:combattre,pays:russie,date :1812
 v:combattre,o:europe,bataille:waterloo,date:1815
 fonction:chef militaire
 v:faire,o:coup de état du 18 brumaire
 fonction:premier consul,date:1799
 fonction:empereur,début:1804,fin:1814
 fonction:empereur,date:1815
 v:conquérir,o:italie
 v:conquérir,o:espagne
 v:conquérir,o:egypte

v:conquérir,o:europa centrale
v:abdiquer,date:1814
v:quitter,l:france,destination:île de sainte hélène,cause:exil

Les œuvres d'un personnage ont de multiples formes : action, invention, livre, musique, chanson, peinture, pont, ...

œuvre=
exploration:pôle nord,date:1926,i:avion
invention:machine à vapeur
fondation:hôpital,u:lépreux,l:lambaréné,pays:gabon
roman:ivanhoé,style:historique,ep:moyen age
pièce:hamlet,style:drame
film:ben hur
opéra:don carlos
peinture:le déjeuner de canotiers
canal:canal de panama,date:1882

par ailleurs le titre et les personnages des œuvres sont déclarés

ivanhoé
npm
num=1
isa=personnage
titre=roman:ivanhoé
ivanhoé
titre
num=2
isa=roman
auteur=walter scott

- Pour les adjectifs, on utilise les attributs suivants :

graduable ou non	chaud
excès	épouvantable
intensité	un sacré travailleur

régent= définit l'attribut que doit avoir le mot auquel s'applique l'adjectif

champ=cet attribut prend une ou plusieurs valeurs en fonction du champ d'application :

âge,santé,effort,notoriété,ordre,hygiène,possSESSION,attitude

sociale,affront,humeur,travail,intelligence,habileté,parole,sexe,dépense,justice,politesse,vérité,
peur,religion,vue,odorat,goût,ouïe,abondance,prix,durée

- schéma du champ sémantique

Le menu « relation » permet d'afficher les relations du mot et du 1^{er} niveau de ses sous mots, à l'exception des relations listées dans outils/filtre_attributes.txt

Il est alors possible de zoomer (Ctrl + et Ctrl -) et de déplacer un nœud.

On peut aussi supprimer des nœuds ou des noms de relation, ou de modifier la taille de police ; il faut alors réafficher le schéma avec ces nouveaux paramètres.

- la polysémie

Cette évolution est majeure, elle vient du LVF de Jean Dubois et Françoise Dubois-Charlier mais elle est récente, et n'est pas encore complètement digérée avec toutes ses conséquences. J'ai récupéré le LVF, DEM, et LEF (dictionnaire des verbes, des mots (noms communs), des expressions figées). Les bases de ces dictionnaires sont dans des fichiers excel, qui sont exportés dans des fichiers textes avec séparateur « tabulation » puis ramenés à la syntaxe de mon premier dictionnaire, telle que décrite ci-dessus. J'y ai ajouté un DEP, dictionnaire des noms propres, j'ai transformé le champ « construction » des verbes du LVF, qui était codé avec des nombres et que j'ai rendu plus lisible, avec des symboles, en particulier pour les prépositions. J'ai supprimé de cette construction syntaxique les références à des prépositions implicites du sens du verbe. Enfin j'ai étendu cette construction pour pouvoir spécifier des critères de sélection basés sur tous les attributs du dictionnaire et pas seulement sur les catégories « humain, inanimé, animal, abstrait »

Chaque entrée décrit un sens de la racine, distingué par l'attribut num (numéro d'entrée homonyme) :

num=

numéro d'entrée

numéro de 1 à n pour une racine au sens lexical, 0 étant considéré comme la valeur par défaut, si num n'est pas spécifié, comme c'est le cas dans les fichiers txt originaux avant cette évolution. (on en a vu un exemple ci-dessus avec ivanhoé)

- Attribut sémantiques

Cette évolution majeure vient aussi du DEM.

On décompose le sens du mot en éléments de sens minimum codés dans les attributs domaine (en majuscule), opérateur (op), op1, op2, type (humain, animal, non animé), isa, partof

L'attribut opérateur « op » regroupe les mots semblables, op1 et op2 sert à distinguer ces mots selon certains critères

Un verger, c'est domaine=SYL, op=cultive ds N, op2=arbres fruitiers

une écurie c'est domaine=EQU, op=m an ds N, op2=cheval

- Les verbes du LVF sont définis avec leurs constructions

chaque entrée du verbe correspond à un sens et a une liste de constructions intransitives (A), transitives (N), pronominales (P) et précisant dans chaque cas les prépositions (T) et les attributs (domaine, opérateurs, type, ...) possibles du sujet et des compléments

C'est donc par ces informations fondamentales que se filtrent les cas de polysémie des verbes et des noms. L'analyse de la phrase détecte un verbe, son sujet, ses compléments, et passe en revue les constructions déclarées de ce verbe en ne gardant que celles où le sujet et les compléments ont les bons attributs et prépositions et en filtre donc l'entrée ayant la sémantique à considérer.

Chaque construction précise par un caractère :

- le mode du verbe : intransitif (A), transitif direct ou indirect (N ou T), pronominal (P)
- le type du sujet (2^{ème} caractère) : h, b, a, c, i, x, t pour homme, animal, abstrait, chose, inanimé (chose ou abstrait), impersonnel (il), tout)
- le type du complément direct (3^{ème} caractère)
- la présence d'un ou deux compléments indirects (introduit par une préposition ou une conjonction) : p ou k

- les attributs des compléments indirects ainsi que leurs prépositions sont indiqués après une virgule. S'il y a plusieurs prépositions possibles on les sépare par « / ».

Quelques exemples de constructions des verbes :

Ah	paul court	intransitif
Pi	le ballon s'élève	pronominal
Nhi	paul prend la bille	transitif, 1 complément direct
Thip,pà-h	paul donne une bille à paulette	1 complément direct et 1 indirect
Pip,pdans/sur-i	le ballon s'élève dans le ciel	pronominal à 1 complément indirect
Thp,pvinf	je veux manger une pomme	compl. groupe verbal à l'infinitif
Thpk,pà-h kde-vinf	Paul demande à paulette de venir	2ème complément infinitif
Le sujet de l'infinitif est le 1 ^{er} complément		
Thip,pà-svinf	Paul dépense son argent à jouer	
Le sujet de l'infinitif est celui de la proposition		
Thp,pque-ind	je sais qu'il vient	un complément proposition indicatif
Thp,pque-subj	je veux que Paul tienne la porte	proposition au subjonctif
Thpk,pà-h kque-subj	Paul demande à paulette qu'elle vienne	
Ax	il pleut	sujet impersonnel
Txp,pque-subj	il arrive qu'il pleuve	
Thip,pà-svinf	paul dépense son argent à jouer	verbe infinitif ou acte
Thip,pà-i	Paul dépense son argent au jeu	

Lorsqu'il y a plusieurs constructions pour une entrée définissant un même sens du verbe, on les sépare d'un « ; »

Lorsque le type (h, b, a, v, c et i) du sujet ou des compléments ne suffit pas pour distinguer les différentes entrées du verbe, on utilise comme critère les noms des champs du DEM et même des formules sur ces types, domaines et op ou op1 du DEM. Les opérateurs sont – (ou), + (et) et #(not) qui ont les mêmes priorités que + , * et ** en algèbre. Il n'y a pas de parenthèse. Ces formules peuvent servir à déclarer un « critère » utilisable alors comme un simple nom de champ.

Nhi,cinstitution-lieu	le complément doit être une institution ou un lieu
Thp,pà-institution+lieu	le complément doit être une institution et un lieu
Thi,c#HAB+#VEH	le complément , inanimé, ne doit être ni un habit ni un véhicule
Thp,pà-vinf-acte	le complément, introduit par la préposition à, est un verbe à l'infinitif ou un acte

Les mots du dictionnaire sont définis, avec ces attributs, entrés en clair à l'aide d'un éditeur de texte, dans des fichiers regroupant les mots par familles syntaxiques ou sémantiques. Puis une phase d'analyse relit tous ces fichiers, et écrit le tout dans un fichier trié par ordre alphabétique : le dictionnaire automatisé, dont auquel que je vous explique présentement et qu'il est si joli que vous allez l'encadrer vite fait ou je frappe, non mais. Le problème consiste à ne pas multiplier les attributs utilisés, à conserver un mode de définition commun d'un bout à l'autre du dictionnaire et surtout à fusionner les définitions d'un même mot situées dans deux fichiers séparés, s'ils ont le même « num » (même entrée sémantique) (ou s'ils n'en ont pas).

1.3 Exemple de définitions

Pour compacter le texte, on représente ici le nom d'attribut et ses valeurs sur une seule ligne alors que dans le fichier c'est une ligne par attribut et une par valeur.

1.3.1 Des verbes

```
afficher V
    conjug=3
    org_a=affiche
    acte=affichage
    agent=afficheur
    adjectivable=affichable
    adjnonable=inaffichable
    syn=placarder,coller,apposer,faire étalage,affecter,proclamer
        publier,annoncer,rendre public,se piquer de
    ant=dissimuler,taire,cacher
    constr=Thi,Thip,psur/dans-i
affranchir    V,conjug=26
    num=1
    constr=Thh,cesclave
    syn=émanciper,rendre libre,libérer,donner la liberté,racheter,rédimer,délivrer
    ant=asservir,jeter en esclavage,soumettre,assujettir,subjurer
affranchir    V,conjug=26
    num=2
    constr=Thc
    syn=timbrer,acquitter la taxe,dégrever
affranchir    V,conjug=26
    syn=mettre au courant,éclairer,ouvrir les yeux,avertir,confier un secret
    num=3
    constr=Thh,c#esclave
enliasser    V,conjug=3
    lieu_in=liasse
    constr=Thc
débander    V,conjug=3
    lieu_out=bande
    constr=Thb;Pb
emprisonner    V,conjug=3
    lieu_in=prison
    constr=Thh;Thhp,pdans-lieu
```

1.3.2 Des noms propres

tintin npm
 isa=personnage
 op1=bd,dom=LIT
 auteur=hergé
 journal=tintin
 compagnon=milou, capitaine hadock, professeurournesol
 métier=reporter
milou npm, isa=chien, personnage
 op1=bd, dom=LIT
 compagnon=tintin
capitaine hadock Npm, isa=personnage
 op1=bd, dom=LIT
 compagnon=tintin
 expression="mille milliard de mille sabords"
asterix npm, isa=personnage
 op1=bd, dom=LIT
 pays=gaule
 auteur=gosciny, uderzo
 journal=spirou
obélix npm, isa=personnage
 op1=bd, dom=LIT
 pays=gaule
 compagnon=asterix
 expression="ils sont fous ces romains"
 métier=livrer, o :menhir
 caractéristique=fort
idéfix npm, isa=chien, personnage
 op1=bd, dom=LIT
 compagnon=asterix
napoléon npm, isa=empereur
 op1=, dom=HIS
einstein npm, isa=physicien
 op1=relativité, dom=PHI
Les attributs dom et op sont en train d'y être renseignés

1.3.3 Des noms

Soit par exemple à définir quelque jeux
on définit d'abord quelques ustensiles standards servant aux jeux :

dé Nm, op=jouer v N, dom=JEU
 forme=cube
 usage=v:indiquer, i:hasard, valeur :1/2/3/4/5/6
 règle=jetter/lancer, o :dé
terrain Nm, op=m qc sr N, dom=JEU
plateau Nm, op=m qc sr N, dom=JEU
 forme=rectangle/carré
échiquier Nm, forme=carré, Isa=plateau

op=m qc sr N,dom=JEU
 ustensilede=échec
 damier Nm,forme=carré,Isa=plateau
 op=m qc sr N,dom=JEU
 ustensilede=dames

 raquetteNf,op=prat v N,dom=SPO
 ustensile=projectile,filet,terrain
 règle=lancer/frapper/rattraper(o=projectile,i=raquette)
 but=rattraper(o=projectile,i=raquette)

Puis on définit quelques jeux :

* op=jouer à N,dom=JEU
 monopoly Nm
 ustensile= :dé,Nb :2
 plateau,carte:caisse de communauté,carte:chance,carte:titre d'achat)
 billet,maison,hotel
 but=v :amasser,o :biens
 v :acheter,o :terrain
 v :construire,o :maison/hotel)
 règle=v :payer,v :aller en prison,v :sortir de prison,v :passer par la case départ
 jeu de l'oie Nm,ustensile=:dé,Nb:2,plateau
 backgammon Nm,ustensile=plateau
 :dé,Nb:2
 règle=v:jouer,Nb:2
 syn=24 flèches,jacquet,tric-trac
 baguenaudier Nm,isa=casse tête
 but=v :séparer
 règle=v :jouer,Nb :1
 mariage Nm,ustensile=cartes
 but=v:rassemble,o:carte,Nb:2,q:identique
 règle=v:retourner,o:carte,Nb :2
 v:jouer,Nb :2
 bridge Nm,ustensile=:carte,Nb:52
 règle=v:jouer,Nb:4,v :poser,o :carte,v :remporter,o :pli
 but=v :respecter,o :annonce
 prise=v :comparer ,o :carte
 score=v:compter,o:pli
 tarot Nm,ustensile= :carte,Nb :78
 règle= v :jouer,Nb :3/5,v :poser,o :carte,v :remporter,o :pli
 prise=v :comparer,o :carte
 score=v :compter,o :valeur
 réussite Nf,ustensile= :carte,Nb :52
 règle=v :jouer,Nb :1
 but=v :réaliser,o :figure
 syn=patience
 casse tête Nm,but=v :réaliser,o :figure

puce Nf,ustensile=plectre,puce,nid
règle=v :faire sauter,o :puce,i :plectre
but=v :placer,o :puce,l=nid
badminton Nm,ustensile=raquette,filet,volant
règle=v :jouer,Nb :2/4,v :rattraper,o :volant,i :raquette

De même soit à définir quelques instruments de musique :

* dom=MUS,op=jouer d N

veuze Nf,Isa=cornemuse,contient=bourdon,embouchure
Région=Poitou
chabrette Nf,Isa=cornemuse,contient=micro bourdon,embouchure
Région=limousin
Ta=cabrette
cabrette Nf,Isa=cornemuse,contient=micro bourdon,soufflet
U=cabretaire
Formation(style=folklore)=accordéon,vielle,violon
Formation(style=musette)=accordéon
Joueur(ep=ancien)=jean bergheaud,antoine bouscatel,martin cayla
biniou Nm,Isa=cornemuse,contient=bourdon,embouchure
Adj=coz,bras
Région=bretagne,Formation=bombarde
U=sonneur
bombarde Nf
Isa=instrument-musique,matière=bois,son=anche double,contient :embouchure
Région=bretagne,Formation=biniou
U=sonneur,talabarder
cornemuse Nf,Isa=instrument-musique,son=anche
Région=répandu
Contient=poche,chalumeau,bourdon,micro bourdon
embouchure/soufflet
Classe=cabrette,grande cornemuse bouronnaise,biniou,gaïta,
cornemuse écossaise,gajdos,duda,zampogna,mezoued,
musette bechonnet,bohausac,musette,chabrette,volynka,biniou coz,
biniou bras,chieuve,veuze,cornemuse berrichone,small pipe,
uillean pipe,bag pipe,doedelsakpfeife
U=sonneur,cornemuseux

et quelque bitos :

* Isa=chapeau,op=vêtir N,dom=HAB,op2=coif

borsalino Nm,pop,forme=tronc de cone,Contient=bord,couleur=gris
casquette Nf,matière=laine,forme=plat,Contient=visière
béret Nm,couleur=noir/bleu/rouge,forme=plat,matière=laine
sombrero Nm,région=mexique,couleur=clair
forme=tronc de cone,Contient=bord
turban Nm,région=inde,couleur=clair,matière=tissus
fez Nm,région=afrique du nord,couleur=rouge,matière=laine/feutre
forme=tronc de cone

bicorne Nm,U=académicien,polytechnicien,couleur=noir

1.3.4 Des adjectifs

* Adj,gradué

élégant	valorisation=favorable,champ=beauté
mirobolent	excès,valorisation=défavorable,champ=beauté
préoccupant	valorisation=défavorable,champ=agent,peur
effrayant	excès,valorisation=défavorable,champ=agent,peur
rassurant	valorisation=favorable,champ=agent,-peur (ou confiance)
courageux	valorisation=favorable,champ=attitude,-peur
intrépide	excès,valorisation=favorable,champ=attitude,-peur
lâche	valorisation=défavorable,champ=attitude,peur
alezant	Régent=cheval,champ=couleur,syn=fauve
rougeâtre	De(Adj=rouge,lien=atténué),champ=couleur
crèmeux	De(N=crème,lien=contenir/comme),champ=consistance

1.3.5 Des expressions figées ou variables : expv, expadv, expnm, ...

Les expv se comportent comme des verbes, et ont donc une construction des compléments

à l'oeil

expadv

syn=gratuit,genre=pop

oeil de boeuf

expnm

Obéir au doigt et à l'oeil

expv

constr=Th;Thp,pà-h

syn=obéir

Tourner de l'oeil

expv

constr=Ah

syn=s'évanouir,genre=pop

Faire de l'oeil

expv

constr=Thp,pà-h

1.4 Reconstitution de la forme racine d'un mot

Un mot d'une phrase ne peut pas être recherché tel que dans le dictionnaire où il n'est entré que sous sa forme de base (infinitif, masculin, singulier). il faut donc reconstituer les racines possibles pour la forme du mot cherché. Cela se fait en analysant le mot en commençant par les dernières lettres et en utilisant une table de finales possibles. cette table donne pour chaque finale possible d'une part le remplacement à effectuer pour obtenir la racine tant désirée et d'autre part les caractéristiques de la dérivation : pluriel, féminin, verbe(temps et personne). dans cette table, les finales sont inversées (dernière lettre en tête), et une finale peut apparaître plusieurs fois en tant que telle ou en finale d'une autre finale.

voici un extrait de cette table :

s		plur
s	re	1-1,1-2,8-1,18-1
		(rendS --> rendRE
		1-1 : 1ère personne du temps 1 : présent de l'indicatif
		1-2 : 2ème personne du temps 1 : présent de l'indicatif
		8-1 : 1ère personne du temps 8 : passé simple
		18-1: 1ère personne du temps 18: impératif présent)
s	tir	1-1,1-2,8-1,18-1
sa	er	3-2
sardnei	enir	4-2
		(4-2 : 2ème personne du futur
		vIENDRAS --> vENIR)
sardu	uloir	4-2
se		fem,plur
se	e	plur
se	er	1-2,5-2
se	ir	1-2,5-2

se	oir	5-2
secirt	teur	fem

Cette table est construite automatiquement à partir d'un fichier contenant les 94 conjugaisons possibles des verbes (dictionnaire Larousse des verbes) et d'un fichier des terminaisons féminin pluriel (le bon usage de Grevisse) et en en supprimant les doublets.

Pour accélérer la recherche et diminuer la place prise par la table, celle ci est organisée en 26 arbres, chacun décrivant toutes les finales possibles pour les mots se terminant par une lettre donnée. On parcourt l'arbre correspondant au mot analysé en progressant d'un noeud à chaque lettre. un noeud n'appartient qu'à un seul arbre et contient la liste des descriptions de la finale constituée des lettres précédentes et la liste des noeuds décrivant les lettres suivantes possibles.

D'où plusieurs racines pour chaque mot avec pour chacune la catégorie grammaticale, le genre, le nombre et (ou) le temps. avec le mot "portes" , on obtient ainsi les racines possibles :

portes	nom propre ou mot non dérivé
porte	nom ou adjectif, pluriel
portere	verbe,1-1,1-2,8-1
portetir	verbe,1-1,1-2,8-1
port	adjectif, féminin pluriel
porte	nom ou adjectif, pluriel
porter	verbe,1-2,5-2
portir	verbe,1-2,5-2
portoir	verbe,5-2

Pour chacune de ces racines possibles pour le mot, on vérifie alors que le mot existe, et qu'il se dérive bien en le mot de la phrase..

1.5 mécanismes de dérivation syntaxique et sémantique

si un mot racine doit être éliminé, on pourrait chercher à reconstituer une racine minimale qui se dérive en la racine cherchée à l'aide des préfixes et des suffixes, lesquels se répartissent en deux groupes : les suffixes syntaxiques d'une part, dont le rôle est de changer la catégorie d'un mot, mais qui n'ont pas de sens :

verbe	able	-> adjectif, qui peut être		
adjectif	ement	-> adverbe		
adjectif	ité	-> nom		
verbe	ailler	-> verbe, diminutif péjoratif		
verbe	asser	-> verbe, diminutif péjoratif		
verbe	onner	-> verbe, diminutif		
verbe	eler	-> verbe, diminutif		
nom	iser	->	verbe,	transformation
verbe	ation	->	nom,	action
verbe	aison	->	nom,	action
verbe	age	->	nom,	action
verbe	eur	->	nom,	agent
verbe	oir	->	nom,	instrument
verbe	ard	->	nom,	péjoratif
nom	ique	->		adjectif
nom	isme	->		nom
nom	iste	->		nom
nom	ien	->		adj
nom	elet	->	nom,	diminutif
nom	cule	->	nom,	diminutif
verbe	ence	->		nom
nom	el	->		adj
contre				
anti				
in				
de				
é				

Et	les	suffixes	ou	préfixes	porteurs	de	sémantique	:
	culteur	->		nom,		qui	cultive	
	tomie	->			nom,		couper	
	phile	->		nom,		qui	aime	
	mètre	->	nom,		instrument	de	mesure	
	bibli		livre					
	bar		pression					
	therm		température					

Le mécanisme serait exactement le même qu'avec les dérivations de base concernant les conjugaisons, le féminin et le pluriel.

Le programme ne gère pas ces dérivations préfixales et suffixales.

1.6 Principe d'analyse d'une phrase.

La reconnaissance d'une phrase se fait alors comme suit :

- Lecture de la phrase
- Extraction de tous les mots, séparés par les caractères spéciaux : espace, virgule, point, point virgule, parenthèse, tiret, ...
Les caractères spéciaux étant conservés pour la suite de l'analyse.
- Pour chacun des mots de la phrase, reconstitution des racines, entrées possibles dans le dictionnaire.
- Pour chaque mot recherche de chaque racine reconstituée dans le dictionnaire et élimination de celles qui ne peuvent se dériver en le mot cherché.
- Si le mot commence par une majuscule, et n'est pas précédé d'un article, c'est un nom propre et on élimine du dictionnaire ce qui n'est pas un nom propre. dans notre grammaire, la France est traitée comme un nom commun, Pierre et Paris comme un nom propre, la pierre comme un nom – (j'ai abusivement confondu « nom propre » et pas d'article et négligé le fait qu'il n'y a qu'une France, encore qu'il y a bien une (la) France des campagnes et une (la) France des villes, sans compter celle des quartiers dits populaires, qui, il est vrai, n'est plus tout à fait la France mais la nouvelle France)

On obtient ainsi la liste des racines des mots de la phrases et pour chacune les règles de dérivation produisant la forme du mot tel qu'il apparait dans la phrase.

- Si le mot n'est pas dans le dictionnaire, on le rajoute. S'il commence par une majuscule c'est un nouveau nom propre, sinon c'est un nouveau nom. Il n'y a pas de nouveau verbe.
Un nouveau nom est déclaré comme répondant oui à tous les critères de verbe.
Un nouveau nom propre est réputé être soit une ville, soit un humain
- On lit alors les définitions syntaxiques et sémantiques de chaque racine dans le dictionnaire, et on les représente sous forme de règles prolog.
- On lance le noyau prolog utilisant ces règles prolog construites dynamiquement et une grammaire du français écrite en prolog.

Dans la 1^{ère} version, le programme, en turbo pascal d'abord puis en delphi, lisait les fichiers .txt contenant les différents champs sémantiques, fusionnait le tout, mais sans gérer la polysémie (il regroupait tout ce qui concerne une racine en une seule entrée) et permettait de faire tout un tas de recherches et de contrôle des attributs, dans le but de vérifier la cohérence et l'uniformité des définitions.

Le couplage avec le noyau prolog a été rajouté en produisant en fin de l'analyse lexicale un fichier .pro contenant les règles prolog décrivant les racines possibles des mots de la phrase.

Dans la version actuelle, le tout, y compris le noyau prolog, a été réécrit en langage perl, puis en python, avec un interface graphique.

Tous les fichiers .txt originaux ont été fusionnés en un seul fichier : tout.txt

La polysémie a alors été rajoutée par les fichiers excel LVF, DEM, DEP, LEF (verbes, noms communs, noms propres, expressions), lesquels sont exportés au format .txt puis transformés sous la même forme que les fichiers txt originaux : lvf.txt, dem.txt, dep.txt, lef.txt

Les constructions des verbes correspondant à chaque entrée du LVF et du LEF sont en cours d'être renseignées, seuls quelques verbes ont leur construction décrite, les autres ont conservé la description du LVF original, insuffisante pour ce que je voulais faire.

Les descriptions syntaxiques en particulier les catégories des mots outils, les numéros de conjugaisons des verbes, ... contenus dans le fichier tout.txt sont indispensables au fonctionnement, mais les attributs sémantiques contenus dans le fichier tout.txt doivent encore être travaillés et exportés sous une forme ou une autre vers les fichiers LVF, DEM, DEP et LEF, les seuls à gérer la polysémie.

Deux fichiers supplémentaires contiennent l'un les relations entre les mots d'une famille de mots (derivés.txt) et l'autre les définitions du sens des mots (difs.txt) il faudrait les combiner dans lvf1.xlsx et dem1.xlsx, actuellement ils sont rajoutés sans tenir compte des homophones.

Le répertoire dico contient :

Répertoire dic

Répertoire lvf : les fichiers txt représentant le dictionnaire utilisé par le programme

dem.txt

lvf.txt

lef.txt

dep.txt

tout.txt

derivés.txt

difs.txt

Répertoires outils, themes, txt : anciens fichiers fusionnés en tout.txt

Répertoire others.txt : autres thesaurus à exploiter

Répertoire divers : anciens programmes

Répertoire lvf – voir ci après

Répertoire doc : divers documents décrivant lvf, dem, ...

fichiers représentant le programme dont :

fran.pl : grammaire du français en prolog

dico.pl : le programme (les .pm en sont les modules)

Le répertoire dico/lvf contient

corpus lvf.txt	fichier test de phrases, verbe conjugué
dem1.txt	export txt du dem
dem1.xlsx	source du dem
dep1.txt	export txt du dep
dep1.xlsx	source du dep
lef1.txt	export txt du lef
lef1.xlsx	source du lef
lvf1.txt	export txt du lvf
lvf1.xlsx	source du lvf
lvf1orig.txt	lvf originel
lvf.pl	transforme lvf1.txt en dic/lvf/lvf.txt
lef.pl	transforme lef1.txt en dic/lvf/lef.txt
dem.pl	transforme dem1.txt en dic/lvf/dem.txt
dep.pl	transforme dep1.txt en dic/lvf/dep.txt

documents divers, liste de valeurs d'opérateurs

doc	
doc dom.txt	
doc domdem.txt	
doc domlvf.txt	
doc nomenc dem.xlsx	
doc op1dem.txt	
doc oplvf.txt	
docold opdem.txt	
doc hierarchie domdem.txt	
doc list opdem.txt	
todo adj	choses à faire
todo.txt	choses à faire

utilitaires spécifiques au dictionnaire

critere.pl	affiche les noms de dem ou les noms propres de dep ayant certains critères param=-h -critere critere mot type -mot mot critere type(n v a) -nompropre mot critere type(n v a) -cmp crit1 crit2 -list critere -cmp critere1 critere2 type(n v a) Pour op, le critère est à mettre sans espace
critv.pl	affiche les critères d'un verbe – param=infinitif
lvfderivation.pl	dérive les verbes du lvf selon le fichier original

modifdem.pl	récupère un champ dans une sauvegarde
modiflvf.pl	récupère un champ dans une sauvegarde
mysearch.pl	recherche un mot dans les anciens fichiers
	mysearch mot donne le nom de fichier et la ligne complète
	ou mysearch mot file donne le nom de fichier
where.pl	creé un fichier xls à partir de tout.txt si le mot n'est pas déjà dans un xls
	param=n/np/v/adj/adv,interj/ono/titre

utilitaires généraux

coldem.pl	extrait des colonnes de dem1.txt - param=liste de nom-colonne
collvf.pl	extrait des colonnes de lvf1.txt - param=liste de nom-colonne
cut.pl	couper les lignes à une string et garder la fin triée
	param=nom-fichier string
cut2space.pl	couper les lignes à ' ' et garder la fin triée
grep.pl	recherche dans le contenu d'un fichier
	param=nom-fichier string
mysort.pl	trie sans doublon un fichier – param=nom-fichier - doublon de trier.pl
trier.pl	trie sans doublon un fichier – param=nom-fichier

Une proposition peut être :

Une affirmation

Sujet, verbe, compléments

Un ordre à l'impératif

Impératif, compléments

Une consigne à l'infinitif

Infinitif, compléments

Un groupe nominal, avec ou sans article, le nom est une nominalisation d'un verbe

Une question (dans ce cas la proposition peut manquer de sujet ou de complément ou remplacé par qui/quoi)

Proposition ? avec éventuellement verbe-t-il

Est-ce que proposition ?

Où/Quand/comment proposition avec éventuellement verbe-t-il

Que/quel/à quel proposition

Un complément peut être direct, un groupe nominal

ou indirect, une préposition et un groupe nominal

Pierre donne une bague à sa jolie fiancée

Le sujet est un groupe nominal

Un groupe nominal c'est :

Un déterminant un nom, des adjectifs éventuels et une relative éventuelle

Un pronom personnel

Un pronom démonstratif (cela plait aux français)

Un pronom indéfini (chacun aime le chocolat)

Un quantitatif de les (beaucoup des vaches produisent du lait)

Un groupe verbal à l'infinitif (manger des pommes rend gai)

Une proposition (ce que dit Mélenchon inquiète le bon peuple)

Un déterminant c'est

Un article (le, la, un

Un pronom démonstratif demandant une relative (celui

Un quantitatif sans article (plusieurs, ...)

Les constructions

Le fait que

beaucoup des (quantitatif_deles avec ou sans article)

La majorité des (nombre_deles)

Le programme en version perl, utilisant myprol, est en :

```
cd F:\pierre\prolog\dico
```

```
>perl dico.pl
```

La dernière version, en python , utilisant swu-prolog est en :

```
cd f:\users\pierre\tspy
```

```
>python dico.pl
```

Dans un premier temps, analysons une proposition affirmative, c'est à dire :

sujet, verbe, complément direct, préposition, complément, préposition, complément

Choisir l'onglet Dialogue

puis Proposition

le chat aime la souris

L'analyseur lexical reconnaît les racines possibles : le, chat, aimer, la, sourire, souris et en déduit les règles prolog suivante concernant l'analyse syntaxique :

declarer racine sourire (le verbe et le nom)

```
lvf(sourire,01,direct,dcrit(h),"","").
```

```
dem(sourire2,PSY,fpreuveN,base,,,m,,nonanimé,expression rieuse)
```

```
dem(sourire,2,masculin,_y).
```

```
constr_verbe(sourire,direct,,intran).
```

```
constr_verbe(sourire,pro,,intran).
```

```
verbe(souris,sourire,passésimple,singulier,2).
```

```
verbe(souris,sourire,présent,singulier,1).
```

declarer racine souris (5 sens du nom : l'animal, l'outil informatique, la souris d'agneau, la jeune femme)

```
dem(souris1,MAM,Aqnutri,omni,,,f,,animal,muridé,rongeur)
```

```
dem(souris,1,fÚminin,_y).
```

```
dem(souris2,PSY,fpreuveN,base,,,m,,nonanimé,sourire)
```

```
dem(souris,2,masculin,_y).
```

```
dem(souris4,INF,facvN,ins,,,f,,nonanimÚ,pour repérer sur écran)
```

```
dem(souris,4,fÚminin,_y).
```

```
dem(souris5,SOC,Nqappartà,fam,,,f,,humain,femme,fille)
```

```
dem(souris,5,fÚminin,_y).
```

```
dem(souris3,ANA,organeN,org,,,f,,nonanimé,de gigot mouton)
```

```
dem(souris,3,féminin,_y).
```

```
impératif(souris,sourire,2).
```

```
verbe(souris,sourire,présent,singulier,2).
```

```
participepassé(souris,sourire,masculin,pluriel).
```

```
conjug(sourire,avoir).
```

```
verbe(souris,sourire,passésimple,singulier,1).
```

```
nom(souris,souris,_x,_y).
```

declarer racine la (le nom, le pronom et l'article)

```
dem(la3,LIN,costrN,pron,,,f,,nonanimé,3e pers complément fém)
```

```
dem(la,3,féminin,_y).
```

```
dem(la1,MUS,jouerN,base,,,m,,nonanimé,note dans game)
```

dem(la,1,masculin,_y).

pronom_anté(la,la,_x,singulier).

nom(la,la,_x,singulier).

article(la,défini,féminin,singulier).

declarer racine chat (3 sens du nom : l'animal, le processus de dialogue internet)

dem(chat2,MAR,constrNd,base,,,m,,nonanimé,trou de chat)

dem(chat,2,masculin,_y).

dem(chat1,MAM,crifeuler,base,,,,,animal,félin domestique)

dem(chat,1,_x,_y).

dem(chat3,LOQ,direpN,base,,,m,,nonanimé,entre internaute)

dem(chat,3,masculin,_y).

nom(chat,chat,masculin,singulier).

declarer racine manger (5 sens du verbe)

lvf(manger,02,direct,dcrit(h),dcrit(c,aliment),",",").

lvf(manger,02,direct,dcrit(h),",",").

lvf(manger,03,direct,dcrit(b),dcrit(op1= fruc ou b ou aliment),",",").

lvf(manger,05,direct,dcrit(h),dcrit(c,non aliment),",",").

lvf(manger,06,direct,dcrit(c,dom=CHM ou dom=GEG ou dom=GEL ou dom=MTO ou liquide),dcrit(c),",",").

lvf(manger,07,direct,dcrit(b),dcrit(dom=TEX ou dom=MEN ou dom=PAP),",",").

lvf(manger,07,direct,dcrit(b),dcrit(c,dom=TEX ou dom=MEN ou dom=PAP),",",").

lvf(manger,08,direct,dcrit(b,dom=ENT),dcrit(h),",",").

lvf(manger,09,direct,dcrit(h),dcrit(op1=com ou op1=industr ou h et dom=ECN),",",").

lvf(manger,12,direct,dcrit(h),dcrit(i,op1=arg),",",").

lvf(manger,13,direct,dcrit(h),dcrit(h),",",").

lvf(manger,14,direct,dcrit(h),dcrit(a,dom=MES),",",").

lvf(manger,16,pro,dcrit(h),",",").

dem(manger1,SOM,aliN,base,,,m,,nonanimé,nourriture)

dem(manger,1,masculin,_y).

verbe(mange,manger,prÚsent,singulier,3).

verbe(mange,manger,subjprÚsent,singulier,1).

impératif(mange,manger,2).

verbe(mange,manger,prÚsent,singulier,1).

verbe(mange,manger,subjprÚsent,singulier,3).

declarer racine le (le pronom et l'article)

dem(le1,LIN,costrN,base,,,m,,nonanimé,3e pers complément masc)

dem(le,1,masculin,_y).

nom(le,le,masculin,singulier).

article(le,défini,masculin,singulier).

pronom_anté(le,le,masculin,singulier).

Ce qui donne comme résultat 2 interprétations possibles :

_resu=prop(nom(commun(chat,1,,),det(article,dÚfini,masculin,singulier)),verbe(manger,03,direct,","prÚsent,3,affirmatif,,),compls(nom(commun(souris,1,,),det(article,dÚfini,fÚminin,singulier)),,))

L'animal chat mange l'animal souris

_resu=prop(nom(commun(chat,1,,),det(article,dÚfini,masculin,singulier)),verbe(manger,03,direct,,"",prÚsent,3,affirmatif,,),compls(nom(commun(souris,3,,),det(article,dÚfini,fÚminin,singulier)),,))

L'animal chat mange la viande de souris d'agneau

Toutes les autres interprétations ont été éliminées par les critères attachés aux sujet et compléments du verbe manger

Rajoutons une proposition relative, qui va filtrer la souris animal :

Le chat mange la souris qui ronge la carotte

2 – Pour aller plus loin : Grammaire de texte

L'étape suivante pourrait être d'élaborer une grammaire de texte, et de récupérer un texte depuis un pdf. Un module préalable à l'analyse prolog réalise une découpe d'un texte en phrases. C'est donc un prétraitement du texte pour le « normaliser », sans utiliser la grammaire ni le dictionnaire. Une phrase se termine par un point, un point virgule, un point d'interrogation ou un point d'exclamation. Mais ce n'est pas évident car il manque souvent des points (aux titres de chapitres, de paragraphes, et même aux phrases lambdas) il faut tester la majuscule en tête de ligne, à moins que la ligne précédente se termine par une préposition, il faudrait tester que ce n'est pas un nom propre. Il faut supprimer les points entre les majuscules des initiales, mais il reste le point final quand il est l'unique point de l'initiale. Mais reste les RATP., les i.e., les vol. 1 p. 23, ... Et il y a aussi des points qui ne sont pas finaux (dans les nombres, dans les numéros de titre de paragraphe, dans les etc, Il faut aussi détecter les listes marquées par un tiret ou un numéro. Les notes de bas de page coupent les phrases n'importe comment, il faut les déplacer. Le problème des guillemets, tirets, virgules, parenthèses sera traité par l'analyse grammaticale. Cela soulève des tas de cas particuliers.

un texte commence par un titre, un auteur, et peut être divisé en parties, chapitres, introduction, préface, postface, scènes, notes, annexes, bibliographie, table des matières, et index ou être un simple contenu

une partie, chapitre, scène ou annexe contient le libellé « partie » ..., un numéro et éventuellement un titre, puis un contenu

une préface contient le libellé « préface », éventuellement un auteur, et un contenu

une introduction, postface, bibliographie, table des matières ou index ne contient que le libellé puis un contenu spécifique

un titre c'est une proposition ou un groupe nominal précédé ou non d'un tiret et ou d'un numéro et presque toujours non terminée par un point

un contenu est constitué de paragraphes, ou de scènes

un contenu de bibliographie contient un titre, une liste d'auteur, éditeur, ville, année et description (des phrases)

un paragraphe est constitué d'un éventuel numéro, un éventuel titre et des phrases

il peut ne pas y avoir de phrases, c'est une énumération de points

le paragraphe peut être un poème ou une chanson

une scène c'est un titre, une description de lieu (un groupe nominal) et une liste de répliques

une réplique c'est un nom de personnage, une éventuelle indication de jeu (une relative), et des phrases

une phrase c'est une proposition précédée ou suivie de circonstants puis d'un point, un point d'interrogation ou un point d'exclamation.

chaque circonstant est constitué d'un conjonction et d'une proposition ou d'une préposition et d'un groupe nominal, ou un groupe nominal seul

RIDEAU

le 2ème courtisan rate la sortie et se retrouve tout surpris
sur l'avant-scène.

I am a poor lonesome body.

Bibliographie

- Duponiac, M ,4891, "l'espace dual"
revue des adorateurs du citron vert
Vol 11,N° 2,Pommé sur le doubs
- Duarehsiob'ed, M ,1984, "tétrabologie des milieux hétéro-aqueux"
in Le petit écho de la mode, N# 458
2ème édition revue et diminuée
- Fotrep,Mc G ,1981, "about something interresting"
Cambridge university press
- Ramirez, S. ,Dellarobia,L ,1882, "quando caliente el sol, es muy bien"
Cielo Argentino editor
- Ssorg, M. ,1984, "yes but it is not the case at that time, godam"
in nineth international disjoint conference -Kolinda,New Zeland
- Weisel, K., 1983, "Ach so kartofelnen winderzinen"
Hermann, Karlsruhe, West Germany
- Limousinus, D., 33 av J.C., "de bonus qualitus"
in algorithmae chroniquae, Livre V - Rome. (à paraître)
- Kerviliec, A., 1979,"gwin ar c'hallaoued"
chouchen press , Kenavo
- Figaro, C., 1990,"lava mavinavette ava lava cavathavy"
friskies and wiskas , lyon, paris, issy , plaisir, paris