Travaux Dirigés de Programmation Logique par Contraintes

François FAGES et al.
(fages@dmi.ens.fr)

Mars 1996

Travaux dirigés 1 : Prolog

L'objectif de la session est de se familiariser avec le langage de programmation Prolog.

Exercice 1 (Interrogation d'une base de données) Le but de cet exercice est de poser différentes requêtes pour exploiter la base de données généalogiques `~fages/public/famille.pl`.

1. Traduire les questions suivantes en Prolog :
   - Est-ce-que Robert est un homme ?
   - Est-ce-que Robert est le père de Joel ?
   - Qui est le père de Benjamin ?
   - Quels sont les hommes qui ont un frère ?
2. Donner les règles définissant les relations `frere`, `soeur`, `grand_pere`, `grand_mere`, `oncle`, `tante`, ..., `ancetre`.

Exercice 2 (`member`)

1. Définir le prédicat `member(X,L)` vrai si `X` est un élément de la liste `L`.
2. Vérifier que `member(X,L)` est utilisable pour énumérer les éléments d'une liste.
3. Examiner quelques traces d'exécution.

Exercice 3 (Concaténation)

1. Définir le prédicat `append(A,B,C)` vrai si la liste `C` est la concaténation des listes `A` et `B`.
2. Vérifier que `append(A,B,C)` peut également énumérer les décompositions binaires de `C`.
3. Examiner quelques traces d'exécution.

Exercice 4 (`reverse`)

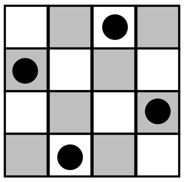
1. Définir en utilisant la concaténation, le prédicat `reverse(A,B)` vrai si `A` et `B` sont les images renversées de la même liste.

2. Évaluer la complexité de `reverse` en examinant quelques traces.

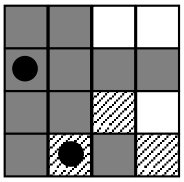
3. En déduire une version linéaire `reverse_lin(A,B)`.

4. Exprimer le prédicat `palindrome(L)` vrai si `L` est sa propre image renversée.

Exercice 5 (N-reines) Le problème des N-reines consiste à placer N reines sans prises sur un échiquier $N \times N$. Cela signifie que deux reines ne peuvent être placées sur une même ligne, colonne ou diagonale.



Une solution pour 4 reines



Construire une solution

On peut modéliser le problème à l'aide d'un prédicat `reines(N,L)` où `N` est la taille de l'échiquier et `L` la liste des positions des reines dans chaque colonne.

1. Ecrire un programme qui génère les positions possibles et teste la condition de prise.

2. Ecrire un programme qui teste la condition de prise au plus tôt pendant l'énumération.

Travaux Dirigés 2 : Parcours d'arbres

Cette session examine le principe de résolution SLD et en particulier les moyens de modifier l'ordre de sélection des sous-buts à prouver. Ceci sera utilisé pour le premier projet de la mineure qui consiste à réaliser une interface en Langage Naturel d'interrogation d'une base de données.

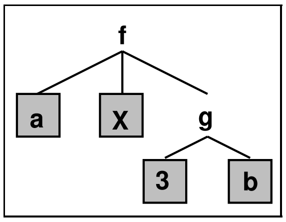
Exercice 6 *(Evaluateur arithmétique)*
Ecrire un prédicat `eval(X,V)` vrai si `V` est égale à la valeur du terme `X` considéré représentant une expression arithmétique formée avec les symboles binaires `plus`, `moins`, `mult`, `div` et les constantes numériques.

```
| ?- eval(plus(2,mult(3,4)),V).
V = 14 ?
| ?- eval(plus(div(3,(moins 3)),4), V).
V = 3.0 ?
```

Exercice 7 *(Feuilles d'un arbre)*

1. Ecrire un prédicat `feuilles(T,L)` vrai si la liste `L` correspond à la liste ordonnée de gauche à droite des feuilles de l'arbre (du terme) `T` quelconque. Les feuilles sont les constantes et les variables libres (utiliser les prédicats `functor/3` et `arg/3` pour décomposer les termes, et `var/1` pour tester si une variable est libre).

```
| ?- feuilles(f(a,X,g(3,b)),L).
L = [a,X,3,b] ? ;
no
```



2. Programmer le prédicat `feuilles` sans concaténation de liste, en utilisant un accumulateur.

Exercice 8 *(Co-routines)*
Modifier le prédicat `feuilles` pour qu'il ne collecte que les feuilles correspondant à des constantes et qu'il bloque la collecte des feuille sur les variables tant qu'elles ne sont pas instanciées. (Utiliser le prédicat de co-routinage `freeze/2`).

```
| ?- Y=h(c),feuilles(f(a,X,Y),L), X=g(b).
L = [a,b,c],
X = g(b),
Y = h(c)
```

Projet 3: Interrogation d'une base de données en français

Le but de ce projet est de réaliser une interface en langage naturel, en français, permettant d'interroger la base de données généalogiques du TD 1 `fages/public/famille.pl`. Pour cela, nous définissons d'abord une grammaire d'un sous ensemble du français, et écrivons l'analyseur correspondant. Ensuite, nous associons à la forme syntaxique produite par l'analyseur une forme sémantique qui permet via prolog d'interroger la base de données sur les liens de parenté.

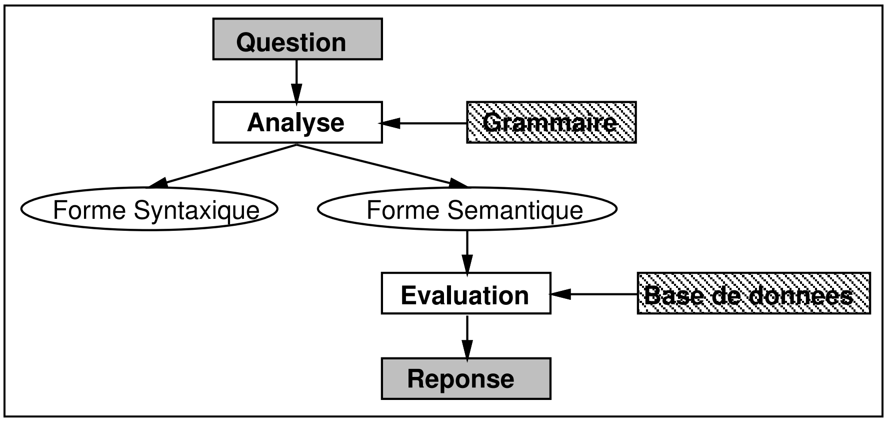


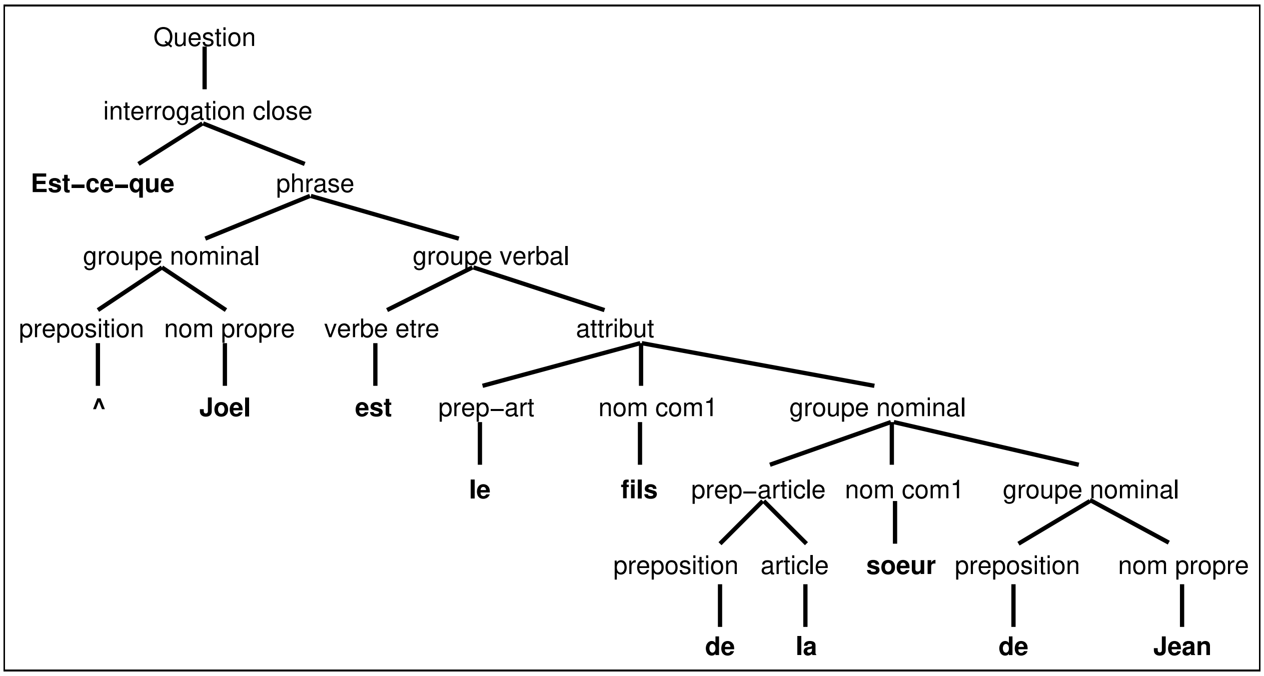
FIG. 0.1 - *Interrogation d'une base de données en Langage Naturel*

Un exemple complet

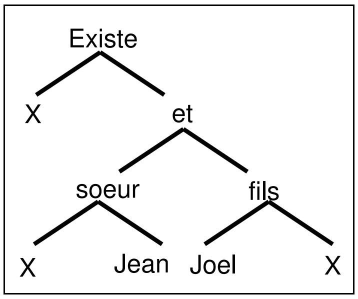
Voici sur un exemple les différentes structures que nous générons pour l'interrogation à partir de la phrase suivante:

Phrase `Est-ce-que Joel est le fils de la soeur de Jean`

Arbre syntaxique



Arbre sémantique



Question `est_ce_que(existe(X,et(soeur(X,jean),fils(joel,X))))`

Requête Prolog

```
| ?-reponse(est_ce_que(existe(X,et(soeur(X,jean),fils(joel,X))))).
 R= oui;
```

Analyse Syntaxique

Analyser une phrase consiste à construire sa forme syntaxique à partir de la grammaire qui définit le langage. Si les règles syntaxiques de la grammaire ne permettent pas la construction de cette forme, la phrase sera dite syntaxiquement incorrecte.

Définir une grammaire consiste à donner un ensemble de syntagmes possibles et un ensemble de règles. Ces règles définissent les juxtapositions correctes des structures syntaxiques ainsi les accords en genre et en nombre (dans le cas du français).

L'objectif de l'**analyse** est de construire (au moins) un arbre syntaxique à partir d'une phrase et d'une grammaire.

Comme le montre l'exemple d'arbre syntaxique précédent, par définition, la liste des feuilles de l'arbre syntaxique correspond à la liste d'entrée.

L'analyseur

Il existe plusieurs techniques d'analyse. Nous utilisons une technique qui a l'avantage de mettre au premier plan l'arbre syntaxique de la phrase analysée. D'autres techniques, bien que plus efficaces en analyse, sont plus compactes et ne permettent pas une approche très pédagogique.

L'analyseur est défini par la clause

```
analyse(L,X) :-
          feuilles(X,L),
          arbre_syntaxique(X).
```

où L est la phrase à analyser et X son arbre de syntaxe abstraite Le prédicat `feuilles` du TD 2 impose que les feuilles de l'arbre syntaxique correspondent à la phrase à analyser. Le mécanisme de coroutinage utilisé dans `feuilles` a pour effet de contrôler la construction de l'arbre syntaxique à partir de la phrase à analyser.

La grammaire

La règle de grammaire ci-dessous est composée de trois syntagmes, `<phrase>`,`<groupe nominal>` et `<groupe verbal>`. Elle déclare qu'une phrase en français est composée d'un groupe nominal suivi d'un groupe verbal.

```
<phrase> ::= <groupe nominal> <groupe verbal>
```

Une instance de l'utilisation de cette règle sera réalisée en prenant:

`<phrase>` = "Le frère de Pierre est le cousin de Jean"

`<groupe nominal>` = "Le frère de Pierre"

`<groupe verbal>` = "est le cousin de Jean."

Une telle règle s'écrit directement en Prolog comme une règle de construction de l'arbre syntaxique.

```
arbre_syntaxique(ph(GN,GV)) :-
        groupe_nominal(GN),
        groupe_verbal(GV).
```

L'annexe   donne un exemple de grammaire indépendante du contexte d'un sous ensemble du français. Les règles d'accord dépendent du contexte et ne font pas partie de cette grammaire. Elles seront directement introduites dans le programme Prolog à l'aide de **traits** indiquant la fonction grammaticale, le genre et le nombre. Ainsi, pour la règle précédente nous écrirons en fait le programme prolog suivant:

```
arbre_syntaxique(ph(GN,GV)) :-
         groupe_nominal([sujet,Nombre,Genre], GN),
         groupe_verbal([sujet,Nombre,Genre], GV).
```

où `[Sujet,Nombre,Genre]` représente la liste de traits, et les variables GN, GV représentent respectivement les sous-arbres du syntagme nominal et du syntagme verbal. Les seuls traits que nous considérons sont:

Fonction grammaticale : sujet (`suj`) ou complément d'objet direct (`dir`), indirect (`ind`), de nom (`de`).

Genre : masculin (`masc`), féminin (`fem`).

Nombre : singulier (`sin`), pluriel (`plu`).

Le lexique

Bien que l'analyse morphologique représente généralement un des problèmes majeurs dans le traitement des langages naturels nous ne nous y intéressons pas. Nous donnons un lexique complet c'est-à-dire comportant toutes les formes lexicales. L'accès se fait par l'intermédiaire des traits.

```
verbe_etre([suj,sin,G],etre(X1)) :- etre(X1,X2).
verbe_etre([suj,plu,G],etre(X2)) :- etre(X1,X2).

etre(est,sont).

nom_commun0(homme,hommes).
......
adjectif(beau,beaux,belle,belles).
......
article_def([C,sin,mas],art(le)) :- dif(C,de),dif(C,ind).
article_indef([C,plu,G],art(des)) :- dif(C,de).
......
preposition([de,N,G],prep(de)).
preposition([ind,N,G],prep(a)).
preposition([suj,N,G],prep([])).
preposition([ind,N,G],prep([])).
```

Exercice 9 Ecrire en Prolog le programme complet de la grammaire du français permettant d'interroger la base de données sur les liens de parenté.

Génération de la forme sémantique d'une phrase

Afin de réaliser l'interface d'une base de donnée, nous devons générer une structure qui permettra d'accéder à la base de donnée. Cette structure dépend fortement de l'outil utilisé pour coder la base de donnée. Dans notre cas se sera Prolog. Nous devons donc générer une structure sémantique interprétable par Prolog. De plus, cette structure sémantique doit être générée au cours de l'analyse afin de pouvoir à la fois utiliser la sémantique pour ôter certaines ambiguitées et accélérer son traitement. La règle principale se transforme de la façon suivante:

```
analyse(L,X,Y):-
        feuilles(X,L),
        arbre_syntaxique(X,Y).
```

où X représente l'arbre syntaxique dérivable de la grammaire, Y sa transformation sémantique, et L la phrase à analyser.

Arbre sémantique:

Nous avons choisi de représenter les structures sémantiques par un sous-ensemble de la logique du premier ordre bien adapté car Prolog est déjà lui même un sous ensemble de la logique du première ordre. En conséquence nous héritons d'une partie de l'interprétation Prolog. La construction de l'arbre sémantique se fait par composition des sémantiques des parties. Nous associons une structure sémantique à chaque structure syntaxique.

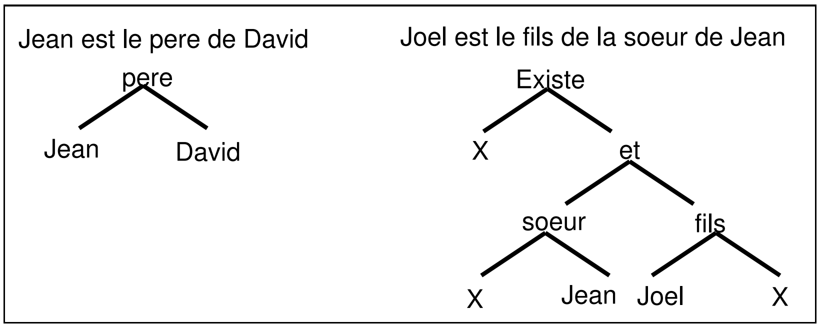


FIG. 0.2 - *Deux formes sémantiques*

Nous ne considérons que des formules closes, c'est à dire dont toutes les variables sont gouvernées par un quantificateur. Voici l'ensemble des formules nécessaires à l'interface:

- $p(i_1, i_2, ..., i_n)$ est vrai ssi $p(i_1, i_2, ..., i_n)$ est vrai.
- `non`(p) est vrai ssi p n'est pas vrai.
- `et`(p, q) est vrai si p et q sont vrai.
- `ou`(p, q) est vrai si p est vrai ou q est vrai.

- `existe`(X, p) est vrai ssi $p[X/i]$ est vrai pour au moins un individu i.

- `tout`(X, p) est vrai ssi $p[X/i]$ est vrai pour chaque individu i.

Le principe de composition des sémantiques est illustrée sur un exemple dans l'annexe .

Exercice 10 Transformer le programme Prolog d'analyse pour inclure la génération de la forme sémantique.

L'évaluateur:

Exercice 11 Ecrire le programme Prolog qui permet d'évaluer la valeur sémantique des formules générées comme forme sémantique d'une phrase. Cette valeur de vérité sera calculée par rapport à la base de données sur les liens de parenté. Nous définirons le but reponse(Y,R), ou Y est une forme sémantique et R la réponse.

```
| ?-reponse(est-ce-que(pere(pierre,gerard)),R).
R = oui.
| ?-reponse(quel(X,pere(pierre,X)),I).
X = gerard,
I = gerard,
```

Interrogation d'une base de données en français

Exercice 12 Regrouper les différents programmes réalisés précédents afin de réaliser l'interface en français de la base de données sur les liens de parenté avec le prédicat `interface(X,R)`, où X est la phrase en français et R la réponse. Eventuellemnt, enjoliver l'interface !

```
Question :
Est ce que pierre est le pere de gerard?
Reponse : oui

Question
Qui est le pere de gerard
Reponse : pierre

Question :
Quels sont les fils de jean?
Reponse : david;
Reponse : benjamin;
```

Grammaire

```
<question>               ::= <interrogation close> | <interrogation ouverte>

<interrogation close>    ::= Est-ce-que <phrase>

<interrogation ouverte> ::= Qui <groupe verbal>
                          |  <groupe pronom> <nom commun 1> <groupe nominal>

<phrase>                 ::= <groupe nominal> <groupe verbal>

<groupe nominal>         ::= <preposition> <nom propre>
                          |  <prep-article> <nom commun 0>
                          |  <prep-article> <nom commun 1> <groupe nominal>

<groupe verbal>          ::= <verbe etre> <attribut>
                          |  <verbe 0>
                          |  <verbe 1> <groupe nominal>
                          |  <verbe 2> <groupe nominal> <groupe nominal>

<attribut>               ::= <article indef> <nom commun 0>
                          |  <article def> <nom commun 1> <groupe nominal>

<prep-article>           ::= du | des | au | aux | <preposition> <article>

<nom propre>             ::= Pierre | Jean | Michel...
<nom commun 0>           ::= homme | hommes | femme | femmes
<nom commun 1>           ::= frere | freres | soeur | .... | tante | tantes

<verbe 0>                ::= vient | viennent
<verbe 1>                ::= aime | aiment
<verbe 2>                ::= donne | donnent
<verbe etre>             ::= est | sont

<article def>            ::= le | la | les
<article indef>          ::= un | une | des

<article>                ::= <article def> | <article indef>

<preposition>            ::= ^ | de | a         % ^ denote la chaine vide

<groupe pronom>          ::= Quel est le
                          |  Quels sont les
                          |  Quelle est la
```

| Quelles sont les

Forme sémantique

Prenons la phrase “Pierre est le frère d’un homme” nous allons construire pas à pas sa forme sémantique.

Partie de Phrase	Partie de sémantique	Remarque
PHRASE	F	formule logique
Pierre est le frere d’un homme	existe(i,et(homme(i),frere(Pierre,i)))	
NOM PROPRE	X	individu
Pierre	Pierre	
GROUPE NOMINAL	$<<X, F1>, F2>$	quantificateur
Pierre	$<<\texttt{Pierre}, F1>, F1>$	
GROUPE VERBAL	$<X, F>$	Propriété
est le frere d’un homme	$<X, \texttt{existe}(i, \texttt{et}(\texttt{homme}(i), \texttt{frere}(X, i)))>$	
ATRIBUT	$<X, F>$	Propriété
le frere d’un homme	$<X, \texttt{existe}(i, \texttt{et}(\texttt{homme}(i), \texttt{frere}(X, i)))>$	
GROUPE NOMINAL	$<<X, F1>, F2>$	quantificateur
d’un homme	$<<X, F1>, \texttt{existe}(i, \texttt{et}(\texttt{homme}(i), <X/i, F1>))>$	
ARTICLE	$<<X, F1>, <<Y, F2>, F3>>$	connecteur
un	$<<X, F1>, <<Y, F2>, \texttt{existe}(i, \texttt{et}(<X/i, F1>, <Y/i, F2>))>>$	
NOM COMMUN	$<X, F>$	propriété
homme	$<X, \texttt{homme}(X)>$	
NOM COMMUN	$<X, <Y, F>>$	propriété
frere	$<X, <Y, \texttt{frere}(X, Y)>>$	

Exemple de règle de grammaire transformée:

```
arbre_syntaxique(ph(X,Y), F) :-
                groupe_nominal([sujet,Nombre,Genre], X, [[U,F1],F]),
                groupe_verbal([sujet,Nombre,Genre], Y, [U,F1]).
```

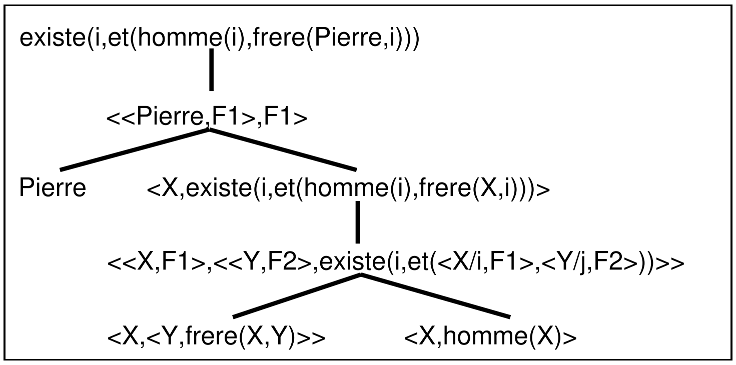


FIG. .3 - *Forme sémantique*

Travaux Dirigés 4 : Démonstration Automatique?

Exercice 13 *(Méta-interpréteur complet de clauses de Horn)*
Écrire un méta-interpréteur de Prolog utilisant :

1. un algorithme d'unification complet avec test d'occurrence,
2. une procédure de recherche complète par approfondissements successifs.

Exercice 14 *(Monoïdes nilpotents)*
On se propose de montrer que tout monoïde nilpotent est commutatif.

Un monoïde nilpotent est un ensemble muni d'une opération binaire, notée $*$, et d'un élément distingué, noté e, vérifiant les axiomes d'associativité

$$(x * y) * z = x * (y * z),$$

d'élément neutre

$$x * e = x, \qquad e * x = x,$$

et de nilpotence

$$x * x = e.$$

1. Exprimer la proposition sous la forme d'un ensemble de clauses de Horn et utiliser le méta-interpréteur pour démontrer automatiquement la proposition.
2. Représenter les preuves par des termes Prolog et effectuer les généralisations basées sur la preuve.
3. Vérifier que l'on peut omettre le test d'occurrence dans l'algorithme d'unification de la résolution SLD sur cet exemple. En déduire une (large) classe de problèmes de démonstration automatique pour laquelle le test d'occurrence est de toute évidence inutile.
4. Que donne l'exécution de Prolog sur cet exemple?

Exercice 15 *(Sous-groupes)*
Démontrer automatiquement que tout sous-ensemble d'un groupe additif, non vide et stable par soustraction, est un sous-groupe.

Travaux Dirigés 5 : Des contraintes !

On charge le système Meta(F) qui est une implantation complète de CLP($\mathcal{FD}$) au dessus de Prolog et de C. On s'intéresse dans ce TD aux seules contraintes de domaine `domain(X,[inf,sup])` et aux contraintes de diséquations `noteq(X,Y+c)` (notée en position infixe `X.=\ =Y+c`) ainsi que `alldifferent([X1,...,Xn])`. On utilisera également le prédicat d'énumération des valeurs possibles des variables `labeling([X1,...,Xn])`.

```
het: meta
SICStus 2.1 #9: Sat Jan 28 14:23:07 MET 1995

Metacompiler of Constraints over Finite Domains
Meta(F) - Version 3.58
Mon Feb 19 10:29:37 MET 1996
Copyright (C) 1992. Thomson-CSF Laboratoire Central de Recherches
| ?- domain([X,Y],[1,3]),noteq(X,Y).

domain(X, [1,3]),
X = X0,
noteq(X0, X1),
domain(Y, [1,3]),
Y = X1,
noteq(X0, X1) ?

yes
| ?- domain([X,Y],[1,3]),noteq(X,Y),labeling([X]).

X = 1,
domain(Y, [2,3]),
Y = X1 ? ;

X = 2,
domain(Y, [[1],[3]]),
Y = X1 ? ;

X = 3,
domain(Y, [1,2]),
```

```
Y = X1 ? ;

no
| ?- domain([X,Y],[1,2]),noteq(X,Y),labeling([X]).

X = 1,
Y = 2 ? ;

X = 2,
Y = 1 ? ;

no
| ?- domain([X,Y,Z],[1,2]),noteq(X,Y),noteq(Y,Z),noteq(Z,X).

domain(X, [1,2]),
X = X0,
noteq(X2, X0),
noteq(X0, X1),
domain(Y, [1,2]),
Y = X1,
noteq(X1, X2),
noteq(X0, X1),
domain(Z, [1,2]),
Z = X2,
noteq(X2, X0),
noteq(X1, X2) ?

yes
| ?- domain([X,Y,Z],[1,2]),noteq(X,Y),noteq(Y,Z),noteq(Z,X),labeling([X,Y,Z]).

no
```

Exercice 16 *(Échauffement)*

Les organisateurs d'un enseignement d'informatique disposent de 2 jours durant lesquels se tiendront onze cours (A,B,C,...,K) qui durent chacun une demi-journée.

Les cours des ensembles

AJ, JI, IE, CF, FG, DH, BD, KE, BIHG, AGE, BHK, ABCH, DFJ

ne peuvent se tenir simultanément (il existe au moins un élève inscrit à tous les cours de ces ensembles). Les organisateurs cherchent à déterminer un emploi du temps.

Ecrire une requête CLP($\mathcal{FD}$) pour résoudre ce problème

Exercice 17 *(Les N-reines revisitées)*

Dans toute la suite de ce TD nous allons étudier comment améliorer les performances du programme de placement de N-reines sur un échiquier $N \times N$ écrit en Prolog au TD 1, grâce à l'utilisation de contraintes dans CLP($\mathcal{FD}$).

Ecrire une version du programme en CLP($\mathcal{FD}$) qui pose toutes les contraintes de prise avant l'énumération des valeurs possibles des variables.

Comparer le temps d'exécution avec le programme précédent qui était limité en performances à $N \leq 13$ environ (on peut mesurer précisément le temps d'exécution en ms avec `statistics(runtime,[_,X]),queens(13,L),statistics(runtime,[_,Y]),Temps is Y-X.`).

Exercice 18 *(Heuristiques)*

Tester différentes stratégies d'énumération des solutions (cf. prédicat `labeling/3`).

Mesurer l'apport des stratégies qui fixent l'ordre de sélection des variables (on doit arriver à résoudre $N = 64$).

Mesurer l'apport des stratégies qui fixent l'ordre d'énumération des valeurs pour la variable selectionnée (résoudre $N = 128$)

(au delà pour $N = 200$ par ex. il est nécessaire d'augmenter la taille mémoire à 8Mo par exemple ce qui est possible dynamiquement avec `cs_reset,cs_setmemsize(8000).`)

Exercice 19 *(Modèles multiples)*

Une autre technique d'amélioration (de la robustesse) des performances consiste à introduire plusieurs modélisations concurrentes du problème à résoudre de façon à améliorer la détection des échecs (qui peuvent se voir dans un modèle et pas dans un autre), la propagation des contraintes, et même les choix heuristiques.

Les programmes précédents reposent sur une modélisation "primale" du problème dans laquelle les reines sont placées par exemple colonne par colonne. La modélisation duale consiste alors à considérer le placement des reines ligne par ligne.

Ecrire une version du programme qui exploite les deux modélisations concurrament. On utilisera des contraintes booléennes d'équivalence `impose(equiv(Xi.=J,Yj.=I))` pour relier les modèles entre eux :

```
| ?- domain([X,Y],[1,10]),impose(equiv(X.=3,Y.=4)),X=3.

X = 3,
Y = 4 ?

yes
| ?- domain([X,Y],[1,10]),impose(equiv(X.=3,Y.=4)),X.=\=3.

domain(X, [[1,2],[4,10]]),
X = X1,
domain(Y, [[1,3],[5,10]]),
Y = X2 ?
```

Comparer les performances de la modélisation primale-duale avec le programme primal sans heuristique.

Comparer la robustesse des performances avec le programme primal avec heuristiques face à l'ajout de nouvelles contraintes de répartition, comme par exemple $max\{X_i \mid 1 \leq i \leq N/5\} \leq 3 * N/5$.

Travaux Dirigés 6 : Ordonnancement

On s'intéresse dans ce TD aux contraintes d'inéquations $\sum a_i Xi \leq \sum b_j Y_j$ de CLP($\mathcal{FD}$), et au prédicat d'ordre supérieur d'optimisation `minimise(But,Cout)` qui permet de trouver les solutions au but `But` qui minimisent une fonction objectif, représentée par le terme `Cout`.

```
het: meta
SICStus 2.1 #9: Sat Jan 28 14:23:07 MET 1995

Metacompiler of Constraints over Finite Domains
Meta(F) - Version 3.58
Mon Feb 19 10:29:37 MET 1996
Copyright (C) 1992. Thomson-CSF Laboratoire Central de Recherches
| ?- domain([X,Y],[1,10]),X.>=Y+2.

domain(X, [3,10]),
X = X0,
supXYC(X0, X1, 2),
domain(Y, [1,8]),
Y = X1,
supXYC(X1, X0, -2) ? ;

no
| ?- domain([X,Y],[1,10]),X.>=Y+2,Y.>=X+1.

no
| ?- X.>=Y+2.

domain(X, [2,4294967292]),
X = X0,
supXYC(X0, X1, 2),
domain(Y, [0,4294967290]),
Y = X1,
supXYC(X1, X0, -2) ?

yes
| ?- minimise((B.>=A+5,C.>=B+2,D.>=B+3,E.>=C+5,E.>=D+5) , E).
```

```
Solution de cout 13

A = 0,
B = 5,
D = 8,
E = 13,
domain(C, [7,8]),
C = X3,
supXYC(X3, 5, 2) ? ;

no
```

On étudie l'application de ces contraintes dans CLP($\mathcal{FD}$) à la résolution de problèmes d'ordonnancement.

Exercice 20 *(Ordonnancement PERT et ordonnancement disjonctif)*
Le problème est de trouver un ordonnancement d'un ensemble de 10 tâches (voir le tableau ci-dessous), qui minimise la durée du projet. Les données du problème sont la durée de chaque tâche, et un ordre partiel sur les tâches. On définira une tâche supplémentaire `Fin` qui représentera la fin du projet.

Tâche	Durée	Tâches précédentes
A	7	
B	3	A
C	1	B
D	8	A
E	2	D,C
F	1	D,C
G	1	D,C
H	3	F
J	2	H
K	1	E,G,J

1. Écrire une requête CLP($\mathcal{FD}$) pour résoudre ce problème.
2. On rajoute la contrainte que les tâches D et C ne peuvent pas s'exécuter simultanément. Écrire une requête CLP($\mathcal{FD}$) permettant de trouver un ordonnancement optimal.

Exercice 21 *(Construction d'un pont)*

Le problème est de trouver un ordonnancement optimal d'un ensemble de 46 tâches nécessaire à la construction d'un pont (voir tableau suivant). En plus des contraintes de précédence classiques entre certaines tâches il y a des contraintes d'exclusion mutuelle entre les tâches partageant une même ressource. Par exemple le carterpillar est utilisé pour les tâches $V1$ et $V2$. Les tableaux ci-dessous indiquent les données du problème. Le fichier de données peut être récupéré dans `fages/public/PROJET/bridge_data.pl`.

1. Écrire un programme CLP($\mathcal{FD}$) permettant de trouver un ordonnancement optimal.
2. Utiliser la contrainte disjonctive `or(C1,C2)` de CLP($\mathcal{FD}$) plutôt que des points de choix, pour exprimer les contraintes d'exclusion mutuelle entre tâches et mesurer l'apport de ces contraintes. Attention, la propagation des contraintes `or` dans CLP($\mathcal{FD}$) n'étant pas complète il est nécessaire de procéder à une énumération (des valeurs des variables par `labeling`, ou bien des précédences par des points de choix)

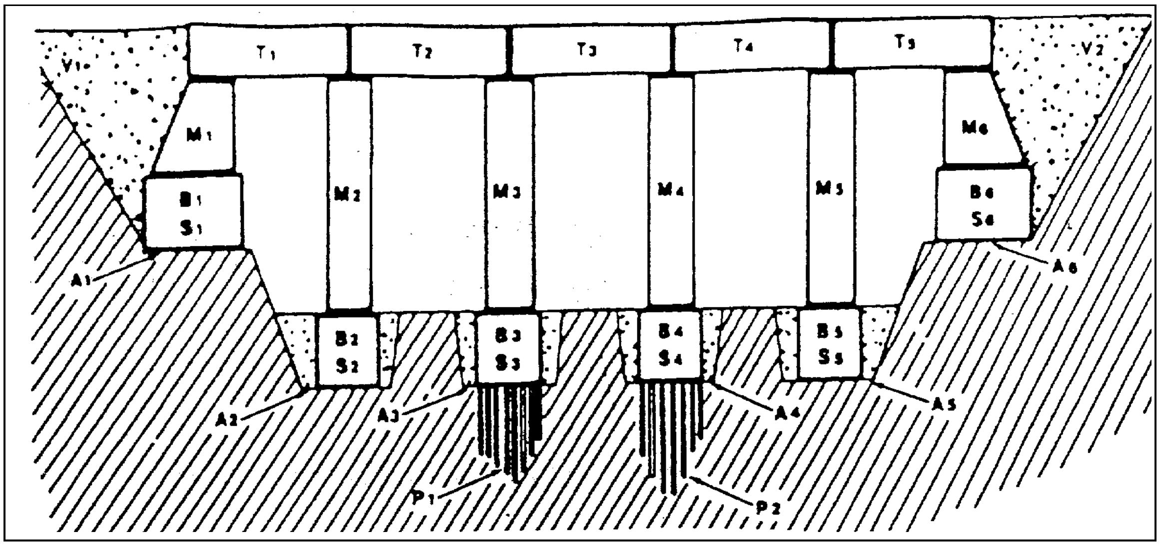


FIG. .4 - *Construction d'un pont*

- The time between the completion of a particular formwork and the completion of its corresponding concrete foundation is at most 4 days.
- There are at most 3 days between the end of a particular excavation (or foundation piles) and the beginning of the corresponding formwork.
- The formworks must start at least 6 days after the beginning of the erection of the temporary housing.
- The removal of the temporary housing can start two days before the end of the last masonry work.
- The delivery of the preformed bearers occurs exactly 30 days after the beginning of the project

N	Name	description	duration	resource
1	PA	beginning of project	0	
2	Al	excavation (abutment 1)	4	excavator
3	A2	excavation (pillar 1)	2	excavator
4	A3	excavation (pillar 2)	2	excavator
5	A4	excavation (pillar 3)	2	excavator
6	A5	excavation (pilar 4)	2	excavator
7	A6	excavation (pilar 5)	5	excavator
8	P1	foundation piles 2	20	pile-driver
9	P2	foundation piles 3	13	pile-driver
10	UE	erection of temporary housing	10	
11	S1	formwork (abutment 1)	8	carpentry
12	S2	formwork (pillar 1)	4	carpentry
13	S3	formwork (pillar 2)	4	carpentry
14	S4	formwork (pilar 3)	4	carpentry
15	S5	formwork (pilar 4)	4	carpentry
16	S6	formwork (abutment 2)	10	carpentry
17	Bl	concrete foundation (abutment 1)	1	concrete-mixer
18	B2	concrete foundation (pillar 1)	1	concrete-mixer
19	B3	concrete foundation (pillar 2)	1	concrete-mixer
20	B4	concrete foundation (pillar 3)	1	concrete-mixer
21	B5	concrete foundation (pillar 4)	1	concrete-mixer
22	B6	concrete foundation (abutment 2)	1	concrete-mixer
23	AB1	concrete setting time (abutment 1)	1	
24	AB2	concrete setting time (pillar 1)	1	
25	AB3	concrete setting time (pillar 2)	1	
26	AB4	concrete setting time (pillar 3)	1	
27	AB5	concrete setting time (pillar 4)	1	
28	AB6	concrete setting time (abutment 2)	1	
29	Ml	masonry work (abutment 1)	16	bricklaying
30	M2	masonry work (pillar 1)	8	bricklaying
31	M3	masonry work (pillar 2)	8	bricklaying
32	M4	masonry work (pillar 3)	8	bricklaying
33	M5	masonry work (pillar 4)	8	bricklaying
34	M6	masonry work (abutment 2)	20	bricklaying
35	L	delivery of the preformed bearers	2	crane
36	T1	positioning (preformed bearer 1)	12	crane
37	T2	positioning (preformed bearer 2)	12	crane
38	T3	positioning (preformed bearer 3)	12	cranc
39	T4	positioning (preformed bearer 4)	12	crane
40	T5	positioning (preformed bearer 5)	12	crane
41	UA	removal of the temporary housing	10	
42	Vl	filling 1	15	Caterpillar
43	V2	filling 2	10	Caterpillar
44	Kl	costing point 1	0	
45	K2	costing point 2	0	
46	PE	end of project	0	

Travaux Dirigés 7 : Voyageur de Commerce

Le but du TD est de résoudre en Meta(F) le problème du voyageur de commerce avec contraintes temporelles (TTSP pour "Timed Traveling Salesman Problem").

Soit un ensemble fini $\mathcal{V}$ de N sites. Pour chaque paire de sites v_1, v_2 on suppose donnée la distance t_{v_1,v_2} (pour simplifier on pourra considérer la **distance de Manhattan** $d((x,y),(x',y')) = |x-x'| + |y-y'|$ entre sites entrés avec leurs coordonnées entières). Chaque site est doté d'un ensemble de dates (fenêtre temporelle) auxquelles il peut être visité. Le problème consiste à trouver un chemin (un cycle) passant une fois par chaque site à une date dans la fenêtre associée, et qui minimise la date de retour au site départ/arrivée supposé donné.

On pourra constater qu'un TSP peut être ramené à un TTSP. En outre, un problème de même type mais pour lequel on n'impose pas que le site d'arrivée soit le site de départ (le chemin n'est pas nécessairement un cycle) peut se ramener à un TTSP. Par exemple, le problème de régulation du trafic aérien du cours peut également être modélisé sous forme d'un TTSP.

Exercice 22 *(Modèle d'ordonnancement)*
On veut modéliser un TTSP en spécifiant:

- pour chaque site, un ensemble de dates de visites possibles, voire une date précise.
- qu'un site puisse être visité avant un autre.

1. Modéliser le TTSP comme un problème d'ordonnancement en associant une date de visite $V_1, ..., V_n$ à chaque site, et en optimisant la date de retour à l'origine.
2. Expérimenter différentes heuristiques d'énumération.

Exercice 23 *(Modèle de séquence)*
On considère des TTSPs pour lesquels on spécifie également pour chaque site un nombre minimum de sites devant être visités auparavant.

1. Modéliser le TTSP comme un problème de construction d'une séquence, en associant à chaque site une position de visite, $P_1, ..., P_n$, dans la séquence.
2. Relier ce modèle au précédent.

3. Expérimenter différentes heuristiques.

Le fichier `fages/public/ttsp.pl` contient l'exemple suivant:

```
% L est une liste de sites (V,(x,y),[min,max],N) donnes avec leur coordonnees
% une fenetre temporelle de visite, ainsi que le nombre N de sites
% devant etre visites auparavant
% On considere la distance de Manhattan
% et une tournee a partir de l'origine (0,0).
% Les contraintes de precedence sont B<<C, B<<D, F<<I, F<<J

test(LV,Lpos):-
   L=[A,B,C,D,E,F,G,H,I,J],
A=(0,(0,0),[0,80],0),
B=(_,(5,5),[0,50],0),
C=(_,(5,9),[0,80],0),
D=(_,(8,1),[0,80],0),
E=(_,(6,9),[0,80],0),
F=(_,(1,1),[10,50],0),
G=(_,(9,1),[0,30],0),
H=(_,(4,9),[0,80],5),
I=(_,(9,0),[0,100],0),
J=(_,(1,0),[0,100],0),
   ttsp(L,[[B,C,D],[F,I,J]],LV,Lpos).

% LV = [0,10,40,25,39,18,26,41,27,53]
% Lpos = [1,2,8,4,7,3,5,9,6,10]

% avec  H=(_,(4,9),[0,80],0),
% LV = [0,10,14,27,17,36,28,15,45,37]
% Lpos = [1,2,3,6,5,8,7,4,10,9]
```