Outils Formels pour les Systèmes d'Information

Jean-Claude Ziswiler
Jean-Claude.Ziswiler@cui.unige.ch

FRI 30JAN2004 0900h Uni Dufour 408

Prolog

Prolog - Introduction

En Prolog, il y a des

Faits (facts)
Règles (rules)
Questions (queries)

Les faits et les règles forment une base de connaissance.

=> Valider des questions à partir des faits et des règles,

soit les questions sont des faits qui se trouvent **explicitement** dans la base de connaissance,

soit Prolog peut valider la question à partir des faits et des règles à l'aide de la **résolution**.

Prolog - Syntaxe

Exemple : parle(X) :- humain(X), not bebe(X).

signifie $\forall x$ (humain(x) $\wedge$ $\neg$ bebe(x) => parle(x))

,	$\wedge$	conjonction
;	v	disjonction
:-	<=	implication
not	$\neg$	négation

Les différents termes :

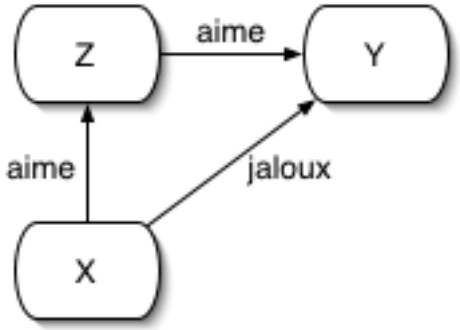
- Constantes (atomes, nombres)
- Variables
- Termes complexes

Prolog est **„case sensitive“.** Les mots qui commencent avec un caractère majuscule sont considérés comme des variables, les mots qui commencent avec un caractère minuscule sont des constantes.
Il ne faut jamais mettre des espaces entre un nom d‘un terme et les „()“.

Prolog - Variables

Exemple :

```
aime(othello,desdemona).
aime(desdemona,jago).
aime(romeo,julia).
aime(julia,romeo).

jaloux(X,Y) :- aime(X,Z), aime(Z,Y).

?- jaloux(othello,P).
```



Est-ce qu'il existe un individu P tel que Othello est jaloux ?

Est-ce qu'il existe des autres individus qui sont jaloux ?

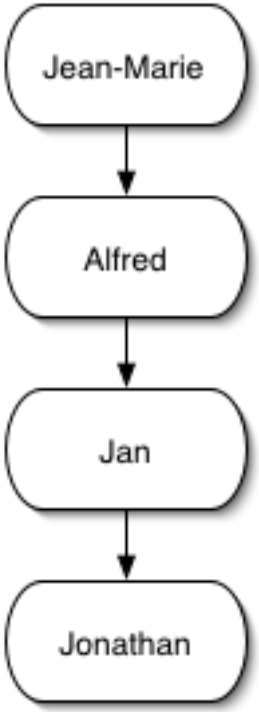
Prolog - Récursivité

Exemple :

```
enfant(alfred,jean-marie).
enfant(jan,alfred).
enfant(jonathan,jan).
```

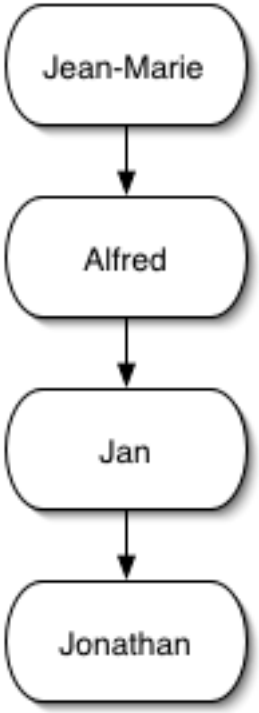
On va définir la relation „descendant“ :

```
descendant(X,Y) :-
    enfant(X,Y).
descendant(X,Y) :-
    enfant(X,Z),
    enfant(Z,Y).
descendant(X,Y) :-
    enfant(X,Z),
    enfant(Z,A),
    enfant(A,Z).
...
```



Prolog - Récursivité (cont.)

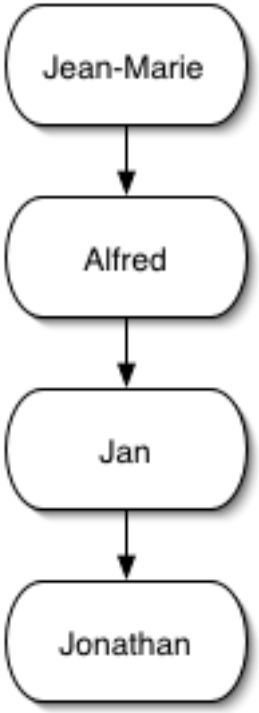
```
enfant(alfred,jean-marie).
enfant(jan,alfred).
enfant(jonathan,jan).

descendant(X,Y) :-                    (1)
    enfant(X,Y).
descendant(X,Y) :-                    (2)
    enfant(X,Z),
    descendant(Z,Y).

?- descendant(jonathan,jean-marie).
```



Tout d'abord, Prolog essaie à appliquer la première règle, mais cette information ne se trouve pas dans la base de connaissance. Donc, Prolog essaie la deuxième règle. On continue jusqu'on trouve une information mise explicitement dans la base de connaissance qui valide la première règle.

Il faut au moins deux clauses : Une clause de base qui peut valider la récursivité (terminer le boucle) et une clause qui contient la récursivité.

Prolog - Récursivité (cont.)




```
enfant(alfred,jean-marie).
enfant(jan,alfred).
enfant(jonathan,jan).

descendant(X,Y) :-                          (1)
    enfant(X,Y).
descendant(X,Y) :-                          (2)
    enfant(X,Z),
    descendant(Z,Y).

?- descendant(jonathan,jean-marie).

descendant(jonathan,jean-marie) :-          (2)
    enfant(jonathan,jan),
    descendant(jan,jean-marie).
descendant(jan,jean-marie) :-               (2)
    enfant(jan,alfred),
    descendant(alfred,jean-marie).
enfant(alfred,jean-marie).                  (1)
```

Prolog - Implémentations

Les implémentations „libres“ :

SWI-Prolog : (University of Amsterdam, www.swi-prolog.org)

Eclipse-Prolog (www.icparc.ic.ac.uk/eclipse)

Open Prolog (www.cs.tcd.ie/open-prolog) pour Apple Macintosh

PROPLOG

Les implémentations commerciales :

SICSTUS-Prolog (www.sics.se/sicstus)

Quintus-Prolog (www.sics.se/quintus)

Prolog - Implémentaions (cont.)

Créer un projet („home folder“ où les fichiers avec le code sont stockés).

Le code s‘écrit dans un éditeur de texte et est mis dans un fichier. GNU Emacs va permettre d‘écrire et compiler le code et aussi d‘effectuer des queries (avec SWI-Prolog). Open Prolog inclut un éditeur de texte.

Charger un programme (compilation du code).

Faire des requêtes (queries).

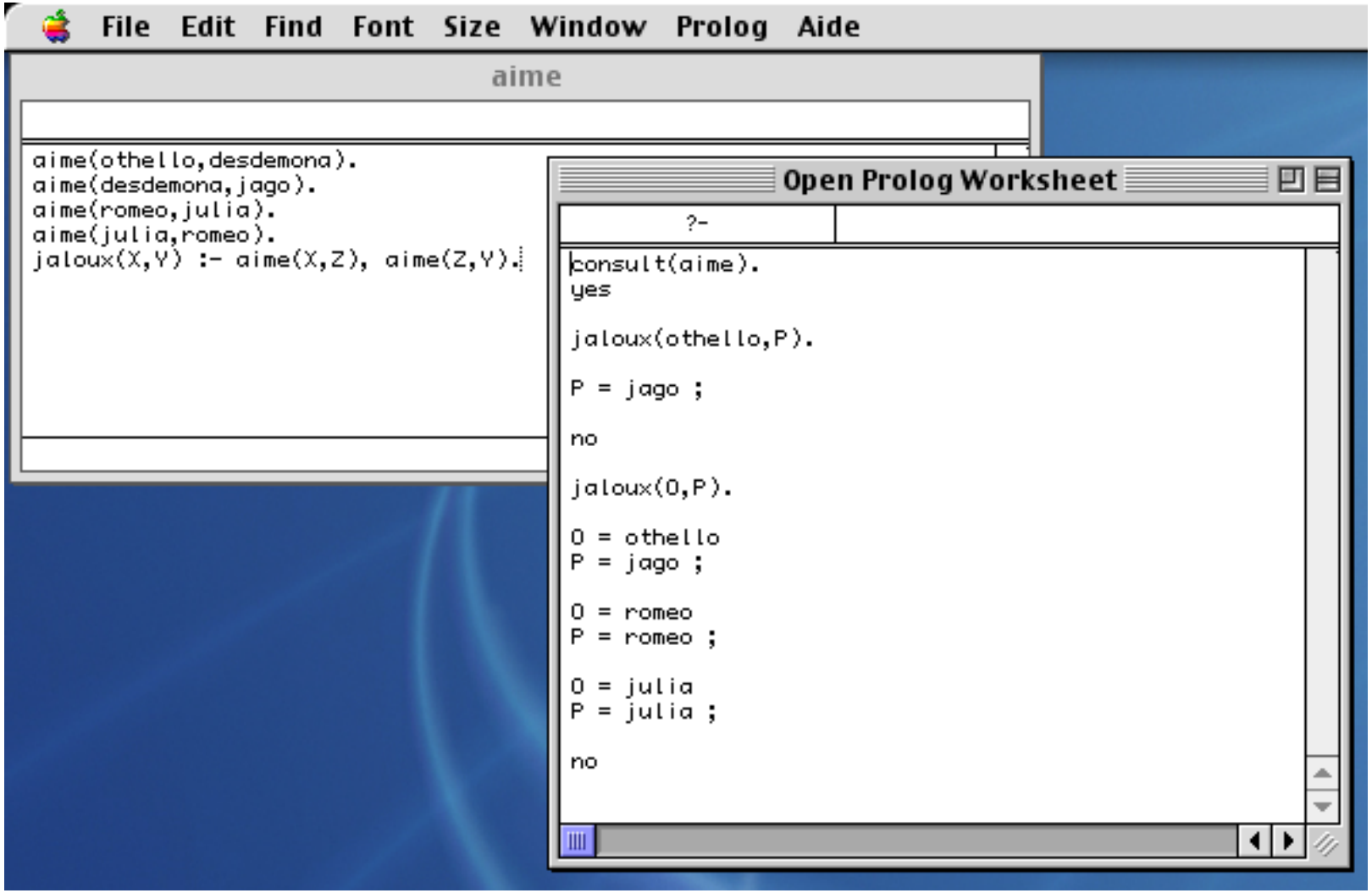
„Trace“ et „Dubug“ les requêtes („step-by-step“). Avec ces fonctions, on peut observer la résolution et corriger les erreurs dans le code.

```
trace(descendant(jonathan,jean-marie)).
```

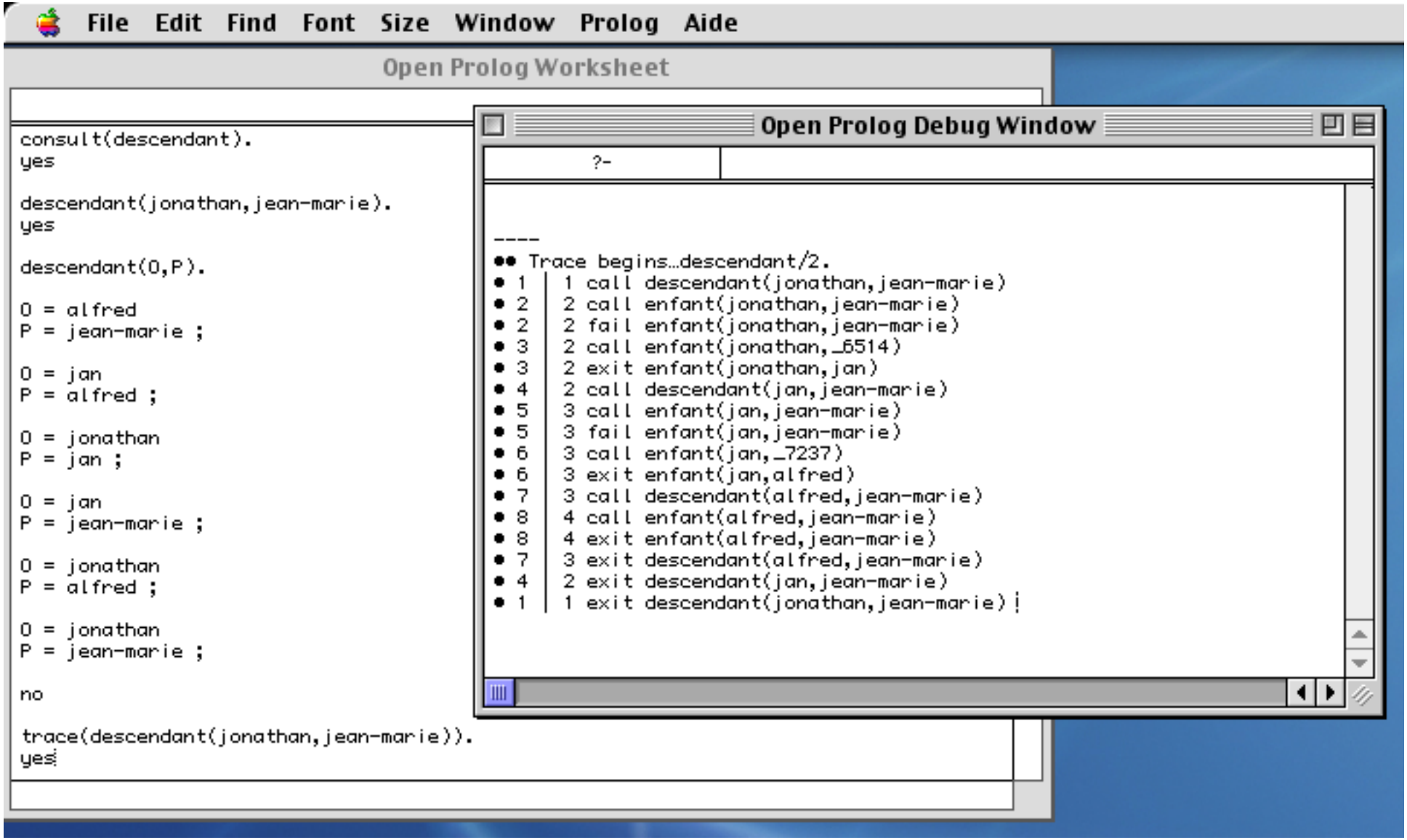
On peut aussi stocker la charge des programmes et les requêtes pour une utilisation future.

On peut mettre plusieurs programmes Prolog dans un seul fichier.

Prolog - Exemple variables



Prolog - Exemple récursivité



Série d'exercices n° 6

3 Modélisation

c) Etablissez une liste d'axiomes pour décrire les contraintes physiques de ce système. Par exemple :

$$\forall x \neg(\text{Cube}(x) \wedge \text{Sphere}(x))$$

Série d'exercices n° 6 (cont.)

„x n'est pas à droite de x" :

$\forall x \neg ADroite(x,x)$
$\forall x \forall y\ (ADroite(x,y) \Rightarrow \neg Egal(x,y))$
$\forall x \forall y\ (ADroite(x,y) \Rightarrow AGauche(y,x))$
$\forall x \forall y\ (AGauche(x,y) \wedge AGauche(y,z) \Rightarrow AGauche(x,z))$

„Si x est un cube, x n'est pas une sphère" :

$\forall x \neg(Cube(x) \wedge Sphere(x))$
$\forall x\ (\neg Cube(x) \vee \neg Sphere(x))$ (De Morgan)
$\forall x\ (Cube(x) \Rightarrow \neg Sphère(x))$ (Implication-ou)

En Prolog :

```
not aGauche(X,X). not aDroite(X,X).
not egal(X,Y) :- aDroite(X,Y).
aGauche(Y,X) :- aDroite(X,Y).
aGauche(X,Z) :- aGauche(X,Y), aGauche(Y,Z).

sphere(X) :- not(cube(X)).
```

Série d'exercices n° 6 (cont.)

Priorité des opérateurs (Rappel)

(cf. „Vers la logique des prédicats“, p. 11)

(∀,∃) avant (¬) avant (v, ∧) avant (=>, <=>)

Example:

∀x∀y aDroite(X,Y) => aGauche(Y,X)

Sans parenthèses, la clause est équivalent avec la suivante:

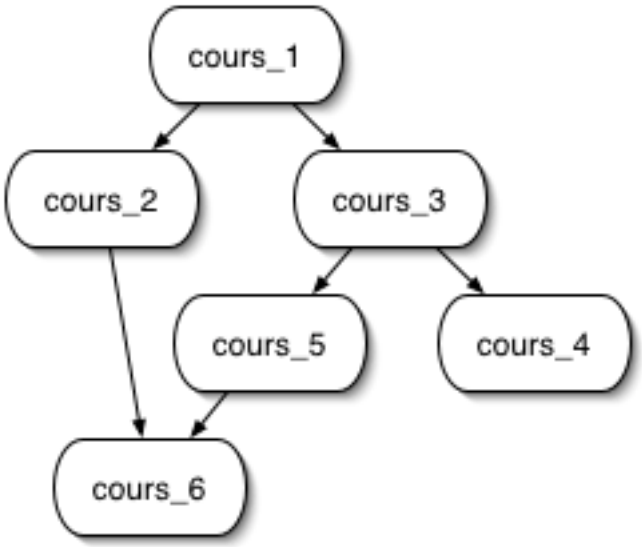
(∀x∀y aDroite(X,Y)**)** => aGauche(Y,X)

Par contre si on ajoute des parenthèses:

∀x∀y **(**aDroite(X,Y) => aGauche(Y,X)**)**

Récursivité: Example

Plan d'études avec pré-requis




```
pre_requis(cours_1,cours_2).
pre_requis(cours_1,cours_3).
pre_requis(cours_3,cours_4).
pre_requis(cours_3,cours_5).
pre_requis(cours_5,cours_6).
pre_requis(cours_2,cours_6).

pri(X,Y) :- pre_requis(X,Y).
pri(X,Y) :-
    pre_requis(X,Z),
    pri(Z,Y).

?- pri(X,cours_6).
```

(1) X = cours_5 ;
(2) X = cours_2 ;
(3) X = cours_1 ;
(4) X = cours_1 ;
(5) X = cours_3 ;
(6) no

Récursivité: Example (cont.)

Plan d'études avec pré-requis

```
peut(X,C) :- not interdit(X,C).
interdit(X,C) :- pre_requis(C,D), not reussi(X,D).
```

Références

„Einführung in die Programmiersprache PROLOG“. Uebungen zur Vorlesung „Künstliche Intelligenz I“ am Institut für Informationssysteme der Universität Hannover (Teile der Uebungen stammen aus Blackburn, Bos, Striegnitz, **„Learn Prolog Now!“**, University of Saarbrücken), Wintersemester 2003/2004. http://www.kbs.uni-hannover.de/Lehre/KI1/WS03/Uebungen/uebungen.html

Jan Wielemaker, **„SWI-Prolog 5.2 Reference Manual“**, University of Amsterdam, 2003. http://www.swi-prolog.org

Mike Brady, **„Getting Started With Open Prolog“**, 2003. http://www.cs.tcd.ie/open-prolog/Documentation/Getting_Started