Logique pour l'informatique

Jean-Yves Antoine

http://www.info.univ-tours.fr/~antoine

Logique pour l'informatique

Chapitre III – Logique des Prédicats du 1er ordre (LP1)

LOGIQUE DES PREDICATS (LP1) - Objectifs

3.1. Notions

3.1.1. Logique des prédicats : quel pouvoir de représentation par rapport à la LP ?
3.1.2. Quantificateurs et termes (variables, constantes)
3.1.3. Prédicats et fonctions
3.1.4. Forme propre : portée des variables
3.1.5. Forme clausale et forme normale : forme prenexe et forme de Skolem
3.1.6. Unification : substitution et application à la résolution (résolvante)
3.1.7 Interprétation en LP1 et domaine d'interprétation

3.2. Pratiques

3.2.1. Représenter un énoncé ou un problème en logique des prédicats et savoir quand cette logique est insuffisante
3.2.2. Dans des cas simples, donner directement les interprétations d'une fbf de la LP1, trouver directement le modèle d'une fbf.
3.2.3. Mettre sous forme clausale une fbf de la LP1
3.2.4. Montrer qu'un raisonnement est valide, ou qu'une formule est contradictoire à l'aide de la méthode de résolution (*englobe les 3.2.3 et 3.2.4*)

3.3. Approfondissement

3.3.1. Lien avec la démonstration mathématique

LOGIQUE DES PREDICATS DU 1er ORDRE

Exemple de raisonnement en LP1

Tout être humain a un père
Le grand-père de quelqu'un est le père du père de ce dernier
Jean est un être humain
Donc, Jean a un grand-père

prédicat : jugement élémentaire dont la valeur de vérité dépend de ses arguments. Tout prédicat a un nombre fixe d'arguments : arité.

Syntaxe LP1 : vocabulaire

connecteurs logiques	$\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$, etc..	
symboles de quantification	$\forall$: *tout* $\exists$: *il existe*	
constantes	symboles commençant par majuscule	*Jean,* $\mathcal{V}$, $\mathcal{F}$
variables	symboles commençant par une minuscule	*x*
fonctions	ident. commençant par un minuscule	*pere(x)*
prédicats	ident. commençant par une majuscule	*Humain(x)*

LP1 : SYNTAXE

Termes représentation des éléments du problème

constantes, variables
fonctions nom_fonction(terme1, ..., terme n)

Formules atomiques représentation des jugements de base

prédicats Nom_predicat(term1, ... , terme n)

Formules bien formées (fbf)

Toute formule atomique P est une fbf
Si P est une fbf (P) et ¬ P sont des fbf
Si P et Q sont des fbf P ∧ Q et P ∨ Q sont des fbf
P ⇒ Q et P ⇔ Q sont des fbf
Si P est une fbf et x variable
∀x (P) est une fbf
∃x (P) est une fbf

Priorités { ∃, ∀, ¬ } prioritaire sur {∧,∨} prioritaire sur {⇒,⇔}

Egalité `Paul = fils(Jean)` équivalent au prédicat : `Egal(Paul, fils(Jean))`

SYNTAXE LP1 : bons … et mauvais exemples

- ∀x (Humain(x) ⇒ Mortel(x))
- ∀x∀y ∀xy ((Humain(x) ∧ Humain(y)) ⇒ Amis(x,y))
- ∀x (Humain(x) ∧ Humain(y) ∧ Chien(Toto))
- ∃y (Amis(Paul,y))
- ∃y (Amis(Paul,Virginie))
- Fatigue(Paul)
- Fatigue(x)
- ¬∃y (Amis(Paul,y))
- ∃y ¬(Amis(Paul,y))
- ∀x (Aime(Virgine,x) ⇒ ¬Paul)
- ∀x (Humain(x) ⇒ Humain(frere(x)))
- ∀x (Humain(x) ⇒ Humain(frere(frere(x))))
- ∀x (Humain(x) ⇒ ∃y (Humain(y) ∧ Frere(x,y)))
- ∀x∃y ((Humain(x) ∧ Humain(y)) ⇒ ¬(Frere(x,y)))
- ∀x frere(x)
- ∀x∃y (Humain(Amis(x,y))

LP1 : VARIABLES LIBRES / LIEES

Portée portée d'un quantificateur : formule à laquelle il s'applique

Exemples $\exists x\ (\ Riche(x) \wedge Avare(x)\ )$

$\exists x\ Riche(x) \wedge \exists x\ Avare(x)$

Variable libre / liée

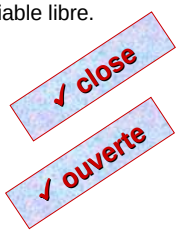
une variable x est dite variable **libre** (respectivement **liée**) d'un formule P ssi x a toutes ses occurrences en dehors (resp. en dedans) de la portée des quantificateurs de P

Formule close / ouverte

Une formule est dite **close** ssi elle n'a aucune variable libre.
On parle sinon de formule **ouverte**

Exemples $\forall x\ (Aime(Virgine,x) \Rightarrow \neg Paul)$

$\forall x\ (\ Humain(x) \wedge Humain(y) \wedge Chien(Toto)\ )$



LP1 : VARIABLES LIBRES / LIEES

Forme propre Ecriture d'une fbf où il n'y a aucun conflit de portée

Exemple $\forall x\ (\exists y\ Père(x,y) \wedge \exists y\ Mere(x,y)\ )$

Standardisation Mise sous forme propre d'une fbf

☑ **Variable dépendant de plusieurs portées**

Formule mal formée, renommer les variables dans les quantificateurs pour la rendre cohérente

Exemple $\forall x\ (\exists y\ Père(x,y) \wedge \exists x\ Mere(x,Jean)\ )$

☑ **Conflit variable libre et liée**

Renommer la variable libre

Exemple $\forall x\ (\exists y\ Père(x,y) \wedge Mere(x,y)\ )$

☑ **Variables liées de portées différentes**

Renommer les variables pour éviter toute erreur ultérieure

LP1 : VARIABLES LIBRES / LIEES

Portée portée d'un quantificateur : formule à laquelle il s'applique

Exemples $\exists x$ (Riche(x) $\wedge$ Avare(x))

$\exists x$ Riche(x) $\wedge$ $\exists x$ Avare(x)

Variable libre / liée

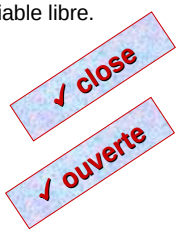
une variable x est dite variable **libre** (respectivement **liée**) d'un formule P ssi x a toutes ses occurrences en dehors (resp. en dedans) de la portée des quantificateurs de P

Formule close / ouverte

Une formule est dite **close** ssi elle n'a aucune variable libre.
On parle sinon de formule **ouverte**

Exemples $\forall x$ (Aime(Virgine,x) $\Rightarrow$ $\neg$Paul)

$\forall x$ (Humain(x) $\wedge$ Humain(y) $\wedge$ Chien(Toto))



CALCUL DES PREDICATS

Interprétation en LP1 **I = { $\mathcal{D}$, I_v, I_f, I_p }**

Domaine d'interprétation

$\mathcal{D}$ = {valeurs que peuvent prendre les termes}

Interprétation des variables I_v

application qui assure une valeur à toute variable

Interprétation des fonctions I_f

application qui associe à toute fonction d'arité n et à tout n-uplet de termes une valeur dans le domaine d'interprétation

Interprétation des prédicats I_p

application qui associe à tout prédicat d'arité n et à tout n-uplet de termes une valeur de vérité

Calcul des prédicats

$\forall x$ (P) vrai ssi P est vraie pour toute interprétation Iv(x)
$\exists x$ (P) vrai ssi il existe une interprétation Iv(x) tq P est vraie

Exemple $\forall x \exists y$ A(x,y) est-elle satisafaible, tautologique, contradictoire ?

Corollaire $\neg \forall x\ A \equiv \exists x\ \neg A$ $\neg \exists x\ A \equiv \forall x\ \neg A$

FORME CLAUSALE

Définition Idem LP

Mise sous forme clausale

Toute fbf de LP1 admet une forme clausale qui en est la conséquence logique. Contrairement à la LP,cette forme clausale n'est pas unique

Algorithme de normalisation

1) **Mise sous fnc** : idem LP, sauf que restent les quantifications
2) **Mise sous forme prenexe** : déplacement des quantifications en tête de fnc
3) **Skolemisation** : élimination des ∃
4) **Mise sous forme clausale** : élimination des ∀

FORME CLAUSALE(2)

Forme normale : formules d'équivalence

réduction de la portée des négations

$\neg\forall x\, A \equiv \exists x\, \neg A$ $\quad\quad$ $\neg\exists x\, A \equiv \forall x\, \neg A$

mise sous forme prenexe

$\forall x\, A \wedge \forall y\, B \equiv \forall x\, (A \wedge B)$	**$\exists x\, A \wedge \exists y\, B \equiv \exists x\, (A \vee B)$**
$\forall x\, A \wedge B \equiv \forall x\, (A \wedge B)$	$\forall x\, A \vee B \equiv \forall x\, (A \vee B)$
$A \wedge \forall x\, B \equiv \forall x\, (A \wedge B)$	$A \vee \forall x\, B \equiv \forall x\, (A \vee B)$
$\exists x\, A \wedge B \equiv \exists x\, (A \wedge B)$	$\exists x\, A \vee B \equiv \exists x\, (A \vee B)$
$A \wedge \exists x\, B \equiv \exists x\, (A \wedge B)$	$A \vee \exists x\, B \equiv \exists x\, (A \vee B)$

Forme de Skolem : remplacement des variables quantifiées existentiellement

- par une constante si la variable est dans la portée d'aucune autre quantification universelle.
- par une fonction f(x1, ... , xn) où les xi sont les variables correspondant aux quantifications universelles dans la portée desquelles se trouvait la variable

UNIFICATION

Substitution

On appelle **substitution** toute application σ de l'ensemble des variables V vers l'ensemble des termes T :

σ : V $\longrightarrow$ T

X σ (X) = nom_fonct(terme1, ..., terme n)
σ (X) = Constante
σ (X) = variable

Une **substitution** est dite **finie** ssi il n'existe qu'un nombre fini de variables X telles que σ (X) ≠ X. On représente σ alors par l'ensemble des couples [X → σ (X)] tels que σ (X) ≠ X

Substituée d'une fbf

Soit ϕ une fbf de LP1. On appelle **substituée** de ϕ par une substitution σ = [X → σ (X)] la fbf obtenue par remplacement de toute occurrence de la variable X dans ϕ par le terme substitué σ(X). Par extension, on note σ (ϕ) la substituée.

UNIFICATION (2)

Unification de deux fbf

Soient ϕ1 et ϕ2 deux fbf quelconques de LP1. On dit que ϕ1 et ϕ2 sont **unifiables** ssi il existe une substitution σ telle que σ (ϕ1) = σ (ϕ2). La substitution σ est alors appelée unificateur de ϕ1 et ϕ2.

Exemples

(S1) ≡ P(x) ∨ R(A) (S2) ≡ P(B) ∨ R(y)

⇨Formules unifiables. Un unificateur est : σ [x → B ; y → A]

(S1) ≡ P(x) ∨ R(A) (S2) ≡ P(B) ∨ R(B)

⇨Formules non unifiables.

RESOLUTION

Même principe qu'en LP, mais il faut ici faire l'unification des clauses pour pouvoir les identifier lors de l'application de la règle de résolution

Règle de résolution en LP1

Soient $\phi 1$ et $\phi 2$ deux clauses quelconques de LP1. On dit que $\phi 1$ et $\phi 2$ forment une **paire résolvable** ssi elles contiennent une paire opposée de formules atomiques ayant pour forme respective P(t1, ..., tn) et ¬P(t'1, ..., t'n) et qui peuvent être **unifiées** par un unificateur σ.

On appelle alors **résolvante** de $\phi 1$ et $\phi 2$, la clause :

$$\mathbf{res(\phi 1, \phi 2) = \sigma(\phi 1 \setminus \{P\}) \cup \sigma(\phi 2 \setminus \{\neg P\})}$$

Exemple

$(S1) \equiv P(x) \vee R(A)$ $\qquad$ $(S2) \equiv \neg P(y) \vee R(y)$

⇨ **unificateur** : $\sigma\,[x \rightarrow y]$ $\qquad$ **résolvante** : $R \equiv R(A) \vee R(y)$

Logique pour l'informatique

Chapitre IV – Calcul décidable et programmation logique

PROGRAMMATION LOGIQUE - Objectifs

4.1. Notions

4.1.1. Décidabilité
4.1.2. Décidabilité de la LP et de la LP1
4.1.3. Clauses de Horn
4.1.4. Langage Prolog : syntaxe
4.1.5. Langage Prolog : sémantique opérationnelle (stratégie de résolution)

4.2. Pratiques

4.2.1. Détecter des cas d'indécidabilité en LP1
4.2.2. Transformer en Prolog un raisonnement représenté en LP1 et réciproquement.
4.2.3. Représenter un problème simple en Prolog pur (syntaxe *Edimbourg*)
4.2.4. Faire l'arbre de résolution correspond à une requête Prolog
4.2.5. Utiliser l'environnement de programmation *SWI-Prolog*

4.3. Approfondissement

4.3.1. Savoir représenter et résoudre des paradoxes en LP1

DECIDABILITE

LOGIQUE DES PROPOSITIONS

La LP est décidable, c'est à dire que l'on peut toujours montrer en un nombre fini d'opérations si une fbf de la LP est valide ou contradictoire.

LOGIQUE DES PREDICATS DU 1er ORDRE

La LP1 est **indécidable**, c'est à dire qu'on ne peut pas construire d'algorithme de décision montrant en un nombre fini d'opérations qu'une fbf de la LP1 est valide ou contradictoire.

La LP1 est cependant **semi-décidable** : on peut construire des algorithmes de décision répondant en un nombre fini d'opérations ... si la formule est valide.

CLAUSES DE HORN

La LP1 réduite aux seules clauses de Horn (cf. infra) est **décidable**.

En pratique, les clauses de Horn ont un pouvoir d'expression suffisant pour des raisonnements " classiques ".

CLAUSES DE HORN ET LANGAGE PROLOG

CLAUSE DE HORN

On appelle **clause de Horn** toute clause de la LP1 ayant au plus une formule atomique (littéral) positive

Type 1	$\neg H1 \vee \neg H2 \vee ... \vee \neg Hn \vee C$
Type 2	C
Type 3	$\neg H1 \vee \neg H2 \vee ... \vee \neg Hn$

PROLOG

PROLOG est formé sur le sous-ensemble de la LP1 formé par les clauses de Horn.

Type 1: règles	*Implication : production de nouvelles connaissance* $H1 \wedge H2 \wedge ... \wedge Hn \Rightarrow C$
Type 2: faits	*Affirmation de connaissances supposées vraies a priori* C
Type 3: buts	*Question à laquelle on souhaite répondre par déduction* $C1 \wedge C2 \wedge ... \wedge Cn$?

LANGAGE PROLOG (2)

CLAUSES DE HORN	PROLOG
Règles $H1 \wedge H2 \wedge ... \wedge Hn \Rightarrow C$	*Règles* `C :- H1, H2, …, Hn.`
Faits C	*Faits* `C.`
Questions C ?	*Questions* `?- C.`
$C1 \wedge C2 \wedge ... \wedge Cn$?	`?- C1, C2, …, Cn.`

SYNTAXE PROLOG

Langage Prolog

U. Marseille (A. Colmerauer) & U. Edimburgh (**1975**)
Pas de syntaxe unifiée ; standard principal: **Prolog Edimburgh**

Termes

Atomes (constantes) : *objets définis du problème*

- identificateur commençant par une **lettre minuscule**
- **nombre**
- chaîne de caractères entre apostrophes (messages)

Variables : *objets indéfinis du problème*

- identificateur commençant par une **lettre majuscule**

Prédicats : *jugement ou relations logiques élémentaires*

- Nom prédicat commençant par une **lettre minuscule**
- Formule prédicative : **nom_pred(term1,…,termN)**
- **Arité** : nombre N d'arguments

SYNTAXE PROLOG

Clauses

Faits : *connaissances de base établies a priori*

`fait.` (formule prédicative quelconque)

Règles : *règle de manipulation de la connaissance*

```
conclusion :- premisse.
concl :- prem_1,..,prem_n.
```
(conjonction premisses)

Questions (buts) : *problème à résoudre*

```
?- but.
?- but_1, ... but_n.
```
(but simple : formule prédicative)
(conjonction de buts)

Clauses de Horn et conjonction

Conjonction de faits interdite

~~`fait1, fait2.`~~

```
fait1.
fait2.
```

Règle : conjonction de conclusions interdite

~~`concl1, concl2 :- premisse.`~~

```
concl1 :- premisse
concl2 :- premisse
```

SYNTAXE PROLOG : DANGERS

Portée des variables Prolog

- **Portée d'une constante :** totalité du programme
- **Portée d'une variable** : clause dans laquelle elle se trouve

Exemple

```
majeur(X) :- age(X ,Y ),superieur(Y ,18).
mineur(X) :- age(X ,Y ),inferieur(Y ,18).
```

variable différente

variable unique (partagée)

```
majeur(X):- age(X,A),superieur(A,18).
mineur(X):- inferieur(A,18).
```

variable différente

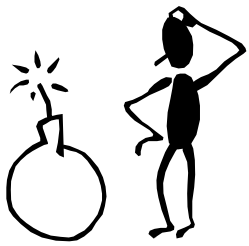
SYNTAXE PROLOG : DANGERS

Variables anonymes Utilisation délicate en cas de **variable partagée**

Exemples

```
majeur(X) :- age(X,Y), sup(Y,18).
majeur(X) :- age(X,_), sup(_,18).
```

anonymat problématique !



```
majeur(X) :- age(X,_Y), sup(_Y,18).
```

semi-anonymat !

PROLOG : SEMANTIQUE OPERATIONNELLE

Stratégie de résolution Prolog

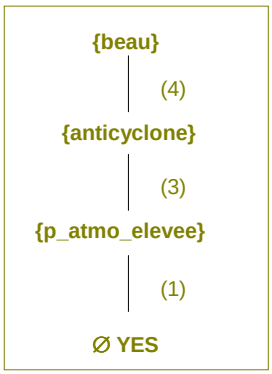
- **Chaînage arrière** résolution de buts
- **Profondeur d'abord** résolution complète du 1er but
- **Régime par tentative** choix 1ère clause et retour-arrière éventuel

Chaînage arrière

Résolution d'un but B Recherche d'une clause ayant B en conclusion.

⇨ **fait** B. Le but B est montré

⇨ **règle** B :- P1, ..., Pn Prémisses nouveaux but
B remplacé par P1, ... Pn



Exemple

```
(1) p_atmo_elevee.
(2) temp_elevee.
(3) anticyclone :- p_atmo_elevee.
(4) beau :- anticyclone.
```

But : `?- beau.`

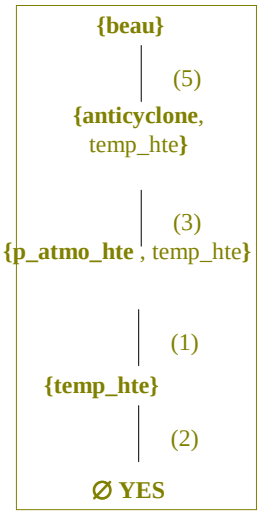
PROLOG : SEMANTIQUE OPERATIONNELLE

Profondeur d'abord

Si un but se décompose en plusieurs sous-buts :

B :- P1, ..., Pn

Prolog tente toujours de résoudre complètement le premier sous-but P1 avant de tenter de résoudre le suivant, et ainsi de suite



Exemple

```
(1) p_atmo_hte.
(2) temp_hte.
(3) anticyclone :- p_atmo_hte.
(4) isobare_hte :- p_atmo_hte.
(5) beau :- anticyclone, temp_hte.
```

But : `?- beau.`

PROLOG : SEMANTIQUE OPERATIONNELLE

Régime par tentative

Si plusieurs clauses peuvent a priori de résoudre le même but :

B :- P1, ..., Pp B :- R1, ..., Rr

L'interpréteur choisit toujours la **première clause** du programme (non encore essayée) : tentative prioritaire sur la 1ère clause.

En cas d'**échec** de la clause essayée, on effectue un **retour-arrière** jusqu'au dernier point de choix : on considère alors la 1ère clause utilisable suivante.

S'il ne reste aucune règle utilisable, on remonte au point de choix précédent.
Si aucune possibilité n'est envisageable au final : échec

Exemple

```
(1) inflation_zero.
(2) chomage.
(3) crise :- deflation.
(4) crise :- chomage.
(5) deflation :- inflation_zero, investissement_zero.
```

But : `?- crise.`

PROLOG : SEMANTIQUE OPERATIONNELLE

Résolution Prolog avec variables : unification

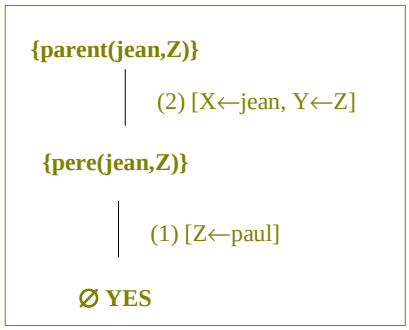
Résolution d'un but B par décomposition en sous-buts :

- Recherche d'une clause ayant une conclusion **s'unifiant** avec le but.
- Les **sous-buts** obtenus sont les **substitués**, par l'unificateur trouvé, **des prémisses** de la clause

Exemple 1

```
(1) pere(jean,paul).
(2) parent(X,Y) :- pere(X,Y).
```

But : `?- parent(jean,Z)`



Exemple 2

```
(1) mere(paul,jean).
(2) mere(sophie, jean).
(3) enfant(E,M) :- mere(M,E).
(4) enfant(E,P) :- pere(P,E).
(5) mere(ernestine,sophie).
(6) petit_enf(PE,GP) :- enfant(PE,P), enfant(P,GP).
```

PROLOG : SEMANTIQUE OPERATIONNELLE

Algorithme de résolution Prolog

Pour résoudre un but simple B, on cherche la première clause C du programme non encore utilisée dont la conclusion s'unifie avec B. Pour résoudre un but complexe B = {B1,...,Bn}, Prolog cherche à résoudre en premier B1 puis passe à B2 si B1 peut se résoudre, et ainsi de suite récursivement.

1 — **Si C est un fait**, alors le but est montré. S'il n'y a que ce but à résoudre, ARRET sur SUCCES. La réponse est le résultat de l'unification de B et C.

Si d'autres buts restent à montrer (**but complexe** par exemple) on continue la résolution (étape 1) sur les nouveaux buts obtenus par application de l'unificateur de B et C à l'ensemble des buts qui restaient à résoudre C.

2 — **Si C est une règle**, résoudre B revient à résoudre récursivement les sous-buts {H1,H2, ... , Hn} obtenus par application de l'unificateur de B et C à l'ensemble des premisses de C.

3 — **Si aucune clause** ne s'unifie avec B, on procède à un retour-arrière jusqu'à retrouver un ancien point de choix pour lequel il existe une règle non encore utilisée pour résoudre le but courant..

3.1. S'il existe une telle règle existe, nouvelle résolution à partir de l'étape 1.
3.2. Si aucune règle unifiable, alors ARRET sur ECHEC.

PROPRIETES DE LA RESOLUTION PROLOG

L'ordre des clauses **dans le programme et** l'ordre des but **dans les clauses** a une influence sur la résolution**.**

Logique pour l'informatique

Chapitre V — Programmation logique effective : récursivité et terminaison

PROGRAMMATION LOGIQUE - Objectifs

5.1. Notions

5.1.1. Récursivité
5.1.2. Terminaison des programmes Prolog (modes d'utilisation)
5.1.3. Structures de liste en Prolog

5.2. Pratiques et méthodes

5.2.1. Analyse : décomposition du problème en sous-problèmes
5.2.2. Gestion de la récursivité : gérer l'ordre des clauses et des sous-buts.
5.2.3. Validation pratique de la terminaison : jeux d'essais
5.2.4. Programmation avec listes

5.3. Approfondissement

5.3.1. Coupure (hors programme)

PROLOG : RECURSIVITE

Prédicat récursif en Prolog

Prédicat dont la définition fait appel (directement ou non) à lui-même

Exemple : `ancetre(A,Desc) :- parent(P,Desc), ancetre(A,P).`

Programmation récursive en Prolog

① **Décomposition récursive en sous-problème** — Ramener le problème, considéré à une complexité donnée, au même problème mais de complexité inférieure (principe de récurrence).

② **Cas particuliers** — Chercher les cas particuliers éventuels ne rentrant pas dans le schéma récursif.

③ **Cas d'arrêts** — Chercher les cas d'arrêts de la récursivité.

Remarque : Les cas d'arrêts sont des cas particuliers, mais certains cas particuliers peuvent ne pas correspondre à l'arrêt de la récurrence générale.

④ **S'assurer de la convergence de la récursivité** — En Prolog, cela revient à chercher les modes d'utilisation (cf. infra). Se prouve formellement.

⑤ **Programmer** et vérifier les modes d'utilisation: **jeux d'essais** complets

PROLOG : RECURSIVITE

Programation récursive : exemple

Problème — Ecrire un prédicat qui vérifie si un nombre est pair ou non. On suppose qu'on ne dispose que de prédicats arithmétiques de soustraction, d'addition (division inconnue...) et de comparaison (supérieur ou égal)..

Analyse et résolution du problème

⓪ **Représentation du problème** : - variables nombre N
- prédicats `pair/1`

① **Récursivité** — Un nombre N est pair si N-2 est pair

② **Cas particuliers** — La question n'a pas de sens pour un nombre négatif : le prédicat est faux dans ce cas.

③ **Cas d'arrêts** — 0 est pair.

④ **Vérifier l'arrête de la récursivité —** Au bout d'un moment, on se retrouvera avec un nombre égal à 0 ou négatif.

⑤ **Programation**

```
pair(N) :- sup(N,0), moins(N,2,N2), pair(N2);
pair(0).
```


PROLOG : RECURSIVITE

Influence de l'ordre des clauses

Exemple

```
/* faits : liens genealogiques directs */
       pere(jean,paul).
       pere(pierre,jean).
       mere(sophie,paul).
/* regles : calcul de relations genealogiques */
       parent(P,E) :- pere(P,E).
       parent(M,E) :- mere(M,E).
       ancetre(A,Desc) :- ancetre(A,P), parent(P,Desc).
       ancetre(A,Desc) :- parent(A,Desc).
```

But `?- ancetre(jean,paul).`

Conclusion

Ordre des clauses	• Pour toute clause récursive, il doit exister au moins une clause non récursive de même tête (clause d'arrêt) • Toute clause récursive doit avoir une clause d'arrêt qui est placée avant elle

PROLOG : RECURSIVITE

Influence de l'ordre des clauses

Exemple 2

```
(1) voisin(jean,paul).
(2) voisin(X,Y) :- voisin(Y,X).
```

Buts

```
?- voisin(paul,jean).
?- voisin(paul,pierre).
```

Conclusion

Ordre des clauses	• Un fait particulier (sans variable) ne constitue pas une clause d'arrêt. • Si un prédicat ne peut être simplement défini qu'à l'aide de clauses récursives, on casse cette récursivité — artificielle — en introduisant un prédicat intermédiaire non récursif

PROLOG : RECURSIVITE

Influence de l'ordre des buts : récursivité à gauche

Exemple

```
/* faits : liens genealogiques directs */
       pere(jean,paul).
       pere(pierre,jean).
       mere(sophie,paul).
/* regles : calcul de relations genealogiques */
       parent(P,E) :- pere(P,E).
       parent(M,E) :- mere(M,E).
       ancetre(A,Desc) :- parent(A,Desc).
       ancetre(A,Desc) :- ancetre(A,P), parent(P,Desc).
```

But

```
?- ancetre(paul,pierre).
```

Conclusion

Ordre des buts	L'ordre de sous-buts récursifs influe sur la terminaison, mais il n'y a pas de règle infaillible pour déterminer leur position dans une clause. En général : - chercher toujours à placer le plus à gauche possible les sous-buts d'une clause qui imposent le plus de restriction sur les variables - sinon, la construction d'un arbre de résolution (avec des exemples négatifs) nous renseigne sur le meilleur ordre possible.

PROLOG : RECURSIVITE

Terminaison et type d'argument

En pratique, un prédicat récursif ne peut généralement terminer dans tous les cas de figure. Il faut donc déterminer ses modes d'utilisation sans bouclage.

Mode d'utilisation d'un argument

Mode (-)	Pour un prédicat donné, un argument est dit utilisable en mode (-) si cet argument peut-être utilisé sous forme de variable sans risque de bouclage.
Mode (+)	Pour un prédicat donné, un argument est dit utilisable en mode (+) si cet argument peut-être utilisé sous forme de constante sans risque de bouclage.

Modes d'utilisation d'un prédicat

Pour un prédicat P d'arité N, c'est l'ensemble des N-uplets $\{m_1, .., m_N\}$ tels que chaque m_i correspond à un mode d'utilisation de l'argument en position i.

Exemple `ancetre/2`

STRUCTURES DE DONNEES : LISTES PROLOG

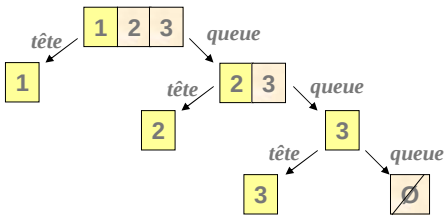
Listes : généralités

Utilité Représentation de données homogènes en nombre variable

Liste Représentation récursive où on distingue le premier élément de l'ensemble de ses autres éléments, qui est représenté lui-même par une liste

Liste = TETE + QUEUE avec
- TETE — 1er élément
- QUEUE — une liste (éventuellement vide) : autres éléments de l'ensemble

Exemple Ensemble {1,2,3}



STRUCTURES DE DONNEES : LISTES PROLOG

Listes Prolog

Deux syntaxes

Écriture récursive	**Usage**	nombre variable d'éléments
	Syntaxe	tête (terme Prolog) et queue (une liste) séparés par un trait vertical
	Exemples	`[]` *liste vide* `[1|[2|[]]]` *liste de 2 entiers* `[T|Q]` *forme générale*
Écriture énumérative	**Usage**	nombre d'éléments connus
	Syntaxe	éléments séparés par une trait virgule
	Exemples	`[]` *liste vide* `[1,2]` *liste de 2 entiers* `[[a,b],[c],[d,e]]` *liste de listes*

STRUCTURES DE DONNEES : LISTES PROLOG

Equivalence entre écritures

- Deux écritures pour une même structure de données en interne
- Ecritures combinables

Écriture récursive	Écriture énumérative
`[1|[2|[3]]]`	`[1,2,3]`
`[1|[2,3]]`	`[1,2,3]`
`[T|Q]`	écriture générale non énumérable
`[a,b|Q]`	écriture mixte non énumérable
`[[1,2]|[3]]`	`[[1,2],3]`

Exemple d'utilisation

Programme groupe([2,3,4]).
Question ?- groupe([T | Q]).
Réponse

```
> YES   T = 2   Q = [3,4]
```

STRUCTURES DE DONNEES : LISTES PROLOG

Prédicats de base de manipulations de listes

TETE /2

- tete(T,L) réussit si T est la tête de la liste L
- mode d'utilisation : prédicat non récursif

écriture brute	Tete(T,L):- L =[T|_].
écriture profitant de l'unification	Tete(T,[T|_]).

QUEUE /2

- queue(Q,L) réussit si Q est la queue de la liste L
- mode d'utilisation : prédicat non récursif

écriture profitant de l'unification	Tete(Q,[_|Q]).

LISTE /1

- liste(L) réussit si L est une liste.
- mode d'utilisation : prédicat récursif

écriture profitant de l'unification	Liste([]). Liste([T|Q]):-Liste(Q).

STRUCTURES DE DONNEES : LISTES PROLOG

Prédicats de base de manipulations de listes

ELEM /2

- elem(E,L) réussit si E est élément de la liste L
- mode d'utilisation : (+,+),(-,+)

analyse du problème
écriture Prolog

CONCAT /3

- concat(L1,L2,LC) réussit si LC est la liste obtenue par concaténation des éléments des listes L1etL2.
- mode d'utilisation : (+,+,+),(+,+,-)

analyse du problème
écriture Prolog

Logique pour l'informatique

Chapitre VI – Approches formelles de la logique

DEDUCTION FORMELLE - Objectifs

6.1. Notions

6.1.1. Rappels notions logique formelle
6.1.2. Système formel

6.2. Pratiques

6.2.1. Trouver la forme des théorèmes d'un système formel donné.
6.2.2. Trouver la démonstration d'un théorème pour un système formel donné
6.2.3. Trouver le système formel répondant à un problème donné

6.3. Approfondissement

6.3.1. Trouver le système formel répondant à un problème donné
6.3.2. Trouver la démonstration de théorèmes de la LP dans le SF de Lukasiewicz

LOGIQUE : APPROCHE FORMELLE (rappels)

- **Axiome**

  Proposition primitive considérée comme non démontrable et admise a priori

  Exemple : *axiomes de la géométrie euclidienne*

- **Théorème**

  Proposition pouvant être démontrée à partir d'axiomes ou d'autres théorèmes à l'aide de raisonnement formels valides. Les axiomes sont considérés comme des théorèmes particuliers.

  Notation $\vdash T$

- **Règle d'inférence**

  Schéma minimal de raisonnement valide qui permet de produire de nouvelles propositions à partir de prémisses qui sont soient des théorèmes, soit des hypothèses.

  Exemple : *modus ponens*

  (P1) Si A alors B
  (P2) A
  (C) Donc B

  Notation $P1, P2 \vdash T$

SYSTEME FORMEL : DEFINITION

Un système formel SF est un quadruplet SF = < V, L, A, R > avec :

- **Langage L** : ensemble des fbf construites sur le vocabulaire V.
- **Axiomes A** : sous-ensemble de L des fbf admises a priori.
- **Règles d'inférences R** qui permettent d'enrichir le système en de nouveaux théorèmes. Elles s'écrivent :

  $\alpha 1\ \alpha 2 \ldots \alpha n \quad |\!- \quad \beta 1\ \beta 2 \ldots \beta p$

  (prémisses) (conséquences)

  Si les fbf en prémisse sont des théorèmes, alors les fbf conséquences sont démontrées (ce sont des théorèmes).

SYSTEME FORMEL : EXEMPLES

Exemple 1 (D. Ofstadter, (1986) *Gödel, Escher, Bach*)

V = { a, b, c }

L = (a* b a* c a*)

A : $\vdash$ bc

R : avec X,Y,Z $\subset$ (a*)

(R1) XbYcZ $\vdash$ aXbYcZa
(R2) XbYcZ $\vdash$ XbYacZa

Exemple 2

V = { a, b, c }

L = (a* b a* c a*)

A : $\vdash$ ab
$\vdash$ bc

R : avec P,C suite quelconque d'éléments de V

(R1) P bb Q $\vdash$ b Q P b
(R2) P Q $\vdash$ P b Q
(R3) a P , P c $\vdash$ c P a

SYSTEME FORMEL DE LA LP

- **Lukasiewicz (1930)** Système complet de connecteurs { $\Rightarrow$, $\neg$ }
- **Axiomes** avec P, Q, R fbf quelconques de la LP

(A1) $P \Rightarrow (Q \Rightarrow P)$
(A2) $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))$ auto-distributivité de $\Rightarrow$
(A3) $(\neg P \Rightarrow \neg Q) \Rightarrow (Q \Rightarrow P)$ contraposition partielle

- **Règle d'inférence** avec P, Q, R fbf quelconques de la LP

(MP) P , $P \Rightarrow Q$ $\vdash$ Q modus ponens

- **Autres systèmes formels**

1910 *Russel et Whitebread* {$\Rightarrow$, $\vee$} 5 axiomes (4 en réalité)
1934 *Hilbert et Bernays* {$\Rightarrow$, $\neg$,$\wedge$,$\vee$,$\Leftrightarrow$} 15 axiomes
1953 *Meredith* {$\Rightarrow$, $\neg$} ou {$\Rightarrow$,$\vee$} 1 axiome

Logique des propositions (LP) : système sain et complet

SYSTEME FORMEL DE LA LP1

- **Lukasiewicz (1930)** Système complet de connecteurs { $\Rightarrow$, $\neg$ }
- **Axiomes** avec P, Q, R fbf quelconques de la LP

(A1) $P \Rightarrow (Q \Rightarrow P)$	
(A2) $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))$	auto-distributivité de $\Rightarrow$
(A3) $(\neg P \Rightarrow \neg Q) \Rightarrow (Q \Rightarrow P)$	contraposition partielle
(A4) $(\forall x\ A(x)) \Rightarrow A(x)$	**particularisation**
(A5) $(\forall x\ A(x)) \Rightarrow \neg(\exists x\ \neg A(x))$	**définition de $\exists$**

- **Règle d'inférence** avec P, Q, R fbf quelconques de la LP

(MP) $P , P \Rightarrow Q$	$\vdash$	Q	modus ponens
(LP1) $A \vdash \forall x\ A(x)$ avec x variable libre de A			**généralisation**

Logique des prédicats (LP1) : système sain et complet