# 7

# La syntaxe complète de Prolog ISO

CETTE SECTION décrit de manière précise la syntaxe des termes, et donc des textes [1] et des données Prolog.

Les termes sont les structures de données manipulées durant l'exécution par les applications Prolog. La sous-section 2 définit comment les termes forment les textes et les données Prolog. La sous-section 3 définit comment les unités lexicales doivent être combinées pour former les termes. La sous-section 4 définit comment les suites de caractères constituent les unités lexicales.

## 7.1 Notations

La syntaxe de Prolog est décrite à l'aide d'une grammaire BNF obéissant aux conventions suivantes.

Les méta-opérateurs '=', '|' et ',' sont associatifs à droite. Le méta-opérateur ',' est plus «fort» que '|', qui est lui-même plus fort que '='. Ainsi, une règle écrite comme ceci :

$$\alpha \ = \ \beta \ , \ \gamma \ | \ \delta$$

se lit comme ceci :

$$( \ \alpha \ = \ ( \ ( \ \beta \ , \ \gamma \ ) \ | \ \delta \ ) \ )$$

Les symboles de la grammaire sont définis de la manière suivante :

- `=` Métasymbole fondamental de réécriture. Il sépare le membre gauche du membre droit de chaque règle.
- `;` Un point-virgule termine chaque règle de grammaire.
- `,` La virgule est le méta-opérateur de concaténation : $\alpha$ , $\beta$ représente la concaténation de chaînes respectivement définies par les suites de symboles $\alpha$ et $\beta$.

1. La notion de programme n'est pas la même en prolog et dans la plupart des autres langages. A la place du mot programme nous emploierons donc l'expression «texte prologIV».

`|` La barre verticale est le méta-opérateur d'alternative : $\alpha$ `|` $\beta$ représente une alternative dont les termes sont deux chaînes respectivement définies par les suites de symboles $\alpha$ et $\beta$.

`(...)` Les parenthèses servent à regrouper les symboles, afin de donner à une règle une structure différente de celle que détermineraient, sans parenthèses, les méta-opérateurs qui y figurent.

`[...]` Les éléments encadrés par des crochets sont optionnels.

`{...}` Les éléments encadrés par des accolades peuvent apparaître un nombre quelconque de fois.

`"..."` Une suite de caractères encadrée par des guillemets constitue un symbole terminal de la grammaire.

`'...'` Une suite de caractères encadrée par des apostrophes constitue également un symbole terminal de la grammaire.

`(*...*)` Un texte compris entre '`(*`' et '`*)`' est tenu pour un commentaire et ne fait pas partie de la grammaire.

*caractères*

Une suite de caractères qui n'est pas encadrée par des guillemets ou des apostrophes et ne contient aucun des méta-symboles précédents constitue un symbole non terminal.

Sous chaque règle de grammaire peuvent apparaître diverses autres lignes qui spécifient des attributs de la règle. Chacune de ces lignes est introduite par une étiquette caractéristique :

*Abstr.* Cette ligne montre les termes abstraits associés aux symboles de la grammaire.

*Prior.* Cette ligne exprime la priorité des termes.

A chaque terme et chaque opérateur est associé une priorité, qui est un entier $r$ vérifiant $0 \leq r \leq 1201$.

Un terme atomique et un terme composé en notation fonctionnelle ont une priorité égale à zéro.

Un terme composé exprimé en notation opérationnelle (i.e. son foncteur principal apparaît comme un opérateur) a une priorité supérieure ou égale à celle de son foncteur principal.

*Cond.* Cette ligne exprime des conditions qui doivent être satisfaites pour que la règle de grammaire s'applique.

*Spécif.* Cette ligne donne les spécificateurs des opérateurs (voyez la table 7.1) intervenant dans la règle.

## 7.2 Textes et données Prolog

Un texte Prolog est une suite de termes lisibles[2] qui expriment des directives ou des clauses.

2. *Terme lisible* est une tentative de traduction de *read term*

### 7.2.1 Texte Prolog

| | | | |
|---|---|---|---|
| | `texte prolog =` | `directive,` | `texte prolog;` |
| *Abstr.* | $d \cdot t$ | $d$ | $t$ |
| | `texte prolog =` | `clause,` | `texte prolog;` |
| *Abstr.* | $d \cdot t$ | $d$ | $t$ |
| | `texte prolog = ;` | | |
| *Abstr.* | *nil* | | |

*Directives*

| | | |
|---|---|---|
| | `directive =` | `terme directive , fin;` |
| *Abstr.* | $d$ | $d$ |
| | `terme directive =` | `terme;` |
| *Abstr.* | $:-(d)$ | $:-(d)$ |
| *Prior.* | | 1201 |

Une directive figurant dans un texte Prolog spécifie :

- des propriétés des procédures définies dans le texte Prolog,
- le format et la syntaxe abstraite de termes lisibles du texte Prolog,
- un but qui devra être exécuté lorsque le texte Prolog aura été préparé pour l'exécution,
- un autre texte Prolog devant être préparé pour l'exécution.

*Clauses*

| | | |
|---|---|---|
| | `clause =` | `terme clause , fin;` |
| *Abstr.* | $c$ | $c$ |
| | `terme clause =` | `terme;` |
| *Abstr.* | $c$ | $c$ |
| *Prior.* | | 1201 |
| *Cond.* | Le foncteur principal de $c$ n'est pas `':-'/1`. | |

Une clause figurant dans un texte Prolog spécifie une clause d'une procédure définie par l'utilisateur qui doit être ajoutée à la base de données.

Les caractères de la clause supportent les mêmes contraintes que ceux d'un terme lisible dans une exécution réussie du prédicat prédéfini `read/1`.

La clause supporte les mêmes contraintes que $terme$ dans une exécution réussie du prédicat prédéfini `assertz(`$terme$`)`, sauf une : il n'y aura pas d'erreur si la clause correspond à une procédure statique[3].

Le P/N[4] de la clause ne doit pas être celui d'un prédicat prédéfini ni d'une

3. Alors que c'est une erreur que d'utiliser `assertz/1` ou `asserta/1` pour tenter d'ajouter des clauses à des procédures atatiques.

4. Par «le P/N» nous désignons le couple constitué du prédicat P et de l'arité N du terme composé qui est le membre gauche de la clause.

structure de contrôle.

Toutes les clauses d'une procédure définie par l'utilisateur doivent être des termes lisibles consécutifs d'un même texte Prolog, sauf si la directive `discontiguous` a été utilisée pour indiquer le contraire.

Toutes les clauses d'une procédure définie par l'utilisateur doivent être des termes lisibles se trouvant dans un même texte Prolog, sauf si la directive `multifile` a été utilisée pour indiquer le contraire.

Si aucune clause n'a été spécifiée pour une procédure mentionnée dans une directive `dynamic/1`, `multifile/1` ou `discontiguous/1`, alors la procédure existe mais elle n'a pas de clause.

### 7.2.2 Données Prolog

Un terme lisible Prolog peut être lu comme une donnée par un appel du prédicat prédéfini `read_term/3`.

| | `terme lisible =` | `terme , fin;` |
|---|---|---|
| *Abstr.* | $a$ | $a$ |
| *Prior.* | | 1201 |

## 7.3 Termes

Un terme Prolog est soit un terme atomique, soit une variable, soit un terme composé.

### 7.3.1 Termes atomiques

*Nombres*

| | `terme =` | `entier;` |
|---|---|---|
| *Abstr.* | $n$ | $n$ |
| *Prior.* | 0 | |

| | `terme =` | `nombre flottant;` |
|---|---|---|
| *Abstr.* | $r$ | $r$ |
| *Prior.* | 0 | |

*Nombres négatifs*

| | `terme =` | `nom ,` | `entier;` |
|---|---|---|---|
| *Abstr.* | $n$ | $-$ | $n$ |
| *Prior.* | 0 | | |

| | `terme =` | `nom ,` | `nombre flottant;` |
|---|---|---|---|
| *Abstr.* | $r$ | $-$ | $r$ |
| *Prior.* | 0 | | |

*Atomes*

| | `terme =` | `atome;` |
|---|---|---|
| *Abstr.* | $a$ | $a$ |
| *Prior.* | 0 | |

*Cond.* $a$ n'est pas un opérateur.

```
        terme =  atome;
Abstr.  a        a
Prior.  0
```

*Cond.* $a$ est un opérateur.

Un atome qui est un opérateur ne peut pas être l'opérande immédiat d'un autre opérateur (cela découle de la priorité maximale donnée à un tel terme).

```
        atome =  nom;
Abstr.  a        a

        atome =  crochet gauche ,   crochet droit;
Abstr.  []

        atome =  accolade gauche ,  accolade droit;
Abstr.  {}
```

Un atome est un nom, ou [] (la liste vide) ou {} (la paire d'accolades vide).

### 7.3.2 Variables

```
        terme =  variable;
Abstr.  v        v
Prior.  0
```

### 7.3.3 Termes composés – notation fonctionnelle

Un terme composé peut toujours être exprimé en notation fonctionnelle. Si le foncteur principal est un opérateur, le terme peut être exprimé aussi en notation opérationnelle. Si le foncteur principal est `'.'/2`, le terme peut aussi être exprimé en notation de liste, et parfois aussi comme une liste à guillemets. Si le foncteur principal est `{}/1`, le terme peut aussi être noté avec une paire d'accolades.

```
    terme =      atome ,   par gauche contr , liste args , par droite;
Abstr. f(l)      f                           l
Prior. 0

    liste args = argument;
Abstr. a         a

    liste args = argument , virgule ,        liste args;
Abstr. a, l      a                           l
```

*Arguments*

Un argument peut apparaître comme argument d'un terme composé ou comme élément d'une liste. Ce peut être un atome qui est un opérateur ou un terme avec une priorité inférieure à 999. Lorsqu'un argument est un terme quelconque, sa priorité est inférieure à celle de la virgule, si bien qu'il ne peut pas y avoir de conflit entre les occurrences de la virgule en tant qu'opérateur infixe et les occurrences de la virgule en tant que séparateur d'arguments.

```
        argument =  atome;
Abstr.  a           a
```

*Cond.* $a$ est un operateur autre qu'une virgule.

TAB. 7.1 – *Spécificateurs des opérateurs*

| Spécificateur | Classe | Associativité |
|---|---|---|
| `fx` | préfixe | non associatif |
| `fy` | préfixe | associatif à droite |
| `xfx` | infixe | non associatif |
| `xfy` | infixe | associatif à droite |
| `yfx` | infixe | associatif à gauche |
| `xf` | postfixe | non associatif |
| `yf` | postfixe | associatif à gauche |

| | | |
|---|---|---|
| | `argument =` | terme ; |
| *Abstr.* | $a$ | $a$ |
| *Prior.* | | 999 |

### 7.3.4 Termes composés – notation opérationnelle

Les termes composés dont le symbole du foncteur est un opérateur défini dans la table 7.4 peuvent être écrits dans une notation opérationnelle.

Un opérateur est défini par son nom, son spécificateur et sa priorité.

La priorité d'un opérateur est un nombre entier $r$ vérifiant $1 \leq r \leq 1200$.

Le spécificateur d'un opérateur est une chaîne de caractères mnémonique qui définit la classe (préfixe, infixe ou postfixe) et l'associativité (associativité à gauche, associativité à droite ou sans associativité) de l'opérateur. La table 7.1 récapitule les différents spécificateurs.

Un opérande avec une priorité égale ou inférieure à celle d'un opérateur associatif à droite, écrit à droite de cet opérateur, n'a pas besoin d'être écrit entre parenthèses. Un opérande avec une priorité inférieure à celle d'un opérateur associatif à gauche, écrit à gauche de cet opérateur, n'a pas besoin d'être écrit entre parenthèses. Un opérande avec une priorité égale à celle d'un opérateur associatif à gauche, écrit à gauche de cet opérateur, doit être écrit entre parenthèses uniquement si le foncteur principal de l'opérande est un opérateur associatif à droite. Un opérande avec la même priorité qu'un opérateur non associatif, écrit à droite ou à gauche de cet opérateur, doit être écrit entre parenthèses. Dans les tables 7.2 et 7.3 on suppose que chaque atome `fx`, `fy`, `xfx`, `xfy`, `yfx`, `xf` et `yf` est un opérateur ayant le spécificateur correspondant. La table 7.2 montre quelques termes invalides et la manière dont ils doivent être parenthésés pour devenir valides.

La table 7.3 montre des termes (valides) non parenthésés en correspondance avec des termes parenthésés équivalents. On suppose ici que les opérateurs `xfy` et `yfx` (resp. `fy` et `yf`) ont même priorité.

*Opérande*

Un opérande est un terme.

Ci-après, le non-terminal `gterme` représente le sous-ensemble de l'ensemble

TAB. 7.2 – *Termes invalides et termes valides correspondants*

| Terme invalide | terme valide |
|---|---|
| `fx fx 1` | `fx (fx 1)` |
| `1 xf xf` | `(1 xf) xf` |
| `1 xfx 2 xfx 3` | `(1 xfx 2) xfx 3` |
| `1 xfx 2 xfx 3` | `1 xfx (2 xfx 3)` |

TAB. 7.3 – *Termes équivalents*

| Termes non parenthésés | Termes équivalents parenthésés |
|---|---|
| `fy fy 1` | `fy (fy 1)` |
| `1 xfy 2 xfy 3` | `1 xfy (2 xfy 3)` |
| `1 xfy 2 yfx 3` | `1 xfy (2 yfx 3)` |
| `fy 2 yf` | `fy (2 yf)` |
| `1 yf yf` | `(1 yf) yf` |
| `1 yfx 2 yfx 3` | `(1 yfx 2) yfx 3` |

des termes formé des termes qui peuvent être l'opérande gauche d'un opérateur associatif à gauche ayant une priorité donnée.

| | `terme =` | `gterme;` |
|---|---|---|
| *Abstr.* | $a$ | $a$ |
| *Prior.* | $n$ | $n$ |

| | `gterme =` | `terme;` |
|---|---|---|
| *Abstr.* | $a$ | $a$ |
| *Prior.* | $n$ | $n-1$ |

A tout endroit où un terme de priorité n est permis peut figurer un terme de priorité inférieure à $n$.

| | `terme =` `par gauche ,` | `terme ,` `par droite;` |
|---|---|---|
| *Abstr.* | $a$ | $a$ |
| *Prior.* | 0 | 1201 |

Les parenthèses servent à contourner la priorité des opérateurs.

*Opérateurs comme fonctions*

| | `gterme =` | `terme ,` | `oper ,` | `terme;` |
|---|---|---|---|---|
| *Abstr.* | $f(a,b)$ | $a$ | $f$ | $b$ |
| *Prior.* | $n$ | $n-1$ | $n$ | $n-1$ |
| *Spécif.* | | | `xfx` | |

| | `gterme =` | `gterme ,` | `oper ,` | `terme;` |
|---|---|---|---|---|
| *Abstr.* | $f(a,b)$ | $a$ | $f$ | $b$ |
| *Prior.* | $n$ | $n$ | $n$ | $n-1$ |
| *Spécif.* | | | `yfx` | |

| | `terme =` | `terme ,` | `oper ,` | `terme;` |
|---|---|---|---|---|
| *Abstr.* | $f(a,b)$ | $a$ | $f$ | $b$ |
| *Prior.* | $n$ | $n-1$ | $n$ | $n$ |
| *Spécif.* | | | `xfy` | |

| | `gterme =` | `gterme ,` | `oper ,` |
|---|---|---|---|
| *Abstr.* | $f(a)$ | $a$ | $f$ |
| *Prior.* | $n$ | $n$ | $n$ |
| *Spécif.* | | | `yf` |

| | `gterme =` | `terme ,` | `oper ,` |
|---|---|---|---|
| *Abstr.* | $f(a)$ | $a$ | $f$ |
| *Prior.* | $n$ | $n-1$ | $n$ |
| *Spécif.* | | | `xf` |

| | `terme =` | `oper ,` | `terme ,` |
|---|---|---|---|
| *Abstr.* | $f(a)$ | $f$ | $a$ |
| *Prior.* | $n$ | $n$ | $n$ |
| *Spécif.* | | `fy` | |

| | `gterme =` | `oper ,` | `terme ,` |
|---|---|---|---|
| *Abstr.* | $f(a)$ | $f$ | $a$ |
| *Prior.* | $n$ | $n$ | $n-1$ |
| *Spécif.* | | `fx` | |

*Opérateurs*

Un opérateur est un nom ou une virgule.

| | `oper =` | `nom;` |
|---|---|---|
| *Abstr.* | $a$ | $a$ |
| *Prior.* | $n$ | $n$ |
| *Spécif.* | $s$ | $s$ |

*Cond.* a est un opérateur (cf. table 7.4)

| | `oper =` | `virgule;` |
|---|---|---|
| *Abstr.* | , | |
| *Prior.* | 1000 | |
| *Spécif.* | `xfy` | |

Il ne peut pas exister deux opérateurs avec la même classe (préfixe, infixe ou postfixe) et le même nom.

Il ne peut pas y avoir un opérateur infixe et un opérateur postfixe avec le même nom.

TAB. 7.4 – *La table des opérateurs*

| Priorité | Spécificateur | Opérateur(s) |
|---|---|---|
| 1200 | xfx | :- --> |
| 1200 | fx | :- ?- |
| 1100 | xfy | ; |
| 1050 | xfy | -> |
| 1000 | xfy | , |
| 900 | fy | \+ |
| 700 | xfx | = \= |
| 700 | xfx | == \== @< @=< @> @>= |
| 700 | xfx | =.. |
| 700 | xfx | is =:= =\= < =< > >= |
| 500 | yfx | + - /\ \/ |
| 400 | yfx | * / // rem mod << >> |
| 200 | xfx | ** |
| 200 | xfy | ^ |
| 200 | fy | - \ |
| 100 | xfx | @ |
| 50 | xfx | : |

*La table des opérateurs*

La table des opérateurs détermine quels atomes doivent être considérés comme des opérateurs lorsque

- une suite d'unités lexicales est analysée comme un terme lisible par le prédicat `read_term/3`, ou bien
- un texte Prolog est préparé pour l'exécution, ou bien
- l'expression écrite d'un terme Prolog est produite par un des prédicats `write_term/...`, `write/...`, etc. La table 7.4 montre les opérateurs prédéfinis, c'est-à-dire ceux qui constituent l'état initial de la table des opérateurs.

Cette table peut être modifiée à l'exécution, grâce au prédicat prédéfini `op/3`. Elle contient les opérateurs associés aux :

1. Prédicats définis comme des opérateurs :
   - unification de termes `'='/2`, `'\='/2`
   - comparaison de termes `'=='/2`, `'\='/2`, `'@<'/2`, `'@=<'/2`, `'@>'/2`, `'@>='/2`,
   - décomposition de termes (univ) `'=..'/2`,
   - évaluation arithmétique `is/2`,
   - comparaison arithmétique `'=:='/2`, `'=\='/2`, `'<'/2`, `'=<'/2`, `'>'/2`, `'>='/2`,
   - négation par échec `'\+'/1`
2. Structures de contrôle définies comme des opérateurs :
   - conjonction `','/2`,
   - disjonction `';'/2`,

– si-alors `'->'/2`

3. Fonctions évaluables définies comme des opérateurs :
   – les opérateurs arithmétiques binaires `'+'/2`, `'-'/2`, `'*'/2`, `'//'/2`, `'/'/2`, `rem/2`, `mod/2`, `'**'/2`,
   – les opérateurs arithmétiques unaires`'-'/1`,
   – les fonctions logiques `'>>'/2`, `'<<'/2`, `'/\'/2`, `'\/'/2`, `'\'/1`

### 7.3.5 Termes composés - notation de liste

Une liste est soit la liste vide, soit un terme composé dont le foncteur principal est `'.'/2`.

| | | | | |
|---|---|---|---|---|
| | `terme =` | `crochet gauche ,` | `éléments ,` | `crochet droit;` |
| *Abstr.* | $l$ | | $l$ | |
| *Prior.* | 0 | | | |
| | `éléments =` | `argument,` | `virgule,` | `éléments;` |
| *Abstr.* | $.(h, l)$ | $h$ | | $l$ |
| | `éléments =` | `argument,` | `sépar tête queue,` | `éléments;` |
| *Abstr.* | $.(h, t)$ | $h$ | | $t$ |
| | `éléments =` | `argument;` | | |
| *Abstr.* | $.(t, [])$ | $t$ | | |

*Exemples*

Voici des termes exprimés en notation de liste et en notation fonctionnelle

```
[a] == .(a, []).

[a, b] == .(a, .(b, [])).

[a | b] == .(a, b).
```

### 7.3.6 Termes composés - notation avec une paire d'accolades

| | | | | |
|---|---|---|---|---|
| | `terme =` | `accolade gauche ,` | `terme ,` | `accolade droite;` |
| *Abstr.* | $(l)$ | | $l$ | |
| *Prior.* | 0 | 1201 | | |

*Exemples*

Voici des termes exprimés en notation avec une paire d'accolades et en notation fonctionnelle

```
{a} == '{}'(a).
{a, b} == '{}'(','(a, b))
```

### 7.3.7 Termes composés - notation en liste à guillemets

Une liste à guillemets est soit un atome, soit une liste non vide.

Si l'indicateur `double_quotes` a la valeur `chars`, une unité lexicale de type liste à guillemets $ldq$ contenant $L$ caractères est une liste $l$ de $L$ éléments dont le nième est le caractère représenté par le nième caractère de $ldq$.

Si l'indicateur `double_quotes` a la valeur `codes`, une unité lexicale de type liste à guillemets $ldq$ contenant $L$ caractères est une liste $l$ de $L$ éléments dont le nième est le nombre entier associé au nième caractère de $ldq$.

Si l'indicateur `double_quotes` a la valeur `atoms`, une unité lexicale de type liste à guillemets $ldq$ contenant $L$ caractères est un atome $l$ formé par la concaténation d'une suite de $L$ caractères dont le nième est le caractère représenté par le nième caractère de $ldq$.

```
        terme =  liste à guillemets;
 Abstr. l        ccl
 Prior. 0
```

*Exemples*

```
( current_prolog_flag(double_quotes, chars),
  atom_chars('bonjour', "bonjour") ;
  current_prolog_flag(double_quotes, codes),
  atom_codes('bonjour', "bonjour") ;
  current_prolog_flag(double_quotes, atom),
  'bonjour' == "bonjour" ).
Succès.
( current_prolog_flag(double_quotes, chars),
  [] == "" ;
  current_prolog_flag(double_quotes, codes),
  [] == "" ;
  current_prolog_flag(double_quotes, atom),
   » == "" ).
Succès.
```

## 7.4 Unités lexicales

```
unité lexicale =
     nom |
     variable |
     entier |
     nombre flottant |
     liste à guillemets |
     par gauche |
     par gauche contr |
     par droite |
     crochet gauche |
     crochet droit |
     accolade gauche |
     accolade droite |
     sépar tête queue |
     virgule;

nom =
     [ suite de textes décoratifs ] ,
     unité nom;

variable =
     [ suite de textes décoratifs ] ,
     unité variable;

entier =
```

```
     [ suite de textes décoratifs ] ,
     unité entier;

nombre flottant =
     [ suite de textes décoratifs ] ,
     unité nombre flottant;

liste à guillemets =
     [ suite de textes décoratifs ] ,
     unité liste à guillemets;

par gauche =
     [ suite de textes décoratifs ] ,
     unité par gauche;

par gauche contr =
     unité par gauche;

par droite =
     [ suite de textes décoratifs ] ,
     unité par droite;

crochet gauche =
     [ suite de textes décoratifs ] ,
     unité crochet gauche;

crochet droit =
     [ suite de textes décoratifs ] ,
     unité crochet droit;

accolade gauche =
     [ suite de textes décoratifs ] ,
     unité accolade gauche;

accolade droite =
     [ suite de textes décoratifs ] ,
     unité accolade droite;

sépar tête queue =
     [ suite de textes décoratifs ] ,
     unité sépar tête queue;

virgule =
     [ suite de textes décoratifs ] ,
     unité virgule;

fin =
     [ suite de textes décoratifs ] ,
     unité fin;
```

### 7.4.1 Textes décoratifs

Les textes décoratifs séparent les unités lexicales et résolvent deux ambiguïtés :

- un point `.` est-il un caractère graphique ou un terminateur ?
- lorsqu'il est suivi d'une parenthèse ouverte, un atome est-il le foncteur d'un terme composé ou bien un opérateur préfixe ?

```
suite de textes décoratifs =
     texte décoratif ,
      texte décoratif ;

texte décoratif =
     caractére décoratif
     | commentaire;
```

Un commentaire de fin de ligne ne doit pas contenir le caractère « nouvelle ligne». Un commentaire avec délimiteurs ne doit pas contenir le délimiteur de fin de commentaire.

```
commentaire =
     commentaire de fin de ligne
     | commentaire avec délimiteurs;

commentaire de fin de ligne =
     début de commentaire de fin de ligne ,
     texte du commentaire ,
     caractère de fin de ligne;

commentaire avec délimiteurs =
     début de commentaire ,
     texte du commentaire ,
     fin de commentaire;

texte du commentaire = { caractère };

début de commentaire =
     limite commentaire 1 , limite commentaire 2;

fin de commentaire =
     limite commentaire 2 , limite commentaire 1;

limite commentaire 1 = "/";

limite commentaire 2 = "*";
```

### 7.4.2 Noms

```
unité nom =
     unité à lettres et chiffres |
     unité graphique |
     unité à apostrophes |
     unité point-virgule |
     unité coupure;

unité à lettres et chiffres =
     caractère lettre minuscule ,
     { caractère alphanumérique };
```

Une unité graphique ne doit pas commencer par la suite de caractères `début de commentaire`.

Une unité graphique ne doit pas être le caractère `.` (point) suivi d'un caractère décoratif.

```
unité graphique =
     caractère d'unité graphique ,
     { caractère d'unité graphique };

caractère d'unité graphique =
     caractère graphique |
     caractère backslash;
```

Une unité à apostrophes est faite d'une suite de caractères encadrée par des apostrophes. Si la suite, sans les apostrophes, constitue un atome valide, alors l'unité désigne ce même atome.

Une unité à apostrophes qui ne contient aucun caractère (c'est-à-dire qui est réduite aux apostrophes) est l'atome nul.

Une unité à apostrophes peut être découpée sur plusieurs lignes à l'aide d'une séquence de continuation. Une unité à apostrophes contenant des séquences de continuation est équivalente à l'unité obtenue en supprimant les séquences de continuation.

```
unité à apostrophes =
     caractère apostrophe ,
     { élément d'unité à apostrophes }
     caractère apostostrophes |

séquence de continuation =
     caractère d'unité à apostrophes |
     séquence de continuation;

séquence de continuation =
     caractère backslash ,
     caractère de fin de ligne;

unité point-virgule = caractère point-virgule;

unité coupure = caractère coupure;
```

Ainsi, `'abc'` et `abc` désignent la même unité lexicale, mais `'\\/'` et `\\/` non, car dans une unité à apostrophes, `\` indique le début d'une séquence d'échappement.

*Caractères d'une unité à apostrophes*

```
caractère d'unité à apostrophes =
     caractère hors quote |
     caractère apostrophe , caractère apostrophe |
     caractère guillemet |
     caractère back quote;

caractère d'unité à guillemets =
     caractère hors quote |
     caractère apostrophe |
     caractère guillemet , caractère guillemet |
```

```
     caractère back quote;

caractère d'unité à back quote =
     caractère hors quote |
     caractère apostrophe |
     caractère guillemet |
     caractère back quote , caractère back quote;

caractère hors quote =
     caractère graphique |
     caractère alphanumérique |
     caractère solitaire |
     caractère espace |
     méta-séquence d'échappement |
     séquence d'échappement de contrôle |
     séquence d'échappement octale |
     séquence d'échappement hexadécimale;
```

Il y a trois sortes d'unités lexicales « quotées » : les unités encadrées par des apostrophes, celles encadrées par des guillemets et celles encadrées par des « back quotes ». Dans les trois cas, lorsque le caractère d'encadrement doit figurer dans l'unité, il doit être doublé.

Dans une unité encadrée, un caractère graphique, alphanumérique, solitaire ou un espace, représente ce caractère lui-même.

Une méta-séquence d'échappement représente le caractère introduit par la méta-séquence.

```
méta-séquence d'échappement =
     caractère backslash ,
     méta-caractère;
```

Une séquence d'échappement de contrôle représente le caractère de contrôle indiqué par le nom du caractère de contrôle symbolique si et seulement si ce caractère appartient à l'ensemble de caractères du processeur.

```
séquence d'échappement de contrôle =
     caractère backslash ,
     caractère de contrôle symbolique;

caractère de contrôle symbolique =
     caractère symbolique alert |
     caractère symbolique backspace |
     caractère symbolique carriage return |
     caractère symbolique form feed |
     caractère symbolique horizontal tab |
     caractère symbolique new line |
     caractère symbolique vertical tab;

caractère symbolique alert = "a";

caractère symbolique backspace = "b";

caractère symbolique carriage return = "r";
```

```
caractère symbolique form feed = "f";

caractère symbolique horizontal tab = "t";

caractère symbolique new line = "n";

caractère symbolique vertical tab = "v";
```

Une séquence d'échappement octale ou hexadécimale représente le caractère, appartenant à l'ensemble de caractères du processeur, dont le code interne correspond au nombre donné par la séquence octale ou hexadécimale.

```
séquence d'échappement octale =
     caractère backslash ,
     chiffre octal ,
     { chiffre octal } ,
     caractère backslash;

séquence d'échappement hexadécimale =
     caractère backslash ,
     caractère symbolique hexadécimal ,
     chiffre octal ,
     { chiffre octal } ,
     caractère backslash;

caractère symbolique hexadécimal = "x";
```

### 7.4.3 Variables

```
unité variable =
     variable anonyme |
     variable nommée;

variable anonyme =
     caractère indicateur de variable;

variable nommée =
     caractère indicateur de variable ,
     caractère alphanumérique ,
     { caractère alphanumérique } |
     caractère lettre majuscule ,
     { caractère alphanumérique };

caractère indicateur de variable =
     caractère blanc souligné;
```

### 7.4.4 Nombres entiers

```
entier =
     constante entière |
     constante code caractère |
     constante binaire |
     constante octale |
     constante hexadécimale;

constante entière =
```

```
     chiffre décimal ,
     { chiffre décimal };

constante code caractère = "0" ,
     caractère apostrophe ,
     caractère d'unité à apostrophes;
```

Une constante code caractère représente la valeur du code interne du caractère.

```
constante binaire =
     indicateur de constante binaire ,
     chiffre binaire ,
     { chiffre binaire };

indicateur de constante binaire = "0b";

constante octale =
     indicateur de constante octale ,
     chiffre octal ,
     { chiffre octal };

indicateur de constante octale = "0o";

constante hexadécimale =
     indicateur de constante hexadécimale ,
     chiffre hexadécimal ,
      chiffre hexadécimal ;

indicateur de constante hexadécimale = "0x";
```

Les constantes entières sont sans signe. Les entiers négatifs sont définis par la syntaxe des termes.

### 7.4.5 Nombres à virgule flottante

```
nombre flottant =
     constante entière ,
     fraction ,
     [ exposant ];

fraction =
     caractère point décimal ,
     chiffre décimal ,
      chiffre décimal ;

exposant =
     caractère exposant ,
     signe ,
     constante entière;

signe =
     caractère signe moins |
     [ caractère signe plus ];

caractère signe plus = "+";

caractère signe moins = "-";
```

```
caractère point décimal = ".";

caractère exposant = "e" | "E";
```

Une constante flottante est sans signe. Les flottants négatifs sont définis par la syntaxe des termes.

### 7.4.6 Listes à guillemets

Une unité lexicale de type liste encadrée par des guillemets représente un terme qui dépend de la valeur de l'indicateur `double_quotes` au moment où le terme ou le texte Prolog contenant l'unité en question est lu.

Une liste à guillemets peut être découpée sur plusieurs lignes à l'aide d'une séquence de continuation. Une liste à guillemets contenant des séquences de continuation est équivalente à l'unité obtenue en supprimant les séquences de continuation.

```
liste à guillemets =
     caractère guillemet ,
     élément d'unité à guillemets  ,
     caractère guillemet;

élément d'unité à guillemets =
     caractère d'unité à guillemets |
     séquence de continuation;
```

### 7.4.7 Chaînes à back quotes

Une chaîne à back quotes est une suite de caractères encadrés par deux caractères ` (« back quote »). Une chaîne à back quotes peut être découpée sur plusieurs lignes à l'aide d'une séquence de continuation. Une chaîne à back quotes contenant des séquences de continuation est équivalente à la chaîne obtenue en supprimant les séquences de continuation.

```
chaîne à back quotes =
     caractère back quote ,
      élément de chaîne à back quote  ,
     caractère back quote;

élément de chaîne à back quote =
     caractère de chaîne à back quote |
     séquence de continuation;
```

La norme ISO définit la syntaxe des chaînes à back quote mais pour le moment elle n'introduit pas d'unité lexicale ni de terme qui obéirait effectivement à cette syntaxe.

### 7.4.8 Autres unités lexicales

```
unité par gauche = caractère par gauche;

unité par droite = caractère par droite;
```

```
unité crochet gauche = caractère crochet gauche;

unité crochet droit = caractère crochet droit;

unité accolade gauche = caractère accolade gauche;

unité accolade droite = caractère accolade droite;

unité sépar tête queue = caractère sépar tête queue;

unité virgule = caractère virgule;

unité fin = caractère fin;

caractère fin = ".";
```

Un caractère de fin ne peut pas être suivi par un caractère autre qu'un caractère décoratif ou un `%`.

## 7.5 Caractères du processeur

L'ensemble des caractères du processeur dépend de chaque implémentation. Il doit contenir tous les caractères définis par la règle caractère, mais peut inclure des éléments additionnels, appelés caractères étendus. Selon l'implémentation, ces caractères sont graphiques, alphanumériques, solitaires, etc.

```
caractère =
     caractère graphique |
     caractère alphanumérique |
     caractère solitaire |
     caractère décoratif |
     méta-caractère;
```

### 7.5.1 Caractères graphiques

```
caractère graphique =
     "#" | "$" | "&" | "*" | "+" | "-" | "." |
     "/" | ":" | "<" | "=" | ">" | "?" | "@" |
     "^" | "~";
```

### 7.5.2 Caractères alphanumériques

```
caractère alphanumérique =
     caractère alphabétique |
     caractère chiffre décimal;

caractère alphabétique =
     caractère blanc souligné |
     caractère lettre;

caractère lettre =
     caractère lettre majuscule |
     caractère lettre minuscule;

caractère lettre minuscule =
```

```
     "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
     "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
     "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
     "y" | "z";

caractère lettre majuscule =
     "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
     "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
     "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" |
     "Y" | "Z";

caractère chiffre décimal =
     "0" | "1" | "2" | "3" | "4" |
     "5" | "6" | "7" | "8" | "9" |

caractère chiffre binaire = "0" | "1";

caractère chiffre octal =
     "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7";

caractère chiffre hexadécimal =
     "0" | "1" | "2" | "3" | "4" |
     "5" | "6" | "7" | "8" | "9" |
     ("a" | "A") | ("b" | "B") | ("c" | "C") |
     ("d" | "D") | ("e" | "E") | ("f" | "f");

caractère blanc souligné = "_";
```

### 7.5.3 Caractères solitaires

```
caractère solitaire =
     caractère de coupure |
     caractère par gauche |
     caractère par droite |
     caractère virgule |
     caractère point-virgule |
     caractère crochet gauche |
     caractère crochet droite |
     caractère accolade gauche |
     caractère accolade droite |
     caractère sépar tête queue |
     caractère commentaire de fin de ligne;
```

Dans une unité quotée (encadrée par des apostrophes, des guillemets, etc.) un caractère solitaire se représente lui même.

Un caractère solitaire qui n'est pas à l'intérieur d'une unité lexicale quotée constitue une unité à lui tout seul (excepté `%` et les caractères restants sur la ligne, qui n'ont pas de signification dans un texte Prolog ou un terme lisible).

Un caractère solitaire n'a pas besoin d'être séparé de ses unités lexicales voisines par des caractères décoratifs.

### 7.5.4 Caractères décoratifs

```
caractère décoratif =
```

```
        caractère espace |
        caractère tabulation horizontale |
        caractère fin de ligne;

caractère espace = " ";

caractère tabulation horizontale =
        dépend de l'implantation;

caractère fin de ligne =
        dépend de l'implantation;
```

Les caractères décoratifs sont parfois requis pour séparer deux unités lexicales. Un texte décoratif ne constitue jamais une unité lexicale.

### 7.5.5 Méta-caractères

```
méta-caractère =
        caractère backslash |
        caractère apostrophe |
        caractère guillemet |
        caractère back quote;

caractère backslash = "\";

caractère apostrophe = "'";

caractère guillemet = """;

caractère back quote = "`";
```

## 7.6 Table des codes des caractères

La table des codes des caractères est définie implicitement par l'association d'un entier à chaque caractère. Cette table dépend de l'implantation, avec les restrictions suivantes :

- les codes des lettres majuscules `A`, `B`, ... `Z` doivent être croissants,
- les codes des lettres minuscules `a`, `b`, ... `z` doivent être croissants,
- les codes des chiffres décimaux `0`, `1`, ... `9` doivent être croissants et contigus.

Les tables ASCII et EBCDIC satisfont ces contraintes.

## 7.7 Lexique

Voici les traductions utilisées de quelques termes et expressions utilisés par les rédacteurs de la norme ISO.

| | |
|---|---|
| back quote | back quote |
| backslash | backslash |
| built-in predicate | prédicat prédéfini |
| clause (in Prolog text) | clause (dans un texte Prolog) |
| close curly | accolade droite |
| close list | crochet droit |
| close | par droite |
| collating sequence | suite des codes internes des caractères |
| control construct | structure de contrôle |
| curly brackets | accolades |
| data (Prolog data) | donnée (donnée Prolog) |
| database (the Prolog database) | base de données (la base de données Prolog) |
| directive (in Prolog text) | directive (dans un texte Prolog) |
| double quoted char | caractère d'unité à guillemets |
| double quoted list / token | liste / unité à guillemets |
| flag | indicateur |
| ht sep | separ tête queue |
| layout (layout text) | décor (texte décoratif) |
| new line char | caractère de fin de ligne |
| open ct | par gauche contr |
| open curly | accolade gauche |
| open list | crochet gauche |
| open | par gauche |
| operator notation | notation opérationnelle |
| P/N (clause P/N) | P/N (P/N d'une clause) |
| predicate indicator | indicateur de prédicat |
| quote, quoted | quote, quoté |
| quoted char | caractère d'unité à apostrophes |
| quoted list / token | liste / unité à apostrophes |
| read term | terme lisible |
| single quoted char | caractère d'unité à apostrophes |
| single quoted list / token | liste / unité à apostrophes |
| solo char | caractère solitaire |
| specifier (of an operator) | spécificateur (d'un opérateur) |
| static procedure | procédure statique |
| text (Prolog text) | texte (texte Prolog) |
| underscore | blanc souligné |
| user-defined procedure | procédure définie par l'utilisateur |