Programmation Logique

Termes

Enseignant: NGUYEN Thi Minh Tuyen

Termes de comparaison: ==/2

- Prolog contient un prédicat important pour comparer des termes
  - ==/2
  - Ce prédicat n'instancie pas de variables
  - C'est à dire, il est différent de =/2
- Exemple:

```
?- a == a.
true.
?- a == b.
false.
```

```
?- a == 'a'.
true.
?- a == X.
false.
```

Comparaison des variables

- Deux variables non-instanciées différentes ne sont pas des termes identiques.
- Des variables instanciées avec un terme T sont identiques à T.
- Exemple:

```
?- X == X.
true.
?- Y == X.
false.
?- a = U, a == U.
U = a.
```

Termes de comparaison: \==/2

- Le prédicat \==/2
  - Vrai dans le cas où ==/2 échoue.
  - Il réussit lorsque deux termes ne sont pas identiques.
  - Il échoue s'il deux termes sont identiques.
- Exemple:

```
?- a \== a.
false.
?- a \== b.
true.
```

```
?- a \== 'a'.
false.
?- a \== X.
true.
```

Termes avec une notion spéciale

- Parfois, les termes semblent différents, mais Prolog les considère identiques.
  - Par exemple: a et `'a'`.
- Pourquoi Prolog fait-il cela?
  - Parce qu'il rendre la programmation plus intéresante.
  - Une façon plus naturelle pour programmer en Prolog.

Termes arithmétiques

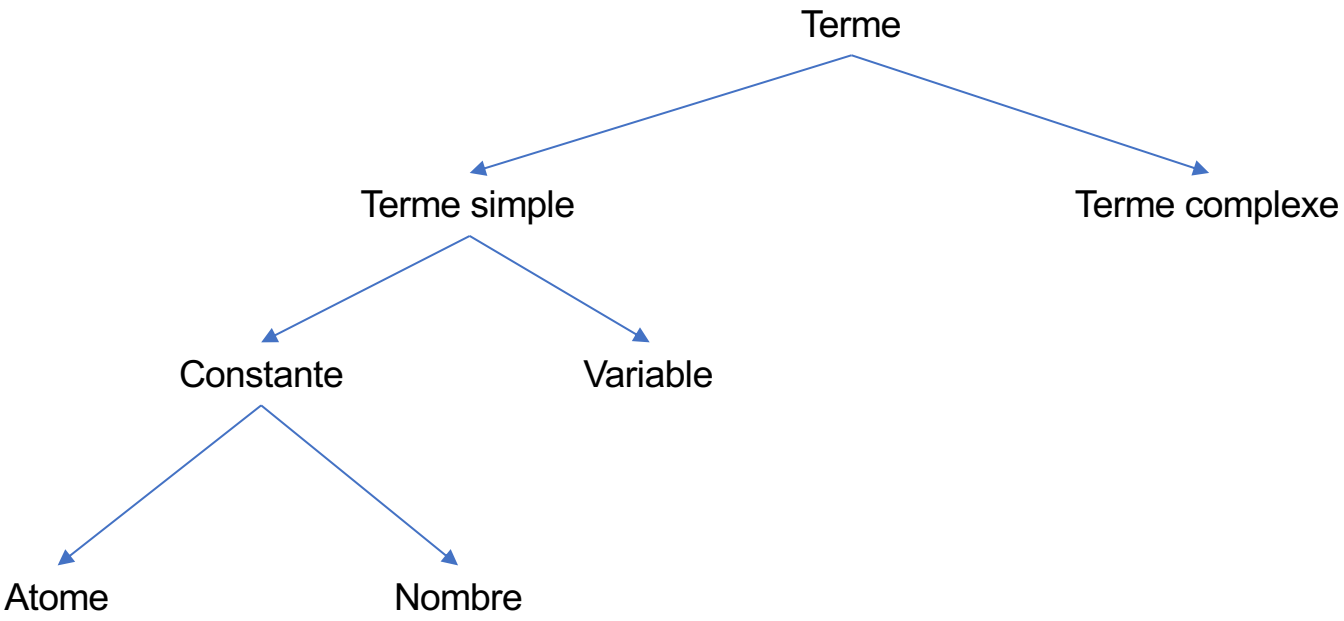
- Arithmétique est introduit dans le cours 5.
- `+, -, <, >`, etc. sont des foncteurs et des expressions telles que `2 + 3` sont des termes complexes.
- Le terme `2 + 3` est identique au terme `+(2,3)`.
- Exemple:

```
?- 2+3 == +(2,3).
true.
?- -(2,3) == 2-3.
true.
?- (4<2) == <(4,2).
true.
```

Prédicats de comparaison

=	Unification
\=	Négation de l'unification
==	Prédicat d'identité
\==	Négation du prédicat d'identité
=:=	Prédicat d'égalité arithmétique
=\=	Négation du prédicat d'égalité arithmétique

Rappel: Termes en Prolog



Prédicats de type

`atom/1`	Est-ce que c'est un atome?
`integer/1`	Est-ce que c'est un entier?
`float/1`	Est-ce que c'est un flottant?
`number/1`	Est-ce que c'est un nombre?
`atomic/1`	Est-ce que c'est une constante?
`var/1`	Est-ce que c'est une variable non instanciée?
`nonvar/1`	Le contraire de `var/1`.

Prédicats de type: atom


```
?- atom(a).
true.
?- atom(7).
false.
?- atom(X).
false.
?- X=a, atom(X).
X = a.
?- atom(X), X=a.
false.
```

Prédicats de type: `atomic`


```
?- atomic(mia).
true.
?- atomic(5).
true.
?- atomic(loves(vincent, mia)).
false.
```

Prédicats de type: var


```
?- var(mia).
false.
?- var(X).
true.
?- X=5, var(X).
false.
```

Prédicats de type: nonvar


```
?- nonvar(X).
false.
?- nonvar(mia).
true.
?- nonvar(23).
true.
```

Structure des termes

- Étant donné un terme complexe d'une structure inconnue, quel type d'information peut on en extraire ?
  - Le foncteur
  - L'arité
  - L'argument
- Prolog fournit des prédicats intégrés pour retirer ces informations.

Prédicat `functor/3`

- Donne le foncteur et l'arité d'une prédicat complexe.
- Par exemple:

```
?- functor(append([1,2,3],[4,5],[1,2,3,4,5]),F,A).
F = append,
A = 3.
?- functor([1,2,3],F,A).
F = '[|]',
A = 2.
```

functor/3 et constantes


```
?- functor(mia,F,A).
F = mia,
A = 0.

?- functor(14,F,A).
F = 14,
A = 0.
```

`functor/3` pour construire des termes


```
?- functor(Term,friends,2).
Term = friends(_G1093, _G1094).
```

Vérifier des termes complexes


```
complexTerm(X):-  nonvar(X),
                  functor(X,_,A),
                  A > 0.
```

Arguments: arg/3

- Prolog fournit le prédicat arg/3: des arguments de termes complexes.
- Trois aguments:
  - un nombre N
  - un terme complexe T
  - n^{ième} argument de T
- Exemple:

```
?- arg(2,likes(lou,andy),A).
A = andy.
```

Chaîne de caractères

- Chaînes de caractères sont représentées en Prolog par une liste de codes des caractères.
- Prolog permet d'utiliser des guillemets pour dénoter une chaîne de caractères.
- Par exemple:

```
?- S = "Vicky".
S = "Vicky".
```

Prédicat pour les chaînes de caractères

- `atom_codes/2:`

```
?- atom_codes(vicky,S).
S = [118, 105, 99, 107, 121].
```

Question?