Cours 4: Listes

- Sujets:
  - solution de l'exo add/3
  - Introduire les listes, une structure de données récursive fréquemment utilisée dans la programmation Prolog
  - Définir le prédicat member/2, un outil élémentaire pour la manipulation de listes en Prolog
  - Illustrer l'idée de traverser des listes de manière récursive
  - l'arithmétique en Prolog, le prédicat is/2

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Addition

?- add(succ(succ(0)),succ(succ(succ(0))), Resultat).
Resultat=succ(succ(succ(succ(succ(0)))))
true.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Addition

```
add(0,X,X).                    %%% clause de base
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Resultat).
Resultat=succ(succ(succ(succ(succ(0)))))
true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Addition

```
add(0,X,X).                          %%% clause de base

                                     %%% clause récurrente
add(succ(X) , Y ,  succ(Z))
      :-    add(X,Y,Z).
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Resultat).
Resultat=succ(succ(succ(succ(succ(0)))))
true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Listes

- Une liste est une séquence finie d'éléments
- Exemples de listes en Prolog:

[mia, vincent, jules, yolanda]
[mia, voleur(lapin), X, 2, mia]
[ ]
[mia, [vincent, jules], [butch, ami(butch)]]
[[ ], mort(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Syntaxe

- Une liste est encadrées par des [crochets]
- Les éléments d'une listes sont séparés par des virgules
- La longueur d'une liste est son nombre d'éléments
- Une liste peut contenir toute sorte de termes Prolog
- Une liste remarquable: la liste vide [ ]

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head-tête Tail-queue

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head-tête Tail-queue

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head-tête Tail-queue

TAIL

HEAD

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail

- Une liste non vide est constituée de deux parties:
  - sa tête (*head)*
  - sa queue (*tail*)
- La tête d'une liste est son premier élément
- La queue est tout ce qui suit.
  - La queue est ce qui reste après avoir éliminé le premier élément d'une liste
  - Toute queue de liste est une liste

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 1

- [mia, vincent, jules, yolanda]

  Head:
  Tail:

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 1

- [mia, vincent, jules, yolanda]

  Head: mia
  Tail:

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 1

- [mia, vincent, jules, yolanda]

  Head: mia
  Tail: [vincent, jules, yolanda]

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Head:
  Tail:

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Head: [ ]
  Tail:

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Head: [ ]
  Tail: [dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 3

- [dead(z)]

  Head:
  Tail:

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 3

- [dead(z)]

  Head: dead(z)
  Tail:

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Head & Tail Exemple 3

- [dead(z)]

  Head: dead(z)
  Tail: [ ]

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

La liste vide

- La liste vide n'a ni tête, ni queue
- En Prolog, [ ] est une liste simple spéciale sans aucune structure interne

- La liste vide joue un rôle central dans les prédicats récurrents pour listes en Prolog: elle est typiquement utilisée pour la clause de base.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

L'opérateur |

- Permet de décomposer une liste en tête et queue.
- Joue un rôle central dans la définition de prédicats pour la manipulation de listes en Prolog: dans la clause récurrente.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

L’opérateur |

```
?- [Head|Tail] = [mia, vincent, jules, yolanda].

Head = mia
Tail = [vincent,jules,yolanda]
true.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

L’opérateur |

```
?- [X|Y] = [mia, vincent, jules, yolanda].

X = mia
Y = [vincent,jules,yolanda]
true.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

L’opérateur |

```
?- [X|Y] = [ ].

false.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

L’opérateur |

```
?- [X,Y|Tail] = [[ ], dead(z), [2, [b,c]], [], Z, [2, [b,c]]] .

X = [ ]
Y = dead(z)
Z = _4543
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]]
true.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exercise 4.1 de LPN

```
   1.  [a,b,c,d] = [a,[b,c,d]].
2. [a,b,c,d] = [a|[b,c,d]].
3. [a,b,c,d] = [a,b,[c,d]].
4. [a,b,c,d] = [a,b|[c,d]].
5. [a,b,c,d] = [a,b,c,[d]].
6. [a,b,c,d] = [a,b,c|[d]].
7. [a,b,c,d] = [a,b,c,d,[]].
8. [a,b,c,d] = [a,b,c,d|[]].
9. [] = _.
10.   [] = [_].
11.   [] = [_|[]].
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Member

- Comment déterminer si une liste contient un certain élément?
- Définissons un prédicat qui, pour un terme X et une liste L, détermine si X est contenu dans L.
- Ce prédicat s'appelle d'habitude member/2

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

```
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

```
?- member(yolanda,[yolanda,trudy,vincent,jules]).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T])  :- member(X,T).
```

```
?- member(yolanda,[yolanda,trudy,vincent,jules]).
true.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T])  :- member(X,T).
```

```
?- member(vincent,[yolanda,trudy,vincent,jules]).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T])  :- member(X,T).
```

```
?- member(vincent,[yolanda,trudy,vincent,jules]).
true.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T])  :- member(X,T).
```

```
?- member(zed,[yolanda,trudy,vincent,jules]).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

```
?- member(zed,[yolanda,trudy,vincent,jules]).
false.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

```
?- member(X,[yolanda,trudy,vincent,jules]).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

```
?- member(X,[yolanda,trudy,vincent,jules]).
X = yolanda;
X = trudy;
X = vincent;
X = jules;
false.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

member/2 reformulé

```
member(X,[X| _ ]).
member(X,[ _ | T]) :- member(X,T).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Requêtes

```
member( X , [ X | _ ] ).
member( X , [ _  | T ] ) :- member( X , T ).
```

```
?- member(a,[c,b,a,y]).
true.
?- member(x,[a,b,c]).
false.
```

Exo: construisez les arbres de résolution!

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Arbre 1

```
   ?- member(a,[c,b,a,y]).
        |
?- member(a,[b,a,y])
        |
?- member(a,[a,y])
        |
    true
```

unifie avec premier prédicat

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Arbre 2

```
    ?- member(x,[a,b,c]).
       |
?- member(x,[b,c]).
       |
?- member(x,[c]).
       |
?- member(x,[]).
        |
   fail
```

n'unifie avec aucun prédicat

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

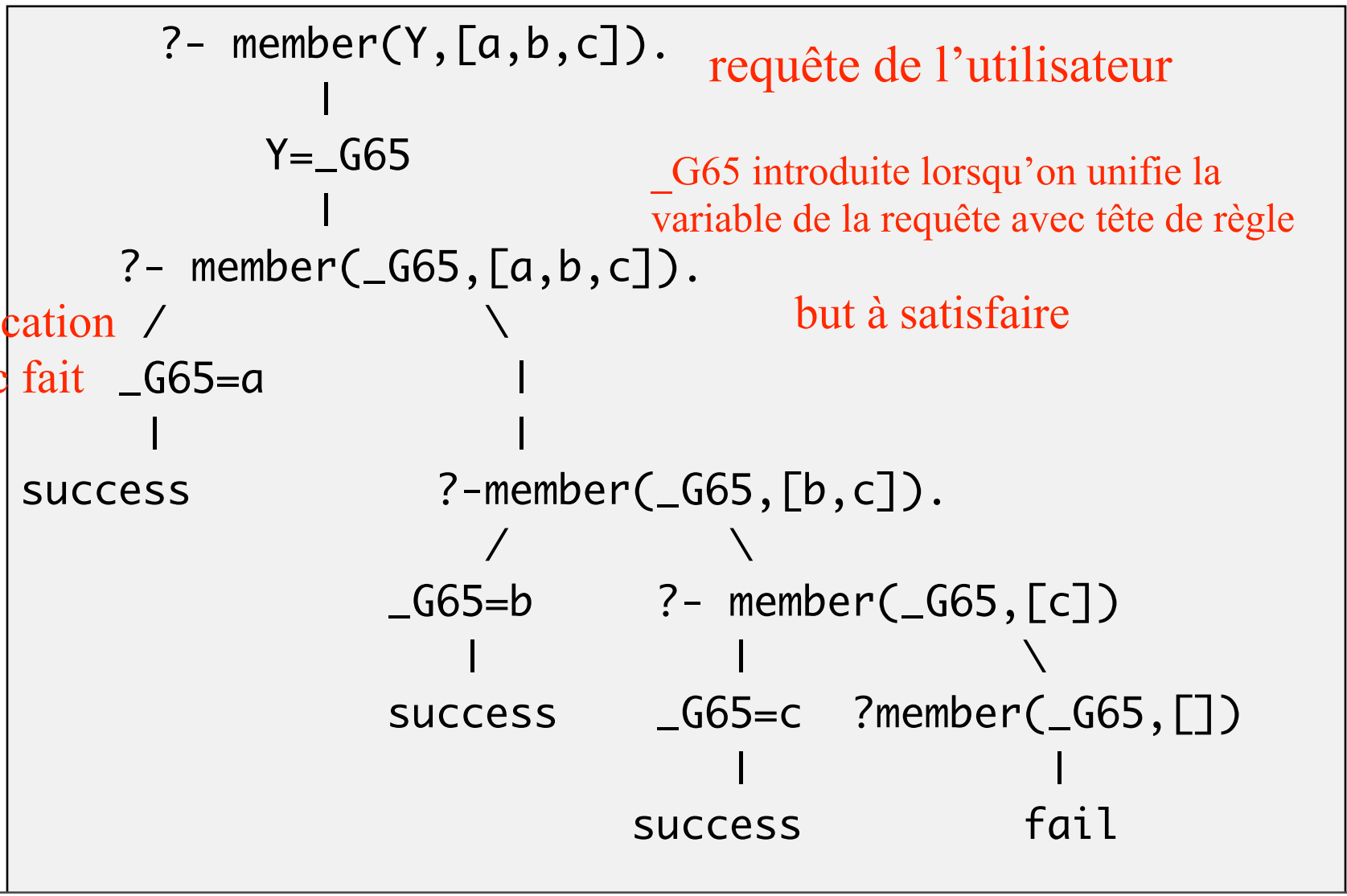
Requête avec variable

```
member(X, [ X | _ ] ).
member(X, [ _ | T ] ) :- member(X,T).
```

```
?- member(Y,[a,b,c]).
Y = a ;                 % utilisateur force retour en arrière
Y = b;                  %  après premier succès
Y = c;
false.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Arbre



© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Récurrence sur listes

- Le principe du prédicat member/2 est de traverser une liste de manière récurrente
  - faire quelque chose avec la tête, puis
  - continuer a faire la même chose avec la queue de la liste, etc.
- Cette technique est très fréquente en Prolog, la maîtriser est essentiel.
- Regardons un autre exemple!

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exemple: a2b/2

- Le prédicat a2b/2 prend deux listes et réussi si
  - le premier argument est une liste d'*a*s
  - le deuxième argument est une liste de *b*s de la même longueur.

```
?- a2b([a,a,a,a],[b,b,b,b]).
true.
?- a2b([a,a,a,a],[b,b,b]).
false.
?- a2b([a,c,a,a],[b,b,b,t]).
false.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Définir a2b/2: étape 1

```
a2b( [ ] , [ ] ).
```

- Souvent, le meilleur moyen de résoudre de tels problèmes est de commencer par le cas le plus simple possible.
- Ici: la liste vide

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Définir a2b/2: étape 2

```
a2b( [ ] , [ ] ).
a2b( [a | L1 ] , [ b | L2 ] ) :- a2b( L1 , L2).
```

- Maintenant, penser récursivement!
- Quand a2b/2 décidera-t-il que deux listes non vides ne contiennent que des a et bs, et ont la même longueur?

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Tester a2b/2

```
a2b( [ ] , [ ] ).
a2b( [a | L1 ] , [b | L2 ] ) :- a2b( L1 , L2).
```

```
?- a2b([a,a,a],[b,b,b]).
true.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Tester a2b/2

```
a2b( [ ] , [ ] ).
a2b( [a | L1 ] , [b | L2 ] ) :- a2b( L1 , L2).
```

```
?- a2b([a,a,a,a],[b,b,b]).
false.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Tester a2b/2

```
a2b( [ ] , [ ] ).
a2b( [a | L1 ] , [b | L2 ] ) :- a2b( L1 , L2).
```

```
?- a2b([a,t,a,a],[b,b,b,c]).
false.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Générer avec a2b/2

```
a2b( [ ] , [ ] ).
a2b( [a | L1 ] , [b | L2 ] ) :- a2b( L1 , L2).
```

```
?- a2b([a,a,a,a,a], X).
X = [b,b,b,b,b]
true.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Générer avec a2b/2

```
a2b( [ ] , [ ] ).
a2b( [a | L1 ] , [b | L2 ] ) :- a2b( L1 , L2).
```

```
?- a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
true.
?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Arithmétique en Prolog

- Prolog fournit quelques outils simples pour l'arithmétique
- Entiers et nombres réels

Arithmétique	Prolog
2 + 3 = 5	?- 5 is 2+3.
3 x 4 = 12	?- 12 is 3*4.
5 – 3 = 2	?- 2 is 5-3.
3 – 5 = -2	?- -2 is 3-5.
4 : 2 = 2	?- 2 is 4/2.
1 is the remainder when 7 is divided by 2	?- 1 is mod(7,2).

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exemples de requêtes

```
?- 10 is 5+5.
true.

?- 4 is 2+3.
false.

?- X is 3 * 4.
X=12
true.

?- R is mod(7,2).
R=1
true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Définir des prédicats arithmétiques

```
ajoute_3_et_double(X, Y):-
    Y is (X+3) * 2.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Définir des prédicats arithmétiques

```
ajoute_3_et_double(X, Y):-
    Y is (X+3) * 2.
```

```
?- ajoute_3_et_double(1,X).
X=8
true.

?- ajoute_3_et_double(2,X).
X=10
true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Vu de plus près

- Il faut comprendre que +, -, / et * ne font pas d'arithmétique
- Les expressions comme 3+2, 4-7, 5/5 sont des termes simples
  - Foncteurs: +, -, /, *
  - Arité: 2
  - Arguments: entiers

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Vu de plus près

```
?- X = 3 + 2.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Vu de plus près

```
?- X = 3 + 2.
X = 3+2          % notez: le terme 3+2 n'est pas évalué!!
true.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Vu de plus près

```
?- X = 3 + 2.
X = 3+2
true.

?- 3 + 2 = X.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Vu de plus près

```
?- X = 3 + 2.
X = 3+2
true.

?- 3 + 2 = X.    % l'opérateur d'unification = est commutatif!
X = 3+2
true.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

is(-Nombre, +Expr)

- Pour forcer Prolog à évaluer les expressions arithmétiques on doit utiliser **is** comme nous l'avons fait ici.
- Ceci fait quitter le mode de raisonnement habituel de Prolog
- Pas surprenant qu'il y ait des restrictions à cette capacité d'évaluation!

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.t

?- 3 + 2 is X.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- Result is 2+2+2+2+2.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- Result is 2+2+2+2+2.
Result = 10
true.

?-
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- Result is 2+2+2+2+2.
Result = 10
true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- 2+2 is 4.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- 2+2 is 4.
false.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- 2+2 is 4.
false.

?- ajoute_3_et_double(X,12).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

```
?- X is 3 + 2.
X = 5
true.

?- 3 + 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated

?- 2+2 is 4.
false.

?- ajoute_3_et_double(A,12).
ERROR: is/2: Arguments are not sufficiently instantiated
   % ajoute_3_et_double(X, Y):- Y is (X+3) * 2.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Le prédicat is/2

- Deux dernières remarques sur les expressions arithmétiques
  - 3+2, 4/2, 4-5 sont des des termes Prolog simples avec un notation commode: **3+2** est en réalité **+(3,2)** etc.
  - Le prédicat **is** est un prédicat Prolog à deux arguments

is(-Nombre, +Expr)

```
?- is(X,+(3,2)).
X = 5
true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Restrictions d'utilisation de is/2

- Les expressions à droite de **is** *peuvent* contenir des variables
- Mais quand Prolog les évalue, ces variables doivent être instanciées par un terme Prolog
  - sans variables
  - représentant une expression arithmétique

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Modes de passage d'arguments

swiprolog help:

\+ Argument must be fully instantiated to a term that satisfies the required argument type. Think of the argument as input.

\- Argument must be unbound. Think of the argument as output.

? Argument must be bound to a partial term of the indicated type. Note that a variable is a partial term for any type. Think of the argument as either input or output or both input and output.

is(-Nombre, +Expr)

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.
3. ERROR: Arguments are not sufficiently instantiated.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.
3. ERROR: Arguments are not sufficiently instantiated.
4. X = Y.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.
3. ERROR: Arguments are not sufficiently instantiated.
4. X = Y.
5. true.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.
3. ERROR: Arguments are not sufficiently instantiated.
4. X = Y.
5. true.
6. true.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.
3. ERROR: Arguments are not sufficiently instantiated.
4. X = Y.
5. true.
6. true.
7. ERROR: Arguments are not sufficiently instantiated.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.
3. ERROR: Arguments are not sufficiently instantiated.
4. X = Y.
5. true.
6. true.
7. ERROR: Arguments are not sufficiently instantiated.
8. X = 3.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
1. X = 3*4.

2. X is 3*4.

3. 4 is X.

4. X = Y.

5. 3 is 1+2.

6. 3 is +(1,2).

7. 3 is X+2.

8. X is 1+2.

9. 1+2 is 1+2.
```

1. X = 3*4. Variable X est instantiée avec le terme complexe 3*4.
2. X = 12.
3. ERROR: Arguments are not sufficiently instantiated.
4. X = Y.
5. true.
6. true.
7. ERROR: Arguments are not sufficiently instantiated.
8. X = 3.
9. fail. Prolog evalue l'expression arithmétique à droite de is/2. Puis il essaie d'unifier ce terme avec le terme à gauche de is/2. Ceci échoue comme le nombre 3 n'unifie pas avec le terme complexe 1+2.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).


12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

Réponses:

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).


12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).


12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

Réponses:

```
10.  X = 3.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).

12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

Réponses:

10. X = 3.

11. true. 3+2 et +(3,2) sont deux manières d'écrire le même terme.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).

12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

Réponses:

10. X = 3.

11. true. 3+2 et +(3,2) sont deux manières d'écrire le même terme.

12. true.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).

12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

Réponses:

```
10.  X = 3.

11.  true. 3+2 et +(3,2) sont
deux manières d'écrire le
même terme.

12.  true.

13.  true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).

12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

Réponses:

10. X = 3.

11. true. 3+2 et +(3,2) sont deux manières d'écrire le même terme.

12. true.

13. true.

14. true.

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).

12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

Réponses:

```
10.  X = 3.

11.  true. 3+2 et +(3,2) sont
deux manières d'écrire le
même terme.

12.  true.

13.  true.

14.  true.

15.  fail.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Exo 5.1: Comment Prolog répond-il?

```
10. is(X,+(1,2)).

11. 3+2 = +(3,2).

12. *(7,5) = 7*5.

13. *(7,+(3,2)) = 7*(3+2).

14. *(7,(3+2)) = 7*(3+2).

15. 7*3+2 = *(7+(3,2))

15. *(7,(3+2)) = 7*(+(3,2)).
```

Réponses:

```
10.  X = 3.

11.  true. 3+2 et +(3,2) sont
deux manières d'écrire le
même terme.

12.  true.

13.  true.

14.  true.

15.  fail.

16.  true.
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

Résumé de ce cours

- Nous avons introduit des prédicats récurrents sur les listes.
- Ce style de programmation est fondamental pour Prolog!

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

last/2

- last(X,L): vrai si X est le dernier élément de la liste L
- cas de base: une liste avec un seul élément
- si X est le dernier élément de T, il le reste lorsqu'on ajoute une tête à T

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

last/2

- last(X,L): vrai si X est le dernier élément de la liste L
- cas de base: une liste avec un seul élément
- si X est le dernier élément de T, il le reste lorsqu'on ajoute une tête à T

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

last/2

- last(X,L): vrai si X est le dernier élément de la liste L
- cas de base: une liste avec un seul élément
- si X est le dernier élément de T, il le reste lorsqu'on ajoute une tête à T

© Patrick Blackburn, Johan Bos & Kristina Striegnitz

last/2

- last(X,L): vrai si X est le dernier élément de la liste L
- cas de base: une liste avec un seul élément
- si X est le dernier élément de T, il le reste lorsqu'on ajoute une tête à T

```
last(X,[X]).

last(X,[_|T])  :- last(X,T).
```

© Patrick Blackburn, Johan Bos & Kristina Striegnitz