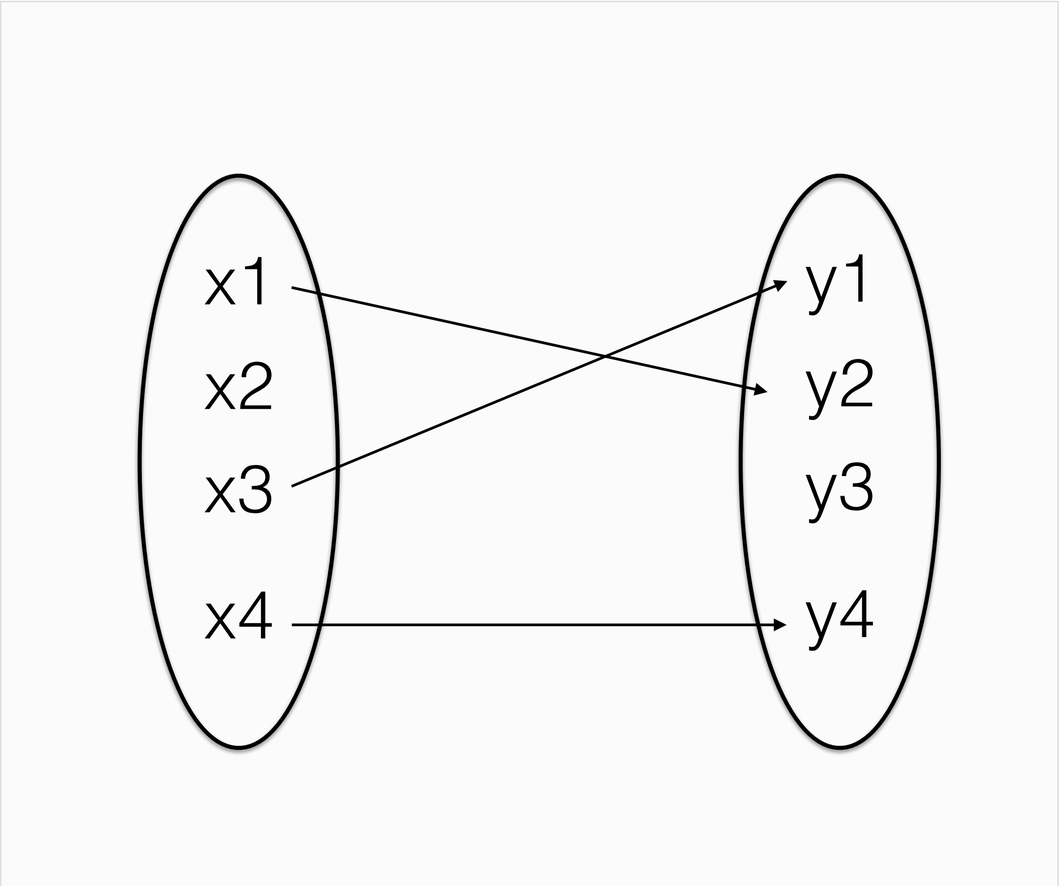
Programmation logique

Vincent Archambault-B
IFT 2035 - Université de Montréal

Fonctions

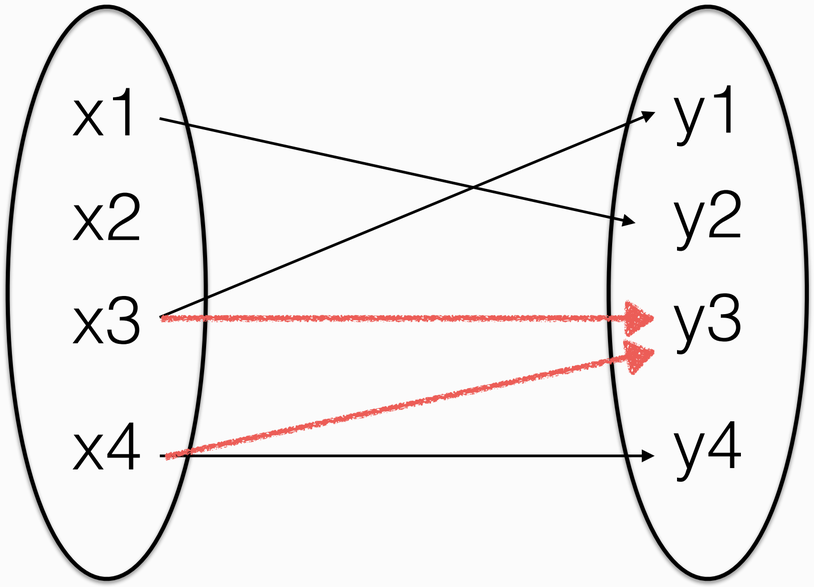
- Ensemble de départ X
- Ensemble d'arrivée Y
- Relation binaire qui met en relation au plus 1 élément de Y avec chaque élément de X



Fonction

Relations binaires

- Ensemble de départ X
- Ensemble d'arrivée Y
- Ensemble de tuple (X, Y). Pour un même élément de X, il peut y avoir plusieurs éléments de Y dans la relation



Relation binaire

Relations n-aire

- Généralisation des relations binaires pour n ensembles X1 … Xn

Programmation impérative

- Style de programmation basé sur une machine à états
- Le programmeur écrit les instructions que la machine doit suivre pour faire un calcul

Programmation fonctionnelle

- Style de programmation qui utilise les fonctions
- Le programmeur écrit les fonctions qui doivent être évaluées pour faire un calcul

Programmation logique

- Style de programmation qui utilise les relations
- Le programmeur écrit les relations qui doivent être «évaluées» pour faire un calcul
- Prolog (1972) est le langage de programmation logique le plus connu

Exemple de programmation logique

```
% Relation qui définit le genre
genre(luc, homme).
genre(julie, femme).
genre(thomas, homme).

% Relation qui définit mâle
male(X) :- genre(X, homme).
```

Prolog

```
% Trouver tous les hommes
genre(X, homme).

% Luc est-il un homme ou une femme ?
genre(luc, X).
```

suite …

La programmation logique va dans les deux sens

- Trouver le genre de luc. Luc est une entrée, X est la sortie. Comme une fonction
- Trouver qui sont les hommes. homme est une entrée, X est la sortie. Sens inverse d'une fonction
- Dans les deux cas, il s'agit de la même relation genre

```
% Trouver tous les hommes
genre(X, homme).

% Luc est-il un homme ou une femme ?
genre(luc, X).
```

Les relations s'évaluent dans les deux sens

Programmation logique : qu'est-ce qu'un programme ?

- Ensemble de relations
- Une requête pour «évaluer» une relation en particulier

Syntaxe de base de Prolog

var	identificateur débutant avec majuscule
atome	identificateur débutant avec minuscule
relation	atome (terme)
terme	nombre | var | atome | relation
fait	terme .
règle	terme :- terme {, terme}*
programme	{fait | règle}$^{+}$

Écrire une relation en prolog

- Énumération de faits

```
genre(luc, homme).
genre(julie, femme).
genre(thomas, homme).
```

Relation définie à partir de fait

Écrire une relation en prolog

- Utilisation de règles

```
genre(luc, homme).
genre(julie, femme).
genre(thomas, homme).

male(X) :- genre(X, homme).
```

Relation définie à partir d'une règle

Écrire une relation en prolog

- Énumération de faits et utilisation de règles

```
lien(a, b).
lien(a ,c).
lien(c, d).
lien(c, e).
lien(d, e).

chemin(Z, Z).
chemin(X, Y) :- lien(X, I), chemin(I,Y).
```

Relation définie à partir d'un fait et d'une règle

Évaluer une relation : substitution

terme clos	terme qui ne contient pas de variable
terme non clos	terme avec variables. Interprété comme l'ensemble infini que l'on peut obtenir en substituant les variables par des termes clos
substitution	liste de couple associant une variable à un terme

Évaluer une relation : substitution

```
% Terme non clos
f(a, X)

% Substitutions possibles
f(a, 0) % Substitution (X,0)
f(a, 1)
f(a, 2)

f(a, a)
f(a, b)

f(a,f(a,a)) % Substitution (X, f(a,a))
```

Exemple de substitution

```
% Terme non clos
f(X,X)

% Substitutions possibles
f(0, 0) % Substitution (X,0)
f(1, 1)
f(2, 2)

f(a, a)
f(b, b)

f(f(a,a),f(a,a))
```

Exemple de substitution

Évaluer une relation : unification

unificateur	une substitution qui rend deux termes égaux
unificateur le plus général	unificateur qui comprend toutes les instances communes des deux termes
unification	trouver un unificateur (idéalement le plus général)

Évaluer une relation : unificateur

```
% Terme non clos
a(X, Y).
a(Y, X).

% La substitution [(X,1), (Y, 1)]
% est un unificateur
a(1,1).
a(1,1).
```

Exemple d'unificateur

```
% Terme non clos
a(X, Y).
a(Y, X).

% La substitution [(X,Y)]
% est l'unificateur le plus général
a(Y,Y).
a(Y,Y).
```

Exemple d'unificateur le plus général

Évaluer une relation

- Évaluer une relation consiste à trouver un unificateur
- Évaluer un programme prolog consiste à trouver un unificateur qui satisfait la requête (règle principale)

Évaluer un programme

- L'évaluation d'un programme consiste à trouver l'unificateur de la requête
- La requête f(3, A, A) peut être unifiée avec [(A, 2)] sous la contrainte que les relations égal et f respectées
- Le résultat du programme est donc [(A, 2)]

```
egal(X,X).

f(X, Y, Z) :- egal(X,Y), egal(Z, 1).
f(X, Y, Z) :- egal(Y,Z), egal(Z, 2).
```

Programme prolog simple

Comment trouver un unificateur ?

- Trouver un unificateur est équivalent à prouver qu'il existe une substitution qui respecte toutes les relations du programme.
- L'algorithme pour trouver / prouver cette substitution est le suivant :

1. Chercher séquentiellement un fait ou une règle applicable
2. Instancier les variables présentes dans le fait ou la règle
3. Pour chaque prémisse de la règle, recommencer au point 1

Évaluer un programme

- La requête est f(3, A, A)

1. egal(X,X) n'est pas applicable.
   f≠egal
2. f(X, Y, Z) :- egal(X,Y), egal(Z, 1)
   est applicable
3. instancier X = 3, Y = A, Z = A
4. La première prémisse est : egal(3,A)
5. egal(X,X) est applicable
6. instancier X = 3, X = A, donc A = 3
7. La deuxième prémisse est : egal(3,1)
8. egal(X,X) n'est pas applicable car X = 3 et X =1 sont impossible

```
egal(X,X).

f(X, Y, Z) :- egal(X,Y), egal(Z, 1).
f(X, Y, Z) :- egal(Y,Z), egal(Z, 2).
```

Programme prolog simple

Évaluer un programme

- La requête est f(3, A, A) … suite

9. f(X, Y, Z) :- egal(Y,Z), egal(Z, 2)
   est applicable
10. instancier X = 3, Y = A, Z = A
11. La première prémisse est : egal(A,A)
12. egal(X,X) est applicable
13. instancier X = A, X = A
14. La deuxième prémisse est : egal(A,2)
15. egal(X,X) est pas applicable
16. instancier A = 2
17. Toutes les relations applicables sont satisfaites,
    fin du programme

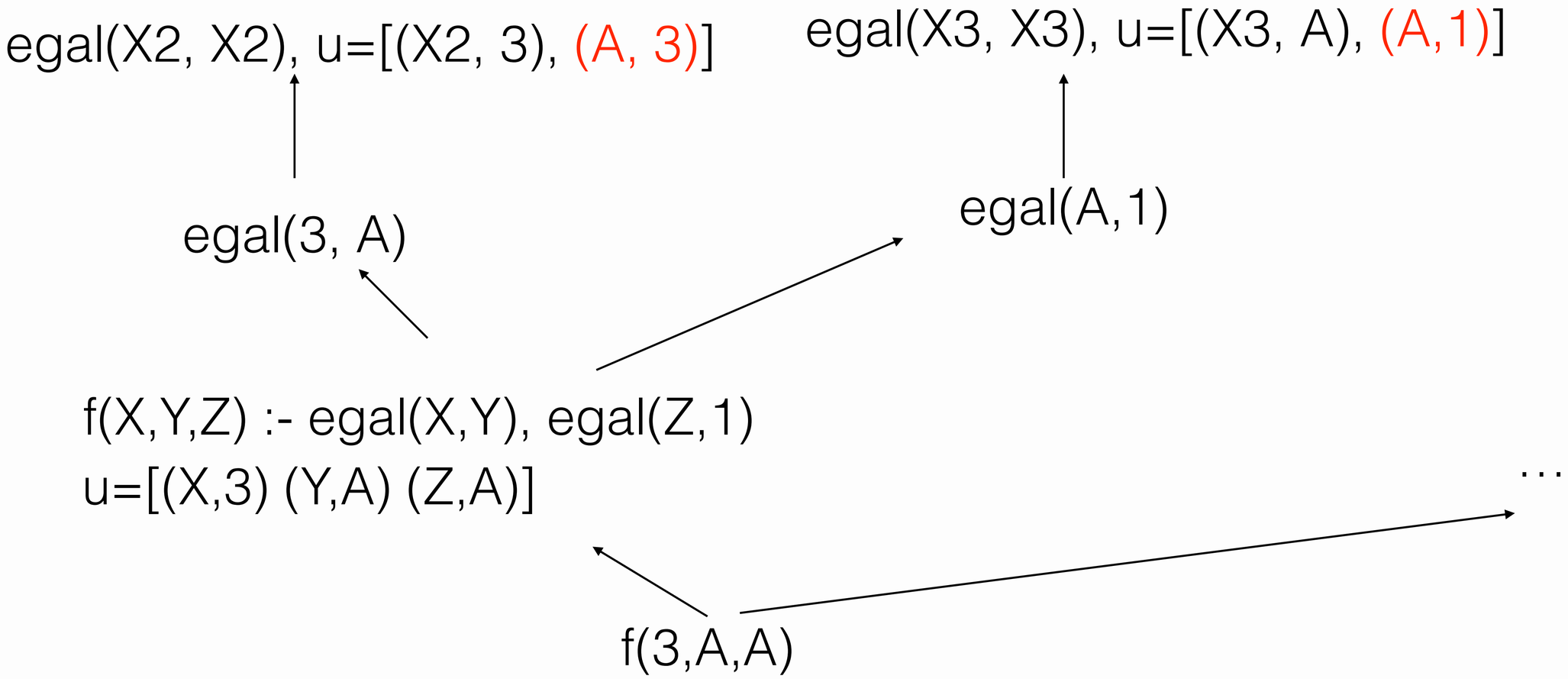
```
egal(X,X).

f(X, Y, Z) :- egal(X,Y), egal(Z, 1).
f(X, Y, Z) :- egal(Y,Z), egal(Z, 2).
```

Programme prolog simple

Arbre de preuves

- Les étapes de l'algorithme d'unification se représente bien à l'aide d'un arbre
- Cet arbre représente les étapes d'une preuve qu'il existe un unificateur

Arbre de preuves



Arbre de preuves