

Analyse de Données Structurées - Cours 12

Ralf Treinen



Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes

treinen@pps.univ-paris-diderot.fr

6 mai 2015

© Ralf Treinen 2015

Exemple d'un programme

```
var
  x: integer;
  y: integer;
  b: boolean;
  c: boolean;
begin
  x := 73;
  if b&&c||!b then
    print 42+1*x+y;
  y := 42;
  else
    y := x+x;
  fi;
  while 42>41 do print 73; od;
end
```

Rappel

- ▶ Jusqu'à maintenant : un petit langage de programmation impératif
 - ▶ Deux types : integer et boolean
 - ▶ Expressions avec des opérateurs arithmétiques et logiques
 - ▶ Déclaration des variables avec leur type au début du programme
 - ▶ Instructions d'affichage et d'affectation
 - ▶ Instructions composées : boucle while ... do ... od, conditionnelle if ... then ... else ...fi

Un langage avec des déclaration locales

- ▶ Extension du langage : déclarations locales
- ▶ On veut maintenant aussi permettre des déclarations de variables avec portée limitée.
- ▶ Pour cela il y a une décision fondamentale à prendre :
est-ce qu'on veut permettre une redéclaration locale d'une variable ?

Exemple : variable locale

```

1  var
2    x: integer;
3    y: integer;
4  begin
5    x := 73;
6    y := x+42;
7    var
8      z: integer;
9    begin
10     z := x + y;
11     print(z);
12   end;
13   print(x+y);
14 end

```

Java : redéclaration pas autorisée

```

/* Erreur de compilation */
class Scope {
    public static void main(String[] args) {
        int x = 42;
        {
            int x = 73;
        }
    }
}

```

Exemple : redéclaration d'une variable

```

1  var
2    x: integer;
3    y: integer;
4  begin
5    x := 73;
6    y := x+42;
7    var
8      x: boolean;
9    begin
10     x := y>17;
11     if x then print(1); else print(2); fi;
12   end;
13   print(x+y);
14 end

```

Redéclaration de variables

- ▶ En Java : la redéclaration d'une variable dans un bloc n'est pas permise (erreur de compilation).
- ▶ Il y a des autres langages de programmation qui admettent des redéclaration locales de variable.
- ▶ Les redéclaration sont aussi pertinentes pour l'implémentation de procédures qui peuvent accéder à des variables globales.
- ▶ Pour cette raison nous allons étudier les deux solutions.

Extension de la syntaxe abstraite

- ▶ L'ensemble DL des Listes de Déclarations est ...
- ▶ L'ensemble IL des listes d'Instructions est ...
- ▶ L'ensemble I des Instructions en syntaxe abstraite est :
 - ▶ si $e \in E'''$ alors $Print(e) \in I$,
 - ▶ si $s \in \Sigma^*$, $e \in E'''$ alors $Assign(s, e) \in I$,
 - ▶ si $e \in E'''$, $il_1, il_2 \in IL$ alors $Cond(e, il_1, il_2) \in I$,
 - ▶ si $e \in E'''$, $il \in IL$ alors $While(e, il) \in I$,
 - ▶ si $dl \in DL$ et $il \in IL$ alors $Bloc(dl, il) \in I$.

Modification : exécution de déclarations

- ▶ Jugement $dl \Rightarrow \Gamma_i, \Gamma_b$: l'exécution de la liste de déclarations dl crée un environnement entier Γ_i , et un environnement booléen Γ_b .
- ▶ Règles :

$$\overline{DeclNil \Rightarrow \emptyset, \emptyset}$$

$$\frac{dl \Rightarrow \Gamma_i, \Gamma_b}{DeclSeq(Decl(s, \text{int}), dl) \Rightarrow \Gamma_i[s \mapsto 0], \Gamma_b}$$

$$\frac{dl \Rightarrow \Gamma_i, \Gamma_b}{DeclSeq(Decl(s, \text{bool}), dl) \Rightarrow \Gamma_i, \Gamma_b[s \mapsto \text{false}]}$$

Petite modification de la sémantique

- ▶ Jusqu'à maintenant, une variable n'a pas reçu une valeur initiale au moment de la déclaration.
- ▶ Modification : au moment de la déclaration, la variable reçoit une valeur initiale (0 pour les entiers, *false* pour les booléens).
- ▶ Raison : simplification technique.
- ▶ On aurait aussi pu étendre la syntaxe abstraite par une expression d'initialisation au moment de la déclaration d'une variable.

Modification : exécution d'un programme

- ▶ Jugement $\Gamma_i, \Gamma_b \vdash il \Rightarrow \Gamma'_i, \Gamma'_b$: l'exécution de la liste d'instructions il dans un environnement initial Γ_i, Γ_b donne un nouvel environnement Γ'_i, Γ'_b
- ▶ Règle modifiée pour un programme complet :

$$\frac{dl \Rightarrow \Gamma_i, \Gamma_b \quad \Gamma_i, \Gamma_b \vdash il \Rightarrow \Gamma'_i, \Gamma'_b}{Prog(dl, il) \Rightarrow \Gamma'_i, \Gamma'_b}$$

Deux opérations sur les environnements

- Union disjointe : si Γ_1 et Γ_2 sont deux environnements (de typage ou de valeur) avec des *domaines disjoints*, alors $\Gamma_1 \oplus \Gamma_2$ est défini comme

$$(\Gamma_1 \oplus \Gamma_2)(x) = \begin{cases} \Gamma_1(x) & \text{si } x \in \text{domaine}(\Gamma_1) \\ \Gamma_2(x) & \text{sinon} \end{cases}$$

- Restriction : si Γ est un environnement, et $D \subseteq \text{domaine}(\Gamma)$, alors la restriction de Γ à D est défini comme

$$\Gamma|_D(x) = \begin{cases} \Gamma(x) & \text{si } x \in D \\ \text{non-défini} & \text{sinon} \end{cases}$$

Exécution des blocs

$$\frac{dl \Rightarrow \Gamma'_i, \Gamma'_b \quad \Gamma_i \oplus \Gamma'_i, \Gamma_b \oplus \Gamma'_b \vdash il \Rightarrow \Gamma''_i, \Gamma''_b}{\Gamma_i, \Gamma_b \vdash \text{Bloc}(dl, il) \Rightarrow \Gamma''_i|_{\text{domaine}(\Gamma_i)}, \Gamma''_b|_{\text{domaine}(\Gamma_b)}}$$

Typage pour les blocs

- Rappel des jugements de typage :
 - $dl : \Gamma$: la liste de déclarations dl donne lieu à l'environnement de typage Γ .
 - $\Gamma \vdash il$: la liste d'instructions il est bien typée par rapport à l'environnement Γ .

- Règle

$$\frac{dl \Rightarrow \Gamma' \quad \Gamma \oplus \Gamma' : il}{\Gamma : \text{Bloc}(dl, il)}$$

Une modélisation plus complexe

- La technique des extensions et restrictions des environnements ne fonctionne plus dans le cas des redéclarations de variable.
- Raison : Une redéclaration d'une variable x masque, pour la durée d'exécution du bloc, une déclaration globale de x . Après avoir terminé le bloc, la variable x globale devient de nouveau visible.
- Nous avons donc maintenant besoin d'une liste d'environnements : quand on entre dans un bloc, un nouvel environnement est ajouté à la fin de la liste; quand on quitte un bloc, l'environnement le plus récent est supprimé.

Listes d'environnements

- ▶ Dans une liste d'environnements, l'environnement le plus globale se trouve à gauche, et l'environnement le plus local se trouve à droite.
- ▶ Pour obtenir la valeur d'une variable, ou pour la modifier, on parcourt la liste de droite à gauche : priorité à la déclaration la plus locale.
- ▶ Pour ne pas trop alourdir la notation nous ignorons maintenant la distinction entre environnement entier et environnement booléen.

Règles à modifier pour les listes d'environnements

- ▶ Évaluation d'une variable

$$\frac{}{\Gamma_1, \dots, \Gamma_n \vdash \text{Ident}(s) \rightarrow \text{get}(s, (\Gamma_1, \dots, \Gamma_n))}$$

- ▶ Affectation d'une variable

$$\frac{\Gamma_1, \dots, \Gamma_n \vdash e \Rightarrow v}{\Gamma_1, \dots, \Gamma_n \vdash \text{Assign}(s, e) \Rightarrow \text{set}(s, v, (\Gamma_1, \dots, \Gamma_n))}$$

Opérations sur les listes d'environnements

- ▶ L'opération *get* permet de récupérer la valeur d'une variable :

$$\text{get}(s, (\Gamma_1, \dots, \Gamma_n)) = \begin{cases} \Gamma_n(s) & \text{si } s \in \text{domaine}(\Gamma_n) \\ \text{get}(s, (\Gamma_1, \dots, \Gamma_{n-1})) & \text{sinon} \end{cases}$$

- ▶ L'opération *set* permet de modifier la valeur d'une variable :

$$\text{set}(s, v, (\Gamma_1, \dots, \Gamma_n)) = \begin{cases} (\Gamma_1, \dots, \Gamma_{n-1}, \Gamma_n[s \mapsto v]) & \text{si } s \in \text{domaine}(\Gamma_n) \\ \text{set}(s, v, (\Gamma_1, \dots, \Gamma_{n-1})), \Gamma_n & \text{sinon} \end{cases}$$

Nouvelle règle pour l'exécution d'un bloc

$$\frac{dl \Rightarrow \Gamma_{n+1} \quad \Gamma_1, \dots, \Gamma_n, \Gamma_{n+1} \vdash il \Rightarrow \Gamma'_1, \dots, \Gamma'_n, \Gamma'_{n+1}}{\Gamma_1, \dots, \Gamma_n \vdash \text{Bloc}(dl, il) \Rightarrow \Gamma'_1, \dots, \Gamma'_n}$$

Extension de la syntaxe

- ▶ Syntaxe concrète : définition d'une procédure (à un argument)
par
proc *nom*(*variable* : *type*)
bloc
- ▶ Syntaxe abstraite :
 - ▶ définition d'une procédure : nom de la procédure, nom du paramètre, type du paramètre, bloc.
 - ▶ liste de définitions de procédures : ...
 - ▶ bloc : liste de déclarations de variables, liste de définitions de procédures, bloc.

Exemple : procédure accédant à une variable globale

```
1 var x: integer;  
2 proc p(y: integer)  
3   var z: integer;  
4   begin  
5     z := 2*y;  
6     x := z;  
7   end  
8 begin  
9   x := 5;  
10  p(x);  
11  print x;  
12 end
```

Exemple : procédure

```
1 var x: integer;  
2 proc p(y: integer)  
3   var z: integer;  
4   begin  
5     z := 2*y;  
6     print z;  
7   end  
8 begin  
9   x := 5;  
10  p(x);  
11 end
```

Exemple : liaison statique ou dynamique

```
1 var x: integer;  
2 proc p(y: integer)  
3   begin  
4     x := x+y;  
5   end  
6 begin  
7   x := 1;  
8   var x: integer;  
9   begin  
10    x := 2;  
11    p(1);  
12    print x;  
13  end  
14 end
```

Liaison statique ou dynamique

- ▶ Liaison en question : la connexion entre l'utilisation d'un nom (d'une variable, par ex.), et la déclaration du même nom.
- ▶ Problème : on peut avoir plusieurs déclarations du même nom, laquelle choisir ?
- ▶ Liaison dynamique : la liaison est faite suivant la priorité à la déclaration la plus locale *au moment de l'exécution*.
- ▶ Liaison statique : la liaison est faite suivant la priorité à la déclaration la plus locale *au moment de la définition de la procédure*.

Tous les langages de programmation modernes suivent cette politique.