# Fondaments de la programmation
# Exercice série 1: Sémantique Denotationelle

April 27, 2000

## 1 SimplE

Extend the abstract syntax and the denotational semantics of the language SimplE adding the following constructs:

**for** *v* := *E1* **to** *E2* **do** *Body*.

### 1.1 Solution

The instruction "**for** *v* := *E1* **to** *E2* **do** *Body*" can be considered equivalent to:

*v*:=*E1*; **while** *v* ≤*E2* **do** (*Body*;*v* := *v* +*1*)

Its denotational semantics can be derived by the semantics of the above instruction and simplifying as much as possible:

```
σ_Instr  i  =
    case i of
        ...
        (for v = E1 to E2 do Body) : λs. lim_{i→∞} s_i;
    end
```

where

$s_0 = \lambda s.s_v^{\sigma_{Expr} E1\, s}$

$s_{i+1} = cond(\sigma_{Expr}(v \leq E2)\, s_i, (\sigma_{Instr} v := v + 1\, (\sigma_{Instr} Body\; s_i)), s_i)$

$= cond(\sigma_{Expr}(v \leq E2)\, s_i, (\sigma_{Instr} Body\; s_i)_v^{\sigma_{Expr} v+1\, (\sigma_{Instr} Body\; s_i)}, s_i)$

$= cond(\sigma_{Expr}(v \leq E2)\, s_i, (\sigma_{Instr} Body\; s_i)_v^{\sigma_{Expr} v\; s_i + 1}, s_i)$

$= cond(s_i(v) \leq \sigma_{Expr} E2\, s_i, (\sigma_{Instr} Body\; s_i)_v^{s_i(v)+1}, s_i).$

**Note:** No constraints have been imposed on the semantics like for instance avoiding the modification of the variable $v$ in the body of the **for** instruction. Usual definitions (e.g. Ada) force expressions E1 and E2 to be evaluated in states, i.e. prior to the iteration steps.

# 2 SimplB

## 2.1 Functions

Extend the abstract syntax and the semantics of LaProSEB with the definition of functions from natural to natural using parameter passing by value. Give versions for static binding and dynamic binding.

### 2.1.1 Solution

**Abstract syntax:**

Expr ::= V(Expr) | *...in addition to usual SimplB expressions*

Decl ::= **def** V = **lambda**(V); Instr; **return** Expr | *...in addition to usual SimplB declarations*

**Semantic domains:**

F = E → S → E

**Semantic functions** (two options):

1. Static Binding

$\sigma_{Decl}$ *i u* =
    case *i u* of
      ...
      (def *f* = lambda(*v*);*Body*; return *e*) u: $\lambda s.u_f^y$
            where $y = \lambda x.\sigma_{Expr} e\,(\sigma_{Instr} instr\; u_v^{(\sigma_{Expr} x\, u\, s)}\; s)$
      ....
    end

$\sigma_{Expr}$ *x u* =
    case *x u* of
      ...
      *f(e) u*: $\lambda s.u(f)\,e\,s$
      ...
    end

2. Dynamic Binding

$\sigma_{Decl}$ *i u* =
    case *i u* of
      ...
      (def *f* = lambda(*v*);*instr*; return *e*) u: $\lambda s.u_f^y$
            where $y = \lambda w.\lambda x.\sigma_{Expr} e\,(\sigma_{Instr} instr\; u_v^{(\sigma_{Expr} x\, w\, s)}\; s)$
      ....
    end

$\sigma_{Expr}$ *x u* =
    case *x u* of
      ...
      *f(e) u*: $\lambda s.u(f)\,u\,e\,s$
      ...
    end

## 2.2 Lists

Extend the abstract syntax and the semantics of SimplEB introducing the notion of List of natural numbers. Add the usual list operations *head*, *tail* et *concat*.

### Syntax

List = nil | cons(Nb,List)

Expr = List | **head** List | **tail** List | **concat** List List | *...in addition to N and B type expressions in SimplB*

### Semantics

- Semantic Domains

  The List domain $L$ satisfies the following equation:

  $L = \{nil\} + (N \times L)$ where '+' denotes the *disjoint union* of sets.

  The *cons:* $N \times L \to L$ is a list constructor and it verifies the following properties:

  $\forall x \in N : cons(x,nil) \in L$
  $\forall x \in N : \forall y \in L : cons(x,y) \in L$

- List Operators

  $head(nil) = \perp$
  $\forall x \in N : \forall r \in L : head(cons(x,r)) = x$

  $tail(nil) = \perp$
  $\forall x \in N : \forall r \in L : tail(cons(x,r)) = r$

  $\forall l \in L : concat(nil,l) = l$
  $\forall l,r \in L : \forall x \in N : concat(cons(x,r),l) = cons(x, concat(r,l))$

- Semantic Function for Exp

  **case** $e$ **of**
  ...
  **head** $l$ : $\lambda s.head(\sigma_{Expr} l\, s)$
  **tail** $l$ : $\lambda s.tail(\sigma_{Expr} l\, s)$
  **concat** $l1$ $l2$ : $\lambda s.concat(\sigma_{Expr} l1\, s, \sigma_{Expr} l2\, s)$
  **cons(**$x$**,**$r$**)** : $\lambda s.cons(\sigma_{Expr} x\, s, \sigma_{Expr} r\, s)$
  **nil**: $\lambda s.nil$
  **end**

**Open problem:** Consider $L = \{nil\} + (N \times L)$ a set-theoretical recursive equation. How can one construct solution $L$?