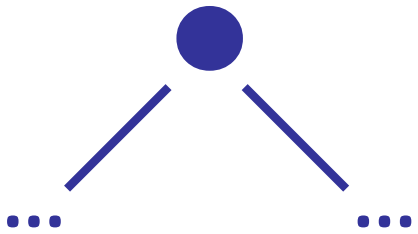
Projet de compilation

spécification du langage WHILE

Olivier Ridoux

Domaine de calcul (1)

- **Arbres binaires = binTree**

1. 

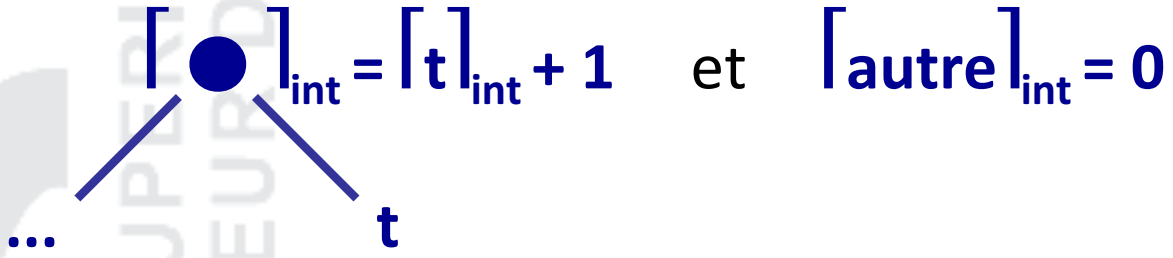
2. $\otimes$
3. **symb (≈ chaîne)**

Domaine de calcul (2)

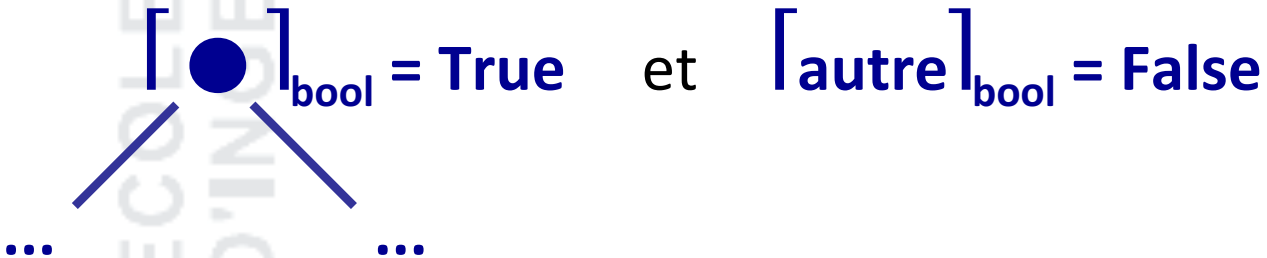
- Un seul type = binTree
- **Pas de int, bool, string, ...**

Domaine de calcul (3)

- Simuler les autres types
  - int



  - bool



  - ...

Domaine de calcul (4)

- Simuler les autres types
  - string

$$\lceil symb \rceil_{string} = symb.str$$

$$\lceil \otimes \rceil_{string} = \varepsilon$$

$$\lceil \bullet \rceil_{string} = \lceil s \rceil_{string} \cdot \lceil t \rceil_{string}$$

Expressions (0)

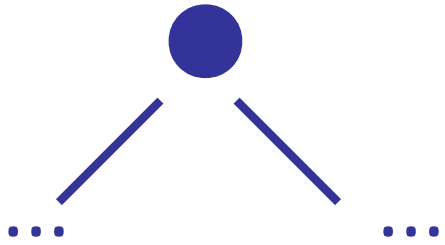
- Le rôle fondamental des expressions

est de

dénoter des arbres binaires

Expressions (1)

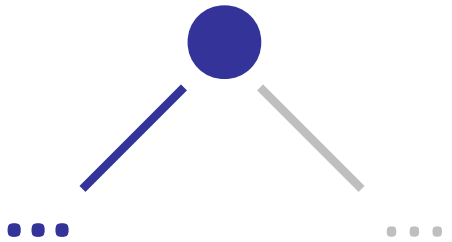
- **(cons ... ...)** ⟶
- **nil** ⟶ ⊗
- **Var** ⟶ **valeur de** Var = binTree
- **symb** ⟶ **symbole de** symb.str



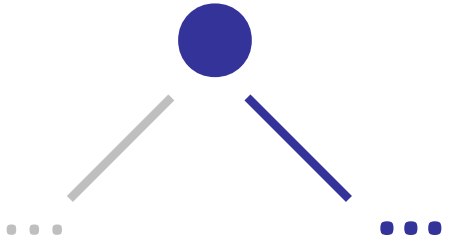
Expressions (1)

- **(hd t)** ⟶

  si t ⟶

  alors (hd t) ⟶ 

  sinon (hd t) ⟶ ⊗

- Pareil pour **(tl t)** 

Pas d'erreurs à l'exécution (1)

- Gérer proprement les erreurs à l'exécution est **complexe**...

...ne pas les gérer proprement n'est **pas intéressant**, voire **irresponsable**

→ s'arranger pour qu'il n'y ait **pas d'erreur** à l'exécution

Pas d'erreurs à l'exécution (2)

- Valeur par défaut = ⊗ pour...

...les variables **pas initialisées**

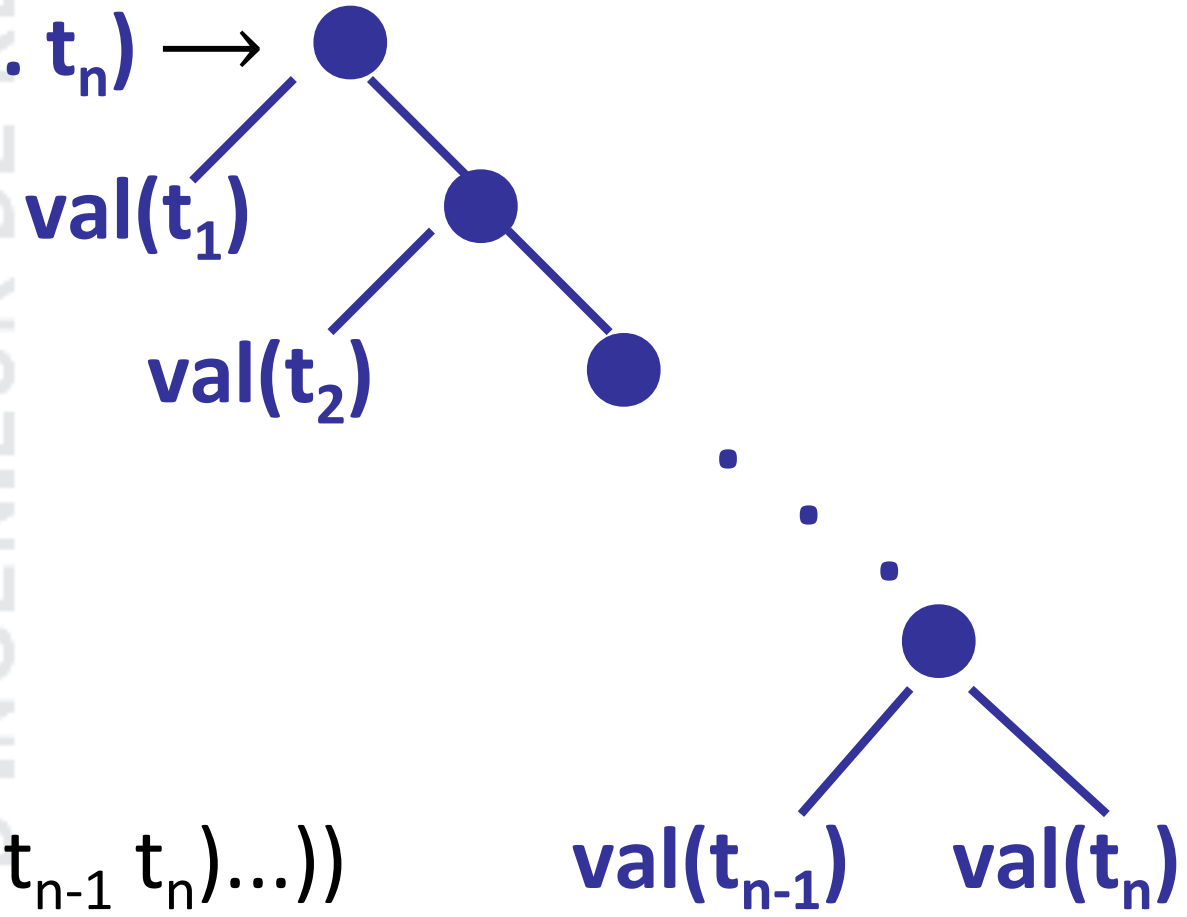
...les opérations **pas définies**

(hd ⊗) = (tl ⊗) = ⊗

(hd toto) = (tl toto) = ⊗

Expressions (2)

- **(cons t_1 t_2 ... t_n) ⟶**

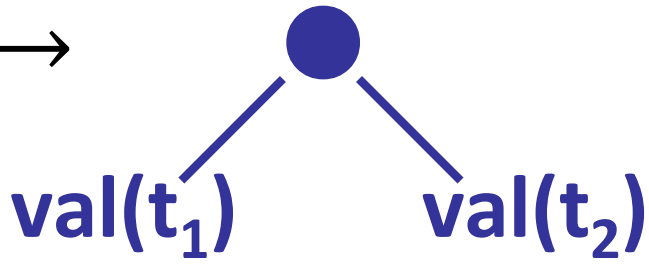


- =

  (cons t_1
  (cons t_2
  ... (cons t_{n-1} t_n)...))

Expressions (3)

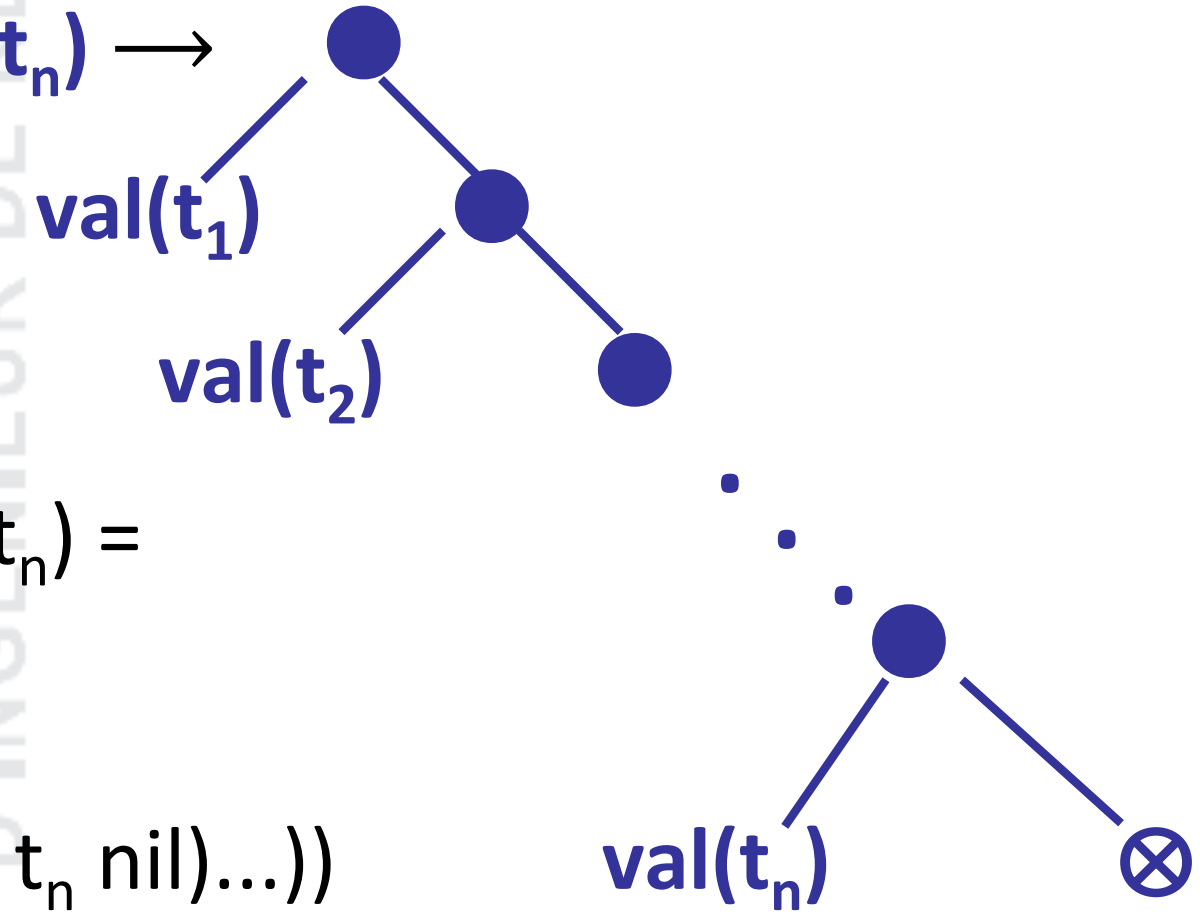
- $(\text{cons } t_1\ t_2) \longrightarrow$



- $(\text{cons } t_1) \longrightarrow \text{val}(t_1)$
- $(\text{cons}) \longrightarrow \otimes$

Expressions (3)

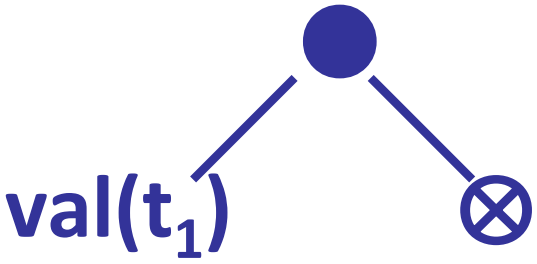
- **(list t_1 t_2 ... t_n) ⟶**



- (list t_1 t_2 ... t_n) =
  (cons t_1
    (cons t_2
      ... (cons t_n nil)...))

Expressions (4)

- **(list t_1)** $\longrightarrow$



- **(list)** $\longrightarrow$ $\otimes$

Expressions (7)

- **$(f\ t_1\ t_2 \ldots t_n) \longrightarrow$**

  si **f** est une fonction **définie**

  function **f**: ...

  si **f a le bon nombre** de paramètres

  read X_1, X_2, ..., X_N

  si **f a le bon nombre** de résultats

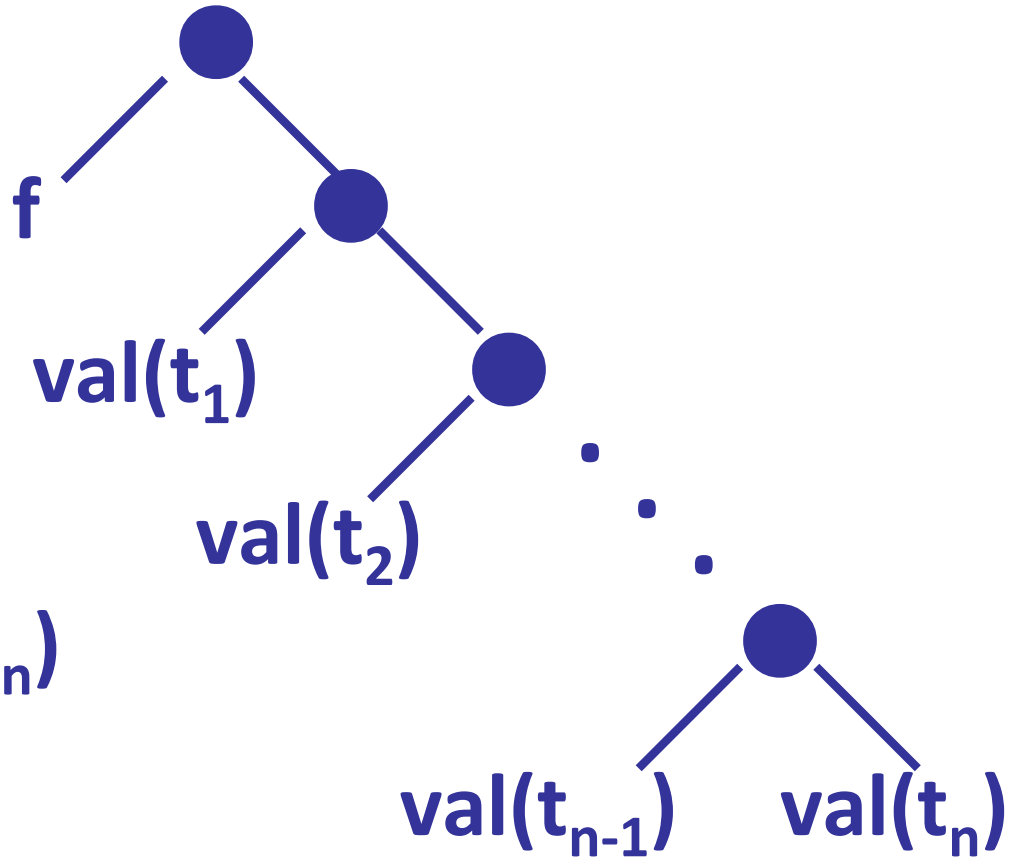
  write **Y**

  alors **valeur de Y**

- **Vérification de type par le compilateur**

Expressions (8)

- $(f\ t_1\ t_2\ \ldots\ t_n) \longrightarrow$



$= (cons\ f\ t_1\ t_2\ \ldots\ t_n)$

si **f** n'est pas une fonction définie

La mémoire

- La mémoire du processus d'exécution stocke :
  1. des **arbres binaires**
  2. des **relations entre variables et arbres binaires**

- **Il n'y a pas de variable dans les arbres binaires**
- **On ne peut pas modifier un arbre binaire**

Variables (1)

- **V**

...s'évalue dans la mémoire courante

...valeur par défaut = $\otimes$

- Mémoire : **variable ⟶ binTree**

Variables (2)

- Les variables sont locales à la fonction où elles apparaissent

- **Pas de variable globale !**

...les programmes sont purement fonctionnels

Variables (3)

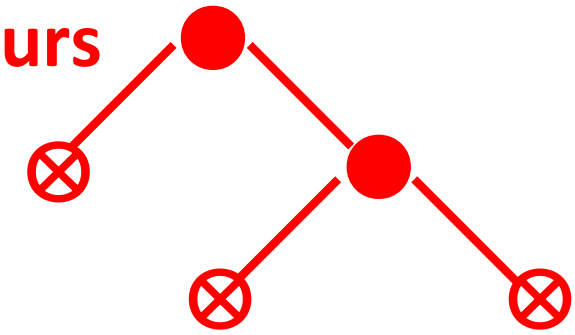
- **Pas d'effet de bord !**

  X := (cons nil nil) ;

  Y := (cons nil X) ;

  X := nil

...Y vaut toujours

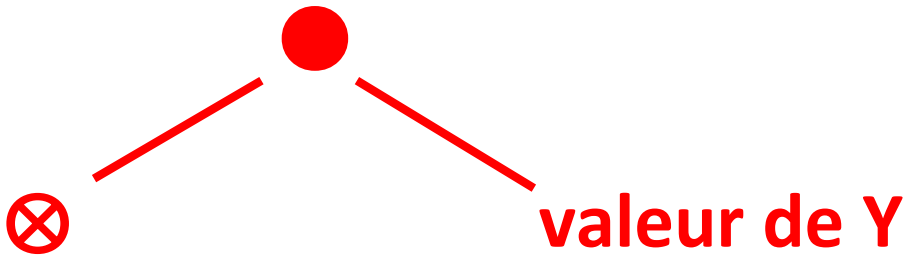


...le seul moyen de modifier Y est Y := ...

Variables (1.4)

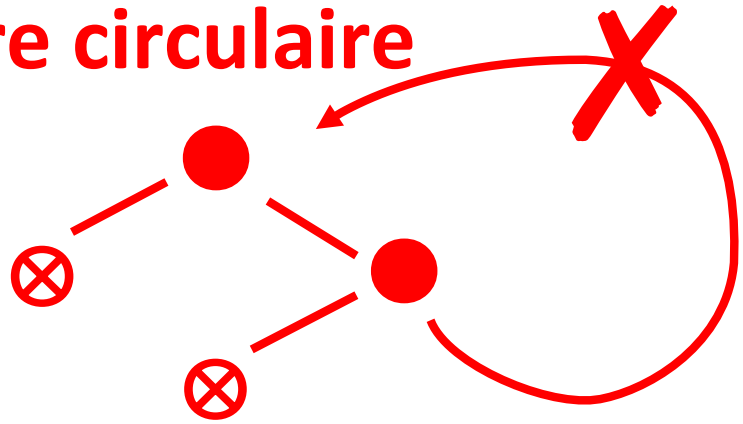
- **Pas de variable dans les termes**

X := (cons nil Y)

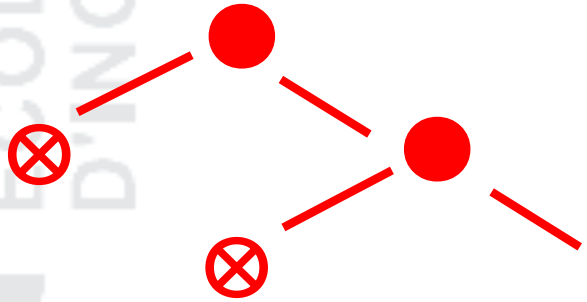


Variables (4)

- **Pas de structure circulaire**



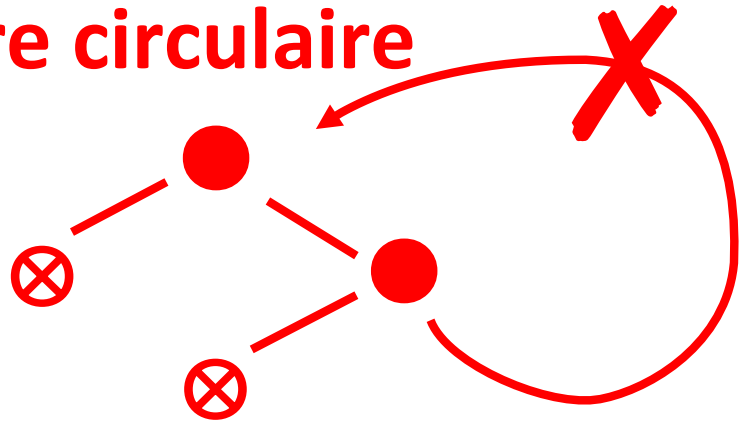
X := (cons nil (cons nil Y)) ; Y := X



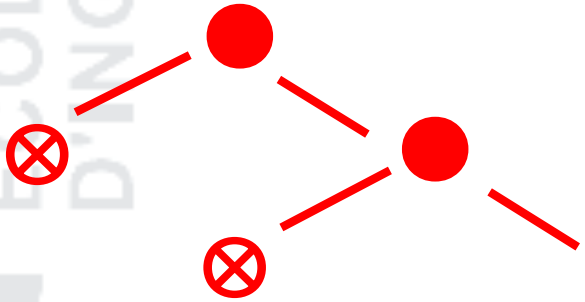
vieille valeur de Y

Variables (5)

- **Pas de structure circulaire**



X := (cons nil (cons nil X))



vieille valeur de X

Commandes (0)

- Le rôle **fondamental** des commandes

est de

modifier la relation
entre variables et arbres binaires

- **On ne peut rien modifier d'autre**

Commandes (1)

- **nop**

  ne fait rien

- Utile pour le principe, et pour le test

Commandes (2)

- **C_1 ; C_2**

exécute C_1 puis C_2

- L'exécution de C_1 modifie l'état de la mémoire...

...**puis** C_2 est exécuté dans le nouvel état,
et modifie à nouveau l'état de la mémoire

Commandes (3)

- **if E then C_1 else C_2 fi**

  si $[\![E]\!]_{bool}$ alors exécuter C_1,
  sinon exécuter C_2

- **if E then C_1 fi**

  si $[\![E]\!]_{bool}$ alors exécuter C_1,
  sinon ne rien faire

...conditionnelle standard où ● joue le rôle de **Vrai**

Commandes (4)

- **while E do C od**

  si $[\![E]\!]_{bool}$ alors exécuter C et recommencer

  sinon ne rien faire

...boucle while standard où ● joue le rôle de **Vrai**

...C est exécuté 0, 1, ..., ou n fois

C*

- **while E do C od peut boucler indéfiniment !**

Commandes (5.1)

- **for E do C od**

  exécuter $\lceil E \rceil_{int}$ fois la commande **C**

  $C^{\lceil E \rceil_{int}}$

- soit v la valeur de E

  Ⓐ si v != ● ne rien faire sinon exécuter C

  et recommencer en Ⓐ avec v = (tl v)

- for E do C od **ne peut pas boucler indéfiniment**

Commandes (5.2)

- L'exécution de C **ne doit pas perturber** le décompte du for

for X do X := (cons nil X) od

...double la longueur de X

X ⟶ (cons nil (cons nil ... X...))

(X fois)

Commandes (5.3)

for E do C od

≠

while E do C ; E := (tl E) od

Commandes (6)

- **foreach X in E do C od**

  exécuter $\lceil E \rceil_{int}$ fois la commande **C**

  $C^{\lceil E \rceil_{int}}$

  en remplaçant **X** par les éléments de **E**

- soit v la valeur de E

  Ⓐ si v != ● ne rien faire

  sinon exécuter C[X⟵(hd v)] et

  recommencer en Ⓐ avec v = (tl v)

Commandes (7.1)

- **$V_1, \ldots, V_n := E_1, \ldots, E_n$**

  évaluer **tous les E_i**, (donne **val(E_i)**)

  puis affecter les **val(E_i)** aux **V_i** correspondants

X, Y := Y, X

...permute les valeurs de X et Y

Commandes (7.2)

$$V_1, \ldots, V_n := E_1, \ldots, E_n$$

$$\neq$$

$$V_1 := E_1 ; V_2 := E_2 ; \ldots ; V_n := E_n$$

$$=$$

$$R_1 := E_1 ; R_2 := E_2 ; \ldots ; R_n := E_n$$
$$V_1 := R_1 ; V_2 := R_2 ; \ldots ; V_n := R_n$$

Commandes (7.3)

- **$V_1, \ldots, V_m := (f\ E_1 \ldots E_n)$**

...évalue l'appel,

puis affecte les résultats aux V_i correspondants

Quotient, Reste := (div X Y)

- **Vérification de type par le compilateur**

function f: ... read $X_1, \ldots, X_n$ % ... % write $Y_1, \ldots, Y_m$

Lancement (1)

- Soit le programme f.wh

  function p: read X % ... % write Y

  function main: read A % ... (p E) ... % write B

  – **main** appelle **p ;** l'affectation du **paramètre effectif E** au **paramètre formel X** est faite par la séquence d'appel de procédure

- **Qui appelle main ? D'où vient le paramètre effectif de main ? Qui l'affecte à A ?**

Lancement (2)

- Soit la ligne de commande

  [unix] % f "(cons nil (cons nil nil))"

  - le lancement du programme **f** appelle **main**
  - le lancement du programme **f** affecte au **paramètre formel de main** le **paramètre effectif** dénoté dans la ligne de commande
  - analyse syntaxique de **(cons nil (cons nil nil))** !

Lancement (2)

- Soit la ligne de commande

[unix] % f 2

  - le lancement du programme **f** appelle **main**
  - le lancement du programme **f** affecte au **paramètre formel de main** le **paramètre effectif** dénoté dans la ligne de commande
  - Traduction de **2** en **(cons nil (cons nil nil))** !

- **Conversion du monde shell au monde WHILE**

Retour (1)

- Soit le programme f.wh

  function p: read X % ... % write Y

  function main: read A % ... (p E) ... % write B

  - **main** appelle **p ;** la captation du **paramètre de retour Y** est faite par la séquence de retour de procédure

- **Qui capte le résultat de main ?**
  Où va la valeur de B ?

Retour (2)

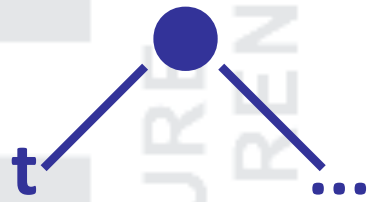
- Soit la ligne de commande

[unix] % f ...

  - le lancement du programme **f** appelle **main** qui appelle **p**
  - le résultat de **main** est affiché sur la console de l'environnement d'appel
  - *pretty-printing* de la valeur de **B** !

- **Conversion du monde WHILE au monde shell**

Retour (3)

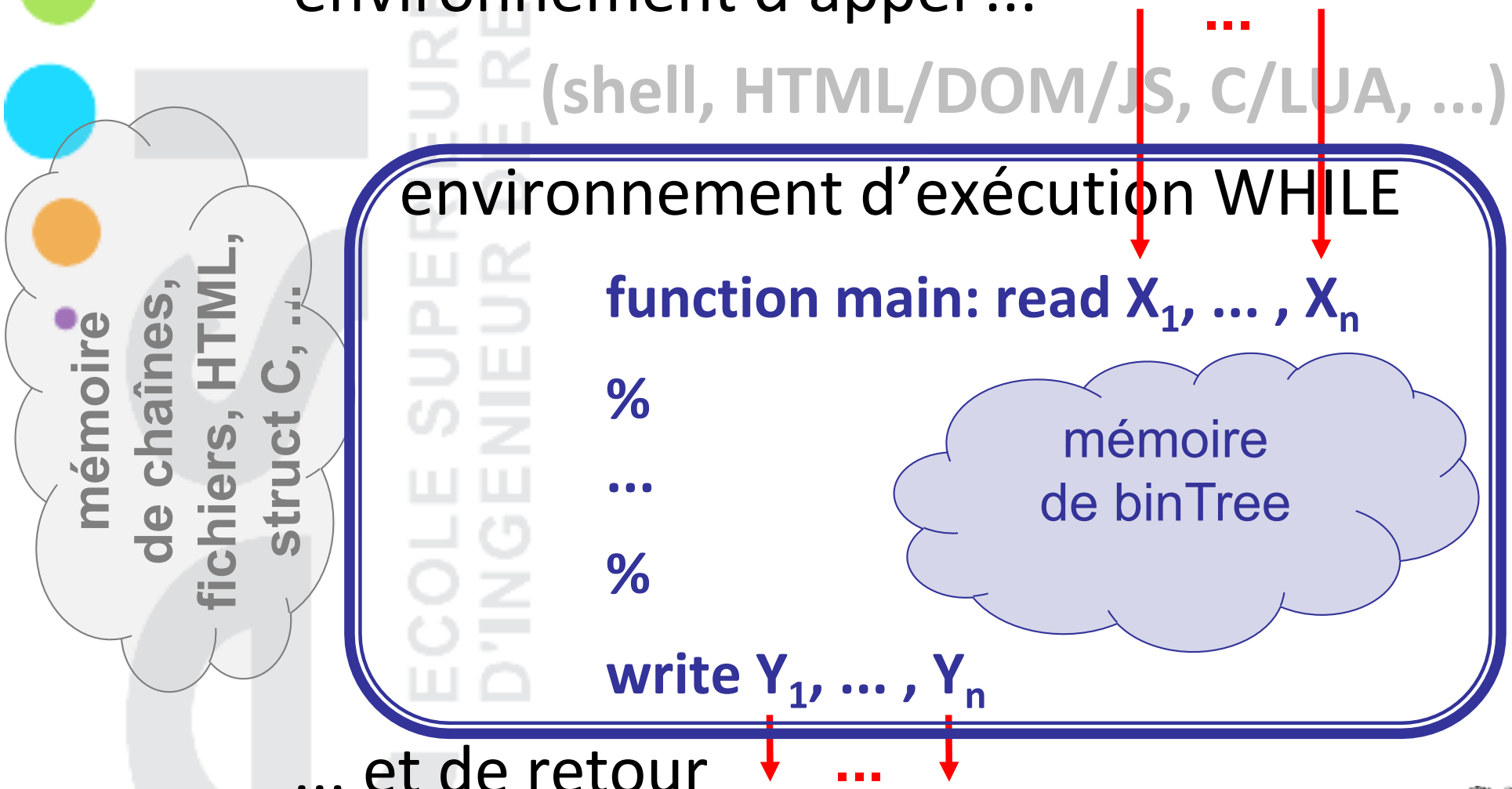
- Conventions de *pretty-printing*
  - 

    ⟶ **(cons** pp(**t**) pp*(**...**)**)**
  - ⊗ ⟶ **nil**
  - **Var** ⟶ pp(**valeur de** Var)
  - **symb** ⟶ **représentation de** symb
  - **(int ...)** ⟶ $\lceil \ldots \rceil_{int}$
  - **(bool ...)** ⟶ $\lceil \ldots \rceil_{bool}$
  - **(string ...)** ⟶ $\lceil \ldots \rceil_{string}$
  - **sp** ⟶ " "

(int (cons nil (cons nil nil))) ⟶ 2 !

En résumé

environnement d'appel ...



En détail

p.wh

```
function f: ...

function g: ...

...

function main: ...
```

p.x

```
f12(...) {...}    // f

f53(...) {...}    // g

...

f0(...) {...}     // main
```

libwh.x

```
main(argc, argv) {
   setupWH();
   in = inWH(argc, argv);
   out = f0(in);
   outWH(out)
}
...
```

spécification