Premiers principes des langages de programmation

http://www.enseignement.polytechnique.fr/informatique/INF32

Les 4 concepts de l'informatique

Algorithme

Langage

Information

Machine

Parmi les langages ...

Les langages de programmation

Au confluent de ces quatre notions (la clé de voûte de l’informatique)

Apprendre un langage : devenir autonome

Mais ... un savoir réputé difficile à enseigner

Une grande diversité : des querelles de chapelles ...

Une grande volatilité : à quoi bon apprendre à conduire une 207 ?

Des détails insignifiants : écrire `:=` et non `=`, terminer par `;`

Confusion avec un savoir faire : écrire un programme

Difficulté à conceptualiser : la machine, elle fait ceci, elle fait cela

Jusqu'à quel degré de détail descendre ? (finalement : propagation d'un champ électrique dans un système horriblement complexe)

Cependant ...

Les différents langages sont organisés autour d'un petit nombre de fonctionnalités

Présentes dans de nombreux langages

Qui sont relativement stables

Et que l'on peut décrire simplement avec les outils adéquats

affectation, séquence, test, boucle, fonction, récursivité, enregistrement, cellule, module, objet, ...

I. Ce que l’on peut expliquer aux élèves

Dans de nombreux langages : quatre instructions

L’affectation, la séquence, le test, la boucle (+ la déclaration)

Le noyau impératif de ce langage

Pour chacune d’entre elles, décrire

– sa syntaxe : la manière dont cette instruction s’écrit
– sa sémantique : ce qui se passe quand on l’exécute

Décrire la syntaxe

La question importante : de quoi est constituée cette instruction

La question accidentelle : comment cela s'écrit-il dans le langage particulier que l'on utilise

L'affectation : constituée d'une variable `x` et d'une expression `t`

(`x = t`, `x := t`, `x = t;`, ...)

La séquence : constituée de deux instructions `p` et `q`

(`p q`, `p;q`, ...)

Le test : constitué d'une expression `t` et de deux instr. `p` et `q`

(`if (t) p else q`, `if t then p else q`, ...)

La boucle : constituée d'une expression `t` et d'une instruction `p`

(`while (t) p`, `while t do p`, ...)

Les instructions et les expressions

`x + 3` : une expression

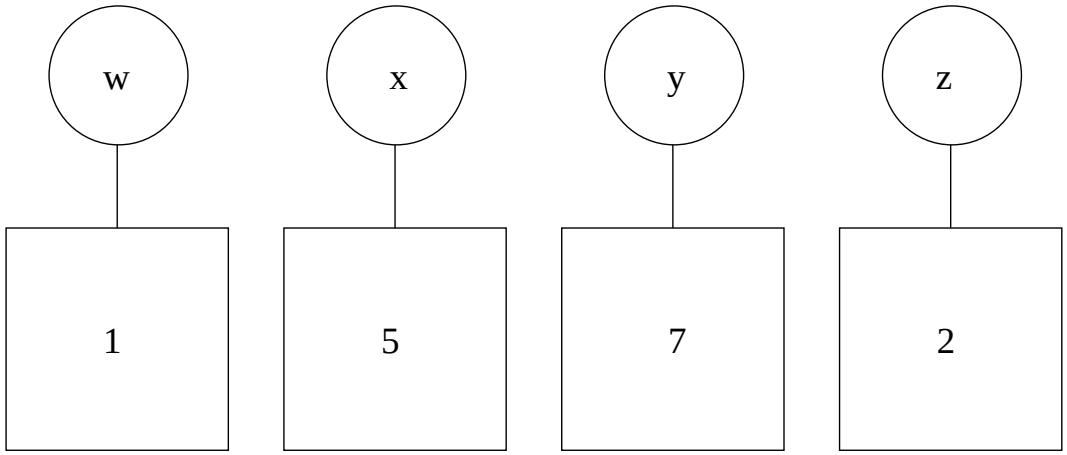
`y = x + 3` : une instruction (en l'occurrence une affectation)

Décrire la sémantique

Ce que fait un programme : transformer un état

Un état indique la valeur de chaque variable à un instant donné

(une fonction d'un ens. fini de variables dans l'ens. des valeurs)



valeur $\neq$ expression (3 valeur, mais pas x + 3)

La valeur d’une expression dépend de l’état

$\Theta(x + 3, [x = 5]) = 8$

Une question difficile

Est-ce que c'est réellement comme cela à l'intérieur de la machine ?

Plus ou moins, mais de moins en moins

On ne parle pas de la machine mais du langage

Une instruction transforme un état en un autre état

$\Sigma(p,s) = s'$

Si on exécute l'instruction p dans l'état s, cela produit l'état s'

L'affectation

Une opération sur les états : `s + (x = v)`

même état que `s` sauf pour la case `x` qui prend la valeur `v`

Exemple :

`s = [x = 4, y = 5, z = 6]`

`s + (y = 7) = [x = 4, y = 7, z = 6]`

$\Sigma(x = t;,s) = s + (x = \Theta(t,s))$

La séquence

$$\Sigma(\{p1\ p2\},s) = \Sigma(p2,\Sigma(p1,s))$$

Le test

si Θ(t,s) = true

Σ(if (t) p1 else p2,s) = Σ(p1,s)

si Θ(t,s) = false

Σ(if (t) p1 else p2,s) = Σ(p2,s)

Avant de voir la boucle ...

Les points importants :

1. `x` rien à voir avec une variable mathématique

2. La valeur de `x + 3` ne change pas miraculeusement au cours du temps

Une expression n'a une valeur que dans un certain état

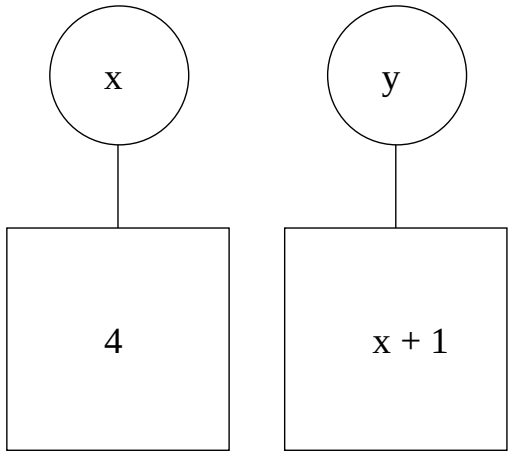
Θ`(t,s)`

3. Dans une affectation, on évalue l'expression dans l'état courant puis on oublie l'expression

```
x = 4; y = x + 1; x = 10;
```

`y` prend pas la valeur `5` et non `11`

Pas absurde mais une autre sémantique



La boucle

```
while (t) q
```

```
while (x < 1000) x = x * 2;
```

Quand on exécute cette instruction : on calcule la valeur de `t`, si c'est `false` on a terminé, si c'est `true`, on exécute `q`, puis on

La boucle

```
while (t) q
```

```
while (x < 1000) x = x * 2;
```

Quand on exécute cette instruction : on calcule la valeur de `t`, si c'est `false` on a terminé, si c'est `true`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `false` on a terminé, si c'est `true`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `false` on a terminé, si c'est `true`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `false` on a terminé, si c'est `true`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `false` on a terminé, si c'est `true`, on exécute `q`, puis on calcule la valeur de `t`, si c'est `false` on a terminé, si c'est

`while (t) q` : notation finie pour une instruction infinie

```
if (t) {q if (t) {q if (t) ...
                                else skip;}
                        else skip;}
          else skip;
```

avec une instruction fictive `skip;` qui ne fait rien

(Σ`(skip;,s) = s`)

```
{int x = 3; while (x <= 1000) x = 2;}
```

ne termine pas

Instruction infinie : potentialité de non terminaison

```
{int x = 3; while (x <= 1000) x = 2;}
```

ne termine pas

Instruction infinie : potentialité de non terminaison

Comment modéliser la non terminaison

Si on exécute l'inst. `p` dans l'état `s`, cela ne produit aucun état

Σ fonction partielle, non définie en `(p,s)`

Approximations finies :

n tours de boucle

si on n'a pas fini on abandonne

```
if (t) {q if (t) {q if (t) giveup;
                          else skip;}
                 else skip;}
        else skip;
```

Σ jamais définie en `(giveup;,s)`

p_0 = if (t) giveup; else skip;

p_{n+1} = if (t) {q p_n} else skip;

Suite $\Sigma(p_n, s)$ jamais définie ou définie à partir d'un certain rang et constante sur son domaine : limite

Σ(while (t) q,s) = $\lim_n \Sigma(p_n, s)$

Un exemple

```
while (x < 1000) x = x*2;
```

p_0 = `if (x < 1000) giveup; else skip;`

p_{n+1} = `if (x < 1000) {x = x * 2; p`$_n$`} else skip;`

Σ(p0,[x=300]) pas définie

Σ(p1,[x=300]) = Σ(p0,[x=600]) pas définie

Σ(p2,[x=300]) = Σ(p1,[x=600]) =

Σ(p0,[x=1200]) = [x=1200]

...

Σ(while (x < 1000) x = x*2;, [x=300]) = [x=1200]

Un exemple

```
while (x < 1000) x = x * 2;
```

demande d'exécuter p dans [x = 300]

demande d'exécuter p dans [x = 600]

demande d'exécuter p dans [x = 1200]

donne [x = 1200]

Les messages essentiels

Un programme fait quelque chose à quelque chose : un état

On peut expliquer pourquoi les programmes font ce qu'ils font

Ils pourraient faire autre chose

(exemple : `x = 4; y = x + 1; x = 10;`)

II. Les états et les transitions

Décrire un processus comme une suite de transition entre états

Une partie d'échec : suite d'états de l'échiquier

La transition d'un état à l'autre décrite par les règles du jeu

Ne rien oublier dans l'état (sinon attention aux hystérésis)

Comment décrire un aéroport ?

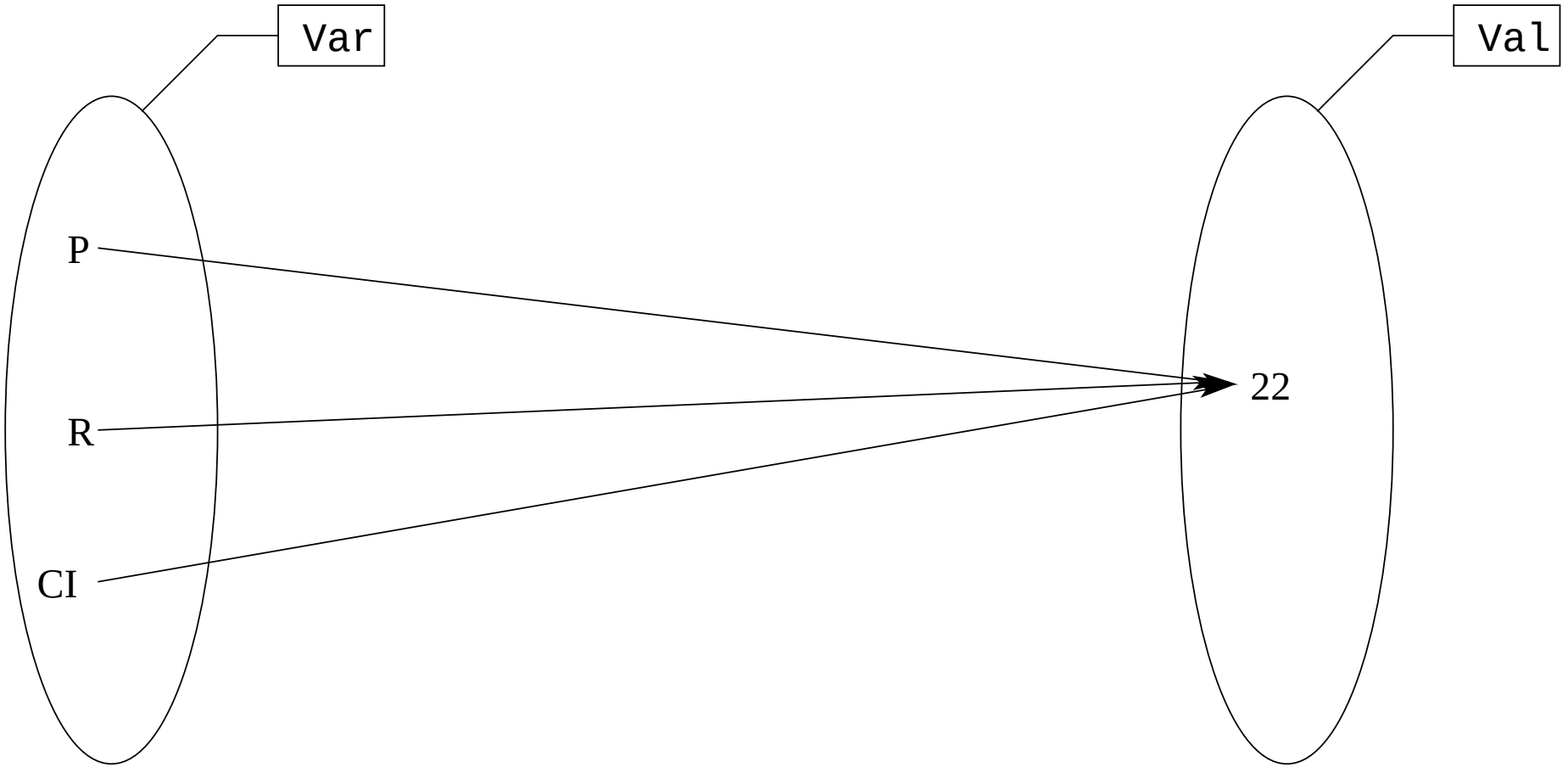
La notion d'automate

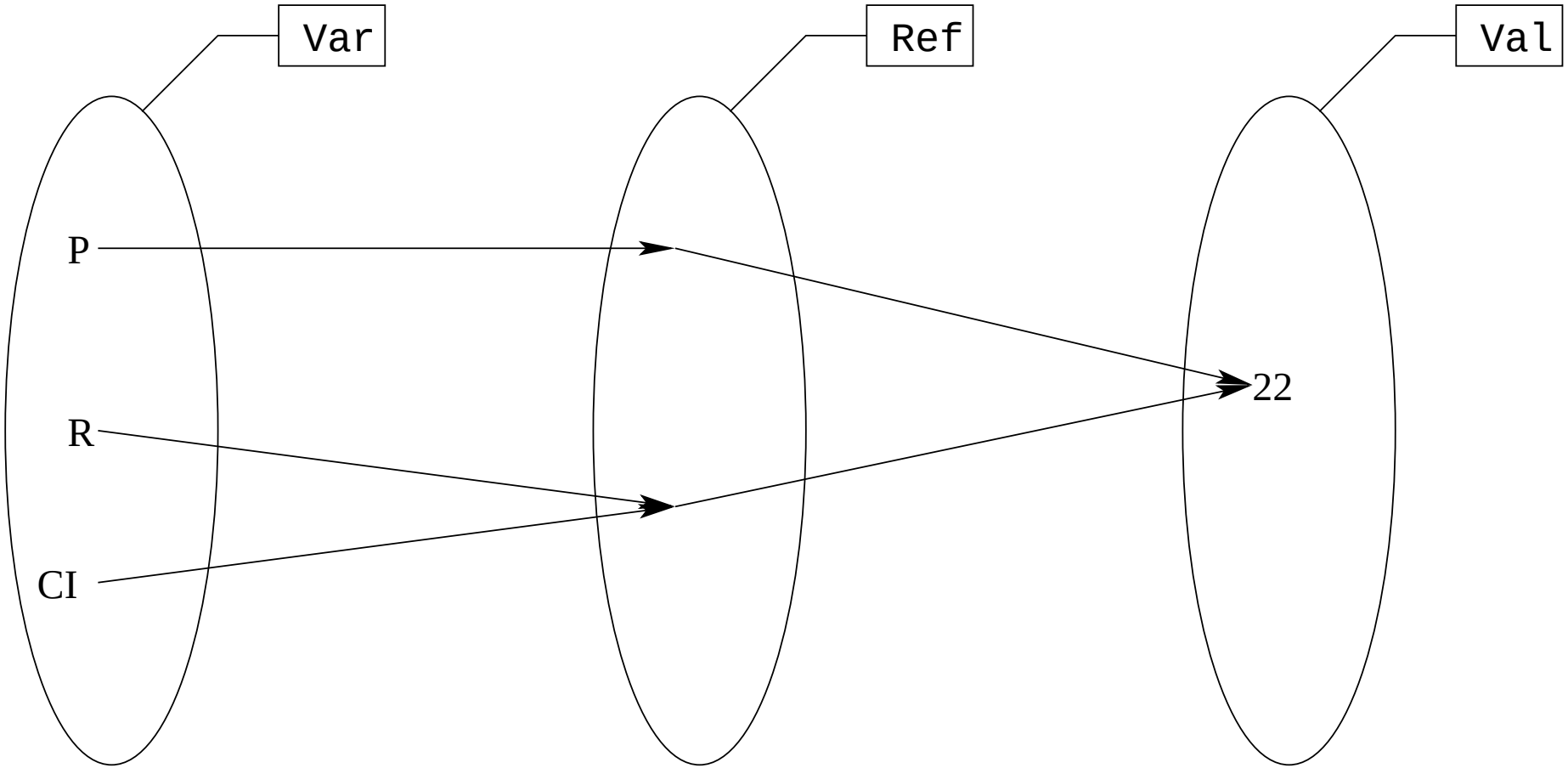
III. Le mot et la chose

Quelle est la température à Paris : $22°C$

Quelle est la température à Rome : $22°C$

Quelle est la température dans la capitale de l'Italie : $22°C$


Var
Val
P
R
CI
22


Var
Ref
Val
P
R
CI
22

Le partage

Devient essentiel quand on ajoute d’autres fonctionnalités (fonctions, enregistrements), ...

Également essentiel pour comprendre le web (copier v.s. mettre un lien)