Interface PM
IFT2905
Travaux pratiques

Franz Girardin

22 mai 2024

Table des matières

Section 1

TP1 Notation

> **Exercice 1**
>
> Le but de cet exercice est de comprendre comment passer d'une notation in- fixe (lisible par les humains) à une notation postfixe et préfixe. Ces notations sont utiles car elles permettent à un ordinateur de les ´evaluer (calculer) plus facilement et sans ambiguité.

1.1

$$\begin{aligned} a+b+c = ((a+b)+c) &= ((+a\,b)+c) \\ &= +(+a\,b)\;c \\ &= ++a\,b\,c \end{aligned}$$

préfixé

$$\begin{aligned} a+b+c = ((a+b)+c) &= ((a\,b+)+c) \\ &= ((a\,b+)\;c+) \\ &= a\,b\;+c+ \end{aligned}$$

postfixé

1.2

$$\begin{aligned} a+(b+c) = (a+(+b\,c)) &= (+a\,(+b\,c)) \\ &= +\,a+b\,c \end{aligned}$$

préfixé

$$\begin{aligned} a+(b+c) = (a+(b\,c+)) &= (a\,(b\,c)\;++) \\ &= a\,b\,c\;++ \end{aligned}$$

postfixé

1.2

$$\begin{aligned} a\times b+c\times d &= (a\times b)+(c\times d) \\ &= (\times a\,b)+(\times c\,d) \\ &= +(\times a\,b)\times c\,d \\ &= +\times a\,b\times c\,d \end{aligned}$$

préfixé

$$\begin{aligned} a\times b+c\times d &= (a\times b)+(c\times d) \\ &= (a\,b\times)+(c\,d\times) \\ &= (a\,b\times)(c\,d\times)+ \\ &= a\,b\times c\,d\times+ \end{aligned}$$

postfixé

$$\begin{aligned} a+b<a\times(c+d) &= (+a\,b)<\big(a\times(+c\,d)\big) \\ &= (+a\,b)<\big(\times a(+c\,d)\big) \\ &=<(+a\,b)\big(\times a(+c\,d)\big) \\ &=<+a\,b\times a+c\,d \end{aligned}$$

préfixé

$$\begin{aligned} a+b<a\times(c+d) &= (a\,b+)<\big(a\times(c\,d+)\big) \\ &= (a\,b+)<\big(a(c\,d+)\times\big) \\ &= (a\,b+)\big(a(c\,d+)\times<\big) \\ &= a\,b+ac\,d+\times<\big) \end{aligned}$$

postfixé

$$\begin{aligned} &(-b+sqrt(b\times b-4\times a\times c))/(2\times a) \\ &= (-b+sqrt((\times b\,b)-(\times\times 4\,a\,c)))/(\times 2\,a) \\ &= \big(+-b\;sqrt\big(-\times b\,b\times\times 4\,a\,c\big)\big)/(\times 2\,a) \\ &= \big(+-b\;sqrt\big(-\times b\,b\times\times 4\,a\,c\big)\big)\times 2\,a/ \\ &= +-b\;sqrt-\times b\,b\times\times 4\,a\,c-/\times 2\,a \end{aligned}$$

préfixé

$$\begin{aligned} &(-b+sqrt(b\times b-4\times a\times c))/(2\times a) \\ &= (b-+sqrt((b\,b\times)-(4\,a\times c\times)))/(2\,a\times) \\ &= (b-+sqrt(b\,b\times 4\,a\times c\times-))/(2\,a\times) \\ &= (b-+b\,b\times 4\,a\times c\times-sqrt)/(2\,a\times) \\ &= (b-b\,b\times 4\,a\times c\times-sqrt+)+/(2\,a\times) \\ &= b-b\,b\times 4\,a\times c\times-sqt+2\,a\times \end{aligned}$$

postfixé

> **Exercice 2**
>
> Le but de cet exercice est de comprendre comment un ordinateur réesout un calcul à partir d'une notation postfixe.

$ab+c+$:

`stack`(a);
`stack`(b);
do :
  `// Opération a + b`
  `stack`$\big($`unstack`$()+$`unstack`$()\big)$;
`stack`(c)
do :
  `// Opération (a + b) + c`
  `stack`$\big($`unstack`$()+$`unstack`$()\big)$

> **Exercice 3**
>
> Donner un exemple d'expression ambigue

`if p then if q then r else s`

L'expression est ambvigue puisqu'on ne sait pas si le bloc `else` est associé à la condition 1, `if p`, ou à la condition 2 `if q`.

> **Exercice 4**
>
> Donner une grammaire non ambigue qui associe les `else` avec les `if` le plus proche, cmme le font les langage de programmation habituels.

$$\begin{aligned} S ::= &\; X \\ \mid &\text{ if } E \text{ then } S \text{ end} \\ \mid &\text{ if } E \text{ then } S \text{ else } S \text{ end} \end{aligned}$$

Cette grammaire force le `else` à être explicitement inclus avec son `if` correspondant. Chaque bloc est finalisé par la déclaration `end`, assurant ansi qu'un else peut uniquement être lié au `if` le plus proche.

$$\begin{aligned} S ::= &\; X \\ \mid &\text{ if (} E \text{) \{ } S \text{ \}} \\ \mid &\text{ if (} E \text{) \{ } S \text{ \} else \{ } S \text{ \}} \end{aligned}$$

Cette grammaire utilise des parenthèses et des accolades pour structurer les blocs, ce qui rend l'association entre if et else explicite et sans ambiguïté.

Section 2

EBNF

> **Exercice 5**
>
> Voici un exemple de programme dans un langage similaire 'a Javascript et la grammaire du langage en notation EBNF Compl´eter la grammaire pour qu'elle accepte le programme.

```
var ackermann = function (m, n) {
  if (m == 0) {
    return n + 1;
  } else if (n == 0) {
      return ackermann (m - 1, 1);
  } else {
      return ackermann (m - 1, ackermann (m , n - 1));
    };
```

Note :

Les éléments de grammaire en **rouge** sont les élément manquants

```
 /* Un programme correspond a une expression */
<program> ::= <instr>


 /* Une instruction est une expression suivis du terminal";" */
<instr> ::= <expr> ";"
             /* Ou une instruction suivie d'une autre */
           | <instr> <instr>

 /* Une expression peut etre une fonction, une comparaisonou un appel de fonction */
<expr> ::= "function" "(" <formals> ")" "{" <instr> "}"
           | <expr> "==" <expr>
           | <identifier> "(" <actuals> ")"
           | <expr> + <expr>
           | <expr> - <expr>
           | <number>
           | "if" "(" <expr.> ")" "{" <instr> "}" <else>


<else> :== "else" "{" <instr.> "}"
           | "else if" "(" <expr.> ")" "{" <instr.>"}" <else>
           | ε




 /* Les parametres formels d'une fonction peuvent etre */
<formals> ::=  ε    vides
              /* Ou un identifiant unique */
           | <identifier> { "," <identifier> }
              /* Ou une serie d'identifiants separes d'une virgule */


 /* Les parametre actuels d'un appel de fonction peuvent etre vide ou une expr. */
<actuals> ::=  ε  | <expr>


 /* Permet a une expression d'etre un nombre */

 /* Un identifiant est une serie de lettres */
 <identifier> ::= <letter> {letter}

 /* Les lettres acceptees sont celles de l'alphabet */
 <letter> ::= "a" - "z" | "A" - "Z"


 <number> ::= <digit> {digit}


 <digit> ::= |"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```