Le langage MISC (Mini-Scala)

Un sous-ensemble de Scala

Quatre types de données :

- 3 types de base : ***Any***, ***Unit*** et ***Int***,
- un type structuré : les listes,
- les types fonctionnels.

Deux constructions de contrôle :

- ***while*** (<*cond*>) <*expr*>
- ***if*** (<*cond*>) <*expr*> [***else*** <*expr*>]

Opérateurs classiques :

- arithmétiques : $+$, $-$, $*$, $/$, $\%$,
- relationnels : $==$, $!=$, $<$, $<=$, $>$, $>=$,
- logiques : $!$, $\&$, $|$.

Exemples

La factorielle en impératif :

```
def fact(x: Int): Int = {
    var p: Int = 1;
    while (x > 0) {
        p = p * x;
        x = x - 1;
    };
    p
};
printInt(fact(5))
```

La factorielle en fonctionnel :

```
def fact(x: Int): Int = if (x == 0) 1 else x * fact(x - 1);
printInt(fact(5))
```

Les types fonctionnels

Les fonctions étant des valeurs comme les autres, elles possèdent aussi un type. Leur syntaxe est donnée par la règle suivante :

$$\begin{aligned} \mathit{Type} &= \text{"("}\ \mathit{Types}\ \text{")"}\ \text{"}\Rightarrow\text{"}\ \mathit{Type} \\ &\mid \ldots \\ \mathit{Types} &= [\ \mathit{Type}\ \{\ \text{","}\ \mathit{Type}\ \}\ ] \end{aligned}$$

Exemple : Le type de la fonction

$$\textbf{\textit{def}}\ f(arg_1 : T_1,\ \ldots,\ arg_n : T_n\ ) \colon T_r\ = \ldots$$

est noté

$$(T_1,\ \ldots,\ T_n\ ) \Rightarrow T_r$$

Les types fonctionnels : exemples

Application de fonctions entières :

```
def apply(f: (Int) ⇒ Int, x: Int): Int = f(x);
def square(x: Int): Int = x * x;
printInt(apply(square, 5))
```

Composition de fonctions :

```
def compose(f: (Int) ⇒ Int, g: (Int) ⇒ Int, x: Int): Int = f(g(x));
def square(x: Int): Int = x * x;
def succ(x: Int): Int = x + 1;
printInt(compose (square, succ, 5))
```

Les listes

Les listes sont similaires aux tableaux dans d'autres langages, mais :

- pas d'accès aléatoire aux éléments,
- possibilité d'ajouter et de supprimer dynamiquement des éléments.

Les types listes sont notés ***List*** [a] où a est le type des éléments de la liste.
Exemples : ***List*** [***Int***], ***List*** [***List*** [(***Int***) $\Rightarrow$ ***Int***]]

Constantes, opérateurs et fonctions liés aux listes :

- liste vide : ***List*** ( ),
- ajout d'un élément : opérateur infixe :: associatif à droite,
- sucre syntaxique : ***List*** (...) (ex. ***List*** (*1*,*2*) est équivalent à *1* ::*2*::***List*** ( )),
- accès à la tête et à la queue : fonctions ***head*** et ***tail***,
- test si vide : fonction ***isEmpty***.

Les listes : exemples

Calcul de la longueur d'une liste d'entiers :

```
def length(l: List[Int]): Int =
  if (isEmpty(l))
    0
  else
    1 + length(tail(l));

printInt(length(List(1,2,3,4,5)))
```

Application d'une fonction à une liste d'entiers :

```
def map(f: (Int) ⇒ Int, l: List[Int]): List[Int] =
  if (isEmpty(l))
    []
  else
    f(head(l)) :: map(f, tail(l));

def square(x: Int): Int = x * x;

map(square, List(1,2,3,4,5))
```

Les fonctions prédéfinies

Lecture et écriture d'entiers :

def *readInt* ( ): ***Int*** = ...
def *printInt* (*i*: ***Int***): ***Unit*** = ...

Lecture et écriture de caractères :

def *readChar* ( ): ***Int*** = ...
def *printChar* (*c*: ***Int***): ***Unit*** = ...

Syntaxe lexicale

$$
\begin{array}{lcl}
\textit{input} & = & \{\ \textit{inputelement}\ \} \\
\textit{inputelement} & = & \textit{whitespace} \\
 & | & \textit{comment} \\
 & | & \textit{token} \\
 & & \\
\textit{token} & = & \textit{ident} \\
 & | & \textit{number} \\
 & | & \ldots \quad /*\ \text{à compléter}\ */ \\
\textit{comment} & = & \text{"/" "/"}\ \{\ \textit{cchar}\ \} \\
\textit{ident} & = & \textit{letter}\ \{\ \textit{letter} \mid \textit{digit} \mid \text{"_"}\ \} \\
\textit{number} & = & \text{"0"} \mid \textit{digit1}\ \{\ \textit{digit}\ \} \\
\textit{string} & = & \text{"\textbackslash""}\ \{\ \textit{schar}\ \}\ \text{"\textbackslash""} \\
 & & \\
\textit{whitespace} & = & \text{" "} \mid \text{"\textbackslash t"} \mid \text{"\textbackslash f"} \mid \text{"\textbackslash n"} \\
\textit{letter} & = & \text{"a"} \mid \ldots \mid \text{"z"} \mid \text{"A"} \mid \ldots \mid \text{"Z"} \\
\textit{digit} & = & \text{"0"} \mid \textit{digit1} \\
\textit{digit1} & = & \text{"1"} \mid \ldots \mid \text{"9"} \\
\textit{cchar} & = & \textit{tout caractère excepté "\textbackslash n".} \\
\textit{schar} & = & \textit{tout caractère excepté "\textbackslash n" et "\textbackslash "".}
\end{array}
$$

Syntaxe (1)

Program	=	*{ Declaration ";" } Body*
Declaration	=	*"**def**" ident "(" [Formals] ")" ":" Type "=" Expression*
Body	=	*{ Statement ";" } [Expression]*
Formal	=	*ident ":" Type*
Formals	=	*Formal { "," Formal }*
Type	=	*"**Unit**"*
	=	*"**Any**"*
	|	*"**Int**"*
	|	*"**List**" "[" Type "]"*
	|	*"(" Types ")" "⇒" Type*
Types	=	*[Type { "," Type }]*
Statement	=	*"**var**" Formal "=" Expression*
	|	*"**while**" "(" Expression ")" Expression*
	|	*Expression*

Syntaxe (2)

$$
\begin{array}{lcl}
\mathit{Expression} & = & \text{"}\textbf{if}\text{"}\ \text{"(" } \mathit{Expression}\ \text{")" } \mathit{Expression}\ [\ \text{"}\textbf{else}\text{"}\ \mathit{Expression}\ ] \\
& | & \mathit{ident}\ \text{"="}\ \mathit{Expression} \\
& | & \mathit{OrExpression} \\
\mathit{OrExpression} & = & \mathit{AndExpression} \\
& | & \mathit{OrExpression}\ \text{"|"}\ \mathit{AndExpression} \\
\mathit{AndExpression} & = & \mathit{CmpExpression} \\
& | & \mathit{AndExpression}\ \text{"\&"}\ \mathit{CmpExpression} \\
\mathit{CmpExpression} & = & [\ \mathit{ListExpression}\ \mathit{CompOp}\ ]\ \mathit{ListExpression} \\
\mathit{CompOp} & = & \text{"=="} \mid \text{"!="} \mid \text{"<"} \mid \text{">"} \mid \text{"<="} \mid \text{">="} \\
\mathit{ListExpression} & = & \mathit{SumExpression} \\
& | & \mathit{SumExpression}\ \text{"::"}\ \mathit{ListExpression} \\
\mathit{SumExpression} & = & \mathit{Term} \\
& | & \mathit{SumExpression}\ \mathit{SumOp}\ \mathit{Term} \\
\mathit{SumOp} & = & \text{"+"} \mid \text{"-"}
\end{array}
$$

Syntaxe (3)

$$
\begin{array}{lcl}
\textit{Term} & = & [\ \textit{Term ProdOp}\ ]\ [\ \textit{NegateOp}\ ]\ \textit{Factor} \\
\textit{ProdOp} & = & \text{"*"} \mid \text{"/"} \mid \text{"\%"} \\
\textit{NegateOp} & = & \text{"-"} \mid \text{"!"} \\
\textit{Factor} & = & \textit{ident} \\
 & \mid & \textit{number} \\
 & \mid & \textit{string} \\
 & \mid & \text{"true"} \\
 & \mid & \text{"false"} \\
 & \mid & \text{"("}\ [\ \textit{Expression}\ ]\ \text{")"} \\
 & \mid & \text{"\{"}\ \textit{Body}\ \text{"\}"} \\
 & \mid & \textit{PredefFun Params} \\
 & \mid & \text{"}\textbf{\textit{List}}\text{"}\ \textit{Params} \\
 & \mid & \textit{Factor Params} \\
\textit{PredefFun} & = & \text{"}\textbf{\textit{head}}\text{"} \mid \text{"}\textbf{\textit{tail}}\text{"} \mid \text{"}\textbf{\textit{isEmpty}}\text{"} \\
\textit{Params} & = & \text{"("}\ \textit{Expressions}\ \text{")"} \\
\textit{Expressions} & = & [\ \textit{Expression}\ \{\ \text{","}\ \textit{Expression}\ \}\ ]
\end{array}
$$