Descrizione del linguaggio Scala40

sssbbf

1 Soluzioni non standard

2 Descrizione soluzione

3 Caratteristiche generali di Scala40

TODO

4 Struttura lessicale di Scala40

Parole riservate

Le parole riservate in **Scala40** sono le seguenti:

`if`, `else`, `do`, `while`, `def`, `return`, `var`, `Array`, `False`, `True`, `Null`, `Char`, `String`, `Int`, `Float`, `Bool`.

Identificatori

Un identificatore $\langle Ident \rangle$ è una lettera seguita da una sequenza arbitraria di lettere, cifre e del carattere '_'.

Letterali

Vi sono letterali per numeri interi, numeri in virgola mobile, singoli caratteri, booleani, stringhe. Essi seguono le convenzioni della maggior parte dei linguaggi di programmazione.

$$
\begin{aligned}
\langle Literal \rangle ::= & \langle Int \rangle \\
& | \langle Float \rangle \\
& | \langle Char \rangle \\
& | \langle Bool \rangle \\
& | \langle String \rangle \\
& | \texttt{Null}
\end{aligned}
$$

Commenti

I commenti in **Scala40** sono di due tipi:

- i commenti di una riga sono sequenze di caratteri che iniziano con `\\` e finiscono al termine della riga;
- i commenti multi-riga sono sequenze di caratteri che iniziano con `/*` e terminano con `*/`. Non possono essere annidati.

Caratteri di spaziatura

I token possono essere separati dai caratteri di spaziatura standard o commenti.

5 Struttura sintattica di Scala40

- Un *programma* è una sequenza di dichiarazioni.
- Una *dichiarazione* ha una delle seguenti forme:
  - *Dichiarazione di variabili*

$$\begin{aligned}\langle Decl\rangle ::= \texttt{var}\ \langle Ident\rangle\texttt{:}\ \langle TypeSpec\rangle\ \texttt{=}\ \langle Expr\rangle\ \texttt{;}\\ \mid \texttt{var}\ \langle Ident\rangle\texttt{:}\ \langle TypeSpec\rangle\texttt{;}\end{aligned}$$

  - *Dichiarazione di funzioni e procedure*

$$\begin{aligned}\langle Decl\rangle ::= \texttt{def}\ \langle Ident\rangle\ \langle ParamClauses\rangle\texttt{:}\ \langle TypeSpec\rangle\ \texttt{=}\ \langle Expr\rangle\ \texttt{;}\\ \texttt{def}\ \langle Ident\rangle\ \langle ParamClauses\rangle\texttt{:}\ \langle TypeSpec\rangle\ \texttt{=}\ \langle Block\rangle\\ \texttt{def}\ \langle Ident\rangle\ \langle ParamClauses\rangle\ \texttt{=}\ \langle Expr\rangle\ \texttt{;}\\ \texttt{def}\ \langle Ident\rangle\ \langle ParamClauses\rangle\ \texttt{=}\ \langle Block\rangle\end{aligned}$$

  dove $\langle TypeSpec\rangle$ è una specifica di tipo, che ha la forma

$$\begin{aligned}\langle TypeSpec\rangle ::= \langle SimpleType\rangle\\ \mid \texttt{*}\ \langle TypeSpec\rangle\\ \mid \texttt{Array [}\ \langle TypeSpec\rangle\ \texttt{]}\ \texttt{(}\ \langle Int\rangle\ \texttt{)}\\ \langle SimpleType\rangle ::= \texttt{Bool} \mid \texttt{Char} \mid \texttt{Int} \mid \texttt{Float} \mid \texttt{String}\end{aligned}$$

  mentre l'elemento $\langle ParamClauses\rangle$ è una sequenza, non vuota, di $\langle ParamClause\rangle$, e ciascun $\langle ParamClause\rangle$ ha la forma

$$\langle ParamClause\rangle ::= \texttt{(}\ \langle Args\rangle\ \texttt{)}$$

⟨*Args*⟩ è una sequenza, che può essere vuota, di elementi separati da virgola della forma

$$\langle Arg \rangle ::= \langle Ident \rangle \texttt{ : } \langle TypeSpec \rangle$$

Ad esempio, una specifica di tipo valida è `Array[*Int](2)`, che indica un array di puntatori ad interi di dimensione 2. Ad esempio, una definizione di funzione valida è

```
def foo(a: Array[*Int](2),p: *Int)(x: Int): Int =
*a[1] + *a[2] + *p + x;
```

Essa prende come parametri un array di puntatori ad interi, un puntatore ad un intero e un intero e restituisce un intero.

- Un *blocco* è una sequenza di istruzioni racchiuse fra parentesi graffe.

$$\langle Block \rangle ::= \texttt{\{ } \langle StmtList \rangle \texttt{ \}}$$

- Una *istruzione* ha la forma:

$$\begin{aligned}
\langle Stmt \rangle ::= & \langle Decl \rangle \\
| & \langle Block \rangle \\
| & \langle LExpr \rangle \ \langle OpAssign \rangle \ \langle Expr \rangle \texttt{ ;} \\
| & \texttt{if (} \langle Expr \rangle \texttt{)} \ \langle Stmt \rangle \\
| & \texttt{if (} \langle Expr \rangle \texttt{)} \ \langle Stmt \rangle \texttt{ else} \ \langle Stmt \rangle \\
| & \texttt{while (} \langle Expr \rangle \texttt{)} \ \langle Stmt \rangle \\
| & \texttt{do} \ \langle Stmt \rangle \texttt{ while (} \langle Expr \rangle \texttt{) ;} \\
| & \texttt{return ;} \\
| & \texttt{return} \ \langle Expr \rangle \texttt{ ;} \\
| & \langle Ident \rangle \ \langle ParamsList \rangle \texttt{ ;}
\end{aligned}$$

dove, nell'ultima istruzione, che corrisponde alla chiamata di procedura o funzione, ⟨*ParamsList*⟩ è una sequenza, non vuota, di ⟨*Params*⟩ della forma

$$\langle Params \rangle ::= \texttt{(} \langle ExprList \rangle \texttt{)}$$

e ⟨*ExprList*⟩ è una sequenza, che può essere vuota, di ⟨*Expr*⟩ separate da virgola.

Invece ⟨*OpAssign*⟩ è uno dei seguenti operatori di assegnamento:

$$\langle OpAssign \rangle ::= \texttt{=} \mid \texttt{+=} \mid \texttt{-=} \mid \texttt{*=} \mid \texttt{/=} \mid \texttt{\%=}$$

- Le *left expressions* del linguaggio hanno la seguente forma:

$$\begin{aligned}\langle LExpr\rangle ::= &\ \langle Ident\rangle \\ &| \langle LExpr\rangle \texttt{ [} \langle Expr\rangle \texttt{]} \\ &| \texttt{*} \langle LExpr\rangle \\ &| \texttt{ (} \langle LExpr\rangle \texttt{)}\end{aligned}$$

  L'operatore accesso ad array `[]` ha la precedenza sull'operatore di dereference `*`. Quindi ad esempio `*a[1]` è sintatticamente equivalente a `*(a[1])`.

- Le *right expressions* del linguaggio hanno la seguente forma:

$$\begin{aligned}\langle Expr\rangle ::= &\ \langle Literal\rangle \\ &| \langle LExpr\rangle \\ &| \texttt{\&} \langle LExpr\rangle \\ &| \texttt{Array (} \langle ExprList\rangle \texttt{)} \\ &| \langle Ident\rangle\ \langle ParamsList\rangle \\ &| \langle Expr\rangle\ \langle BinOp\rangle\ \langle Expr\rangle \\ &| \langle UnOp\rangle\ \langle Expr\rangle \\ &| \texttt{ (} \langle Expr\rangle \texttt{)}\end{aligned}$$

$$\langle BinOp\rangle ::= \texttt{||} \mid \texttt{\&\&} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{==} \mid \texttt{!=} \mid \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{\^{}}$$
$$\langle UnOp\rangle ::= \texttt{!} \mid \texttt{-}$$

  Gli operatori hanno precedenze e associatività standard.

6 Vincoli di semantica statica di Scala40

TODO