Corso di Compilatori A.A. 2023/24

Bacco Luigi, Matricola: 0522501773
Valletta Paolo Carmine, Matricola: 0522501828

Gennaio 2024

1 Introduzione

Il seguente documento contiene le specifiche del linguaggio Toy2.

- La sezione delle specifiche lessicali contiene la lista dei token con i rispettivi pattern
- La sezione delle specifiche sintattiche contiene la grammatica utilizzata con all'interno tutti i non terminali (scritti in minuscolo e maiuscolo) ed i terminali (scritti interamente in maiuscolo), questa sezione contiene inoltre la tabella delle precedenze e i nodi dell'Abstract Syntax Tree
- La sezione delle specifiche semantiche riporta le regole di Type Checking e le tabelle per gli operatori
- La sezione di testing mostra tutti i test effettuati
- La sezione modifiche aggiuntive elenca tutte le modifiche o migliorie effettuate al compilatore

2 Specifiche Lessicali

Token	Pattern
VAR	var
COLON	:
ASSIGN	^ =
SEMI	;
COMMA	,
TRUE	true
FALSE	false
REAL	real
INTEGER	integer
STRING	string
BOOLEAN	boolean
RETURN	return
FUNCTION	func
TYPERETURN	− >
LPAR	(
RPAR	)
PROCEDURE	proc
WHILE	while
ENDPROCEDURE	endproc
ENDFUNCTION	endfunc
OUT	out
WRITE	−− >
WRITERETURN	−− >!
DOLLARSIGN	$
READ	< −−
IF	if
THEN	then
ELSE	else
ENDIF	endif
ELIF	elseif
DO	do
ENDWHILE	endwhile
PLUS	+
MINUS	-
TIMES	*
DIV	/
EQ	=
NE	<>
LT	<
LE	<=
GT	>
GE	>=
AND	&&
OR	||
NOT	!
ENDVAR	\\
REF	@
ID	[a-zA-Z] ([a-zA-Z] | [0-9] | _)*
STRING_CONST	\" ∼\"
INTEGER_CONST	[0-9]+
REAL_CONST	[0-9]+ (“.” [0-9]+)?

3 Specifiche Sintattiche

3.1 Grammatica

Program ::= Iter Procedure Iter

IterNoProcedure ::= VarDecls IterNoProcedure
    | Function IterNoProcedure
    | /* empty */

Iter ::= VarDecl Iter
    | Function Iter
    | Procedure Iter
    | /* empty */

VarDecl ::= VAR Decls

Decls ::= Ids COLON Type SEMI Decls
    | Ids ASSIGN Consts SEMI Decls
    | Ids COLON Type SEMI ENDVAR
    | Ids ASSIGN Consts SEMI ENDVAR

Ids ::= ID COMMA Ids
    | ID

Consts ::= Const COMMA Consts
    | Const

Const ::= REAL_CONST
    | INTEGER_CONST
    | STRING_CONST
    | TRUE
    | FALSE

Type ::= REAL
    | INTEGER
    | STRING
    | BOOLEAN

Function ::= FUNCTION ID LPAR FuncParams RPAR TYPERETURN Types
COLON Body ENDFUNCTION

FuncParams ::= ID COLON Type OtherFuncParams

```
    | /* empty */

OtherFuncParams ::= COMMA ID COLON Type OtherFuncParams
    | /* empty */

Types ::= Type COMMA Types
    | Type

Procedure ::= PROCEDURE ID LPAR ProcParams RPAR COLON Body
ENDPROCEDURE

ProcParams::= ProcParamId COLON Type OtherProcParams
    | /* empty */

OtherProcParams ::= COMMA ProcParamId COLON Type OtherProcParams
    | /* empty */

ProcParamId ::= ID
    | OUT ID

Body ::= VarDecl Body
    | Stat Body
    | /* empty */

Stat ::= Ids ASSIGN Exprs SEMI
    | ProcCall SEMI
    | RETURN Exprs SEMI
    | WRITE IOArgs SEMI
    | WRITERETURN IOArgs SEMI
    | READ IOArgs SEMI
    | IfStat SEMI
    | WhileStat SEMI

FunCall ::= ID LPAR Exprs RPAR
    | ID LPAR RPAR

ProcCall ::= ID LPAR ProcExprs RPAR
    | ID LPAR RPAR

IfStat ::= IF Expr THEN Body Elifs Else ENDIF

Elifs ::= Elif Elifs
    | /* empty */

Elif ::= ELIF Expr THEN Body
```

```
Else ::= ELSE Body
    | /* empty */

WhileStat ::= WHILE Expr DO Body ENDWHILE

IOArgs ::= OtherIOArgs IOArgs
    | DOLLARSIGN LPAR Expr RPAR IOArgs
    | /* empty */

OtherIOArgs ::= OtherIOArgs PLUS OtherIOArgs
    | STRING_CONST

ProcExprs::= Expr COMMA ProcExprs
    | REF ID COMMA ProcExprs
    | Expr
    | REF ID

Exprs ::= Expr COMMA Exprs
    | Expr

Expr ::= FunCall
    | REAL_CONST
    | INTEGER_CONST
    | STRING_CONST
    | ID
    | TRUE
    | FALSE
    | Expr PLUS Expr
    | Expr MINUS Expr
    | Expr TIMES Expr
    | Expr DIV Expr
    | Expr AND Expr
    | Expr OR Expr
    | Expr GT Expr
    | Expr GE Expr
    | Expr LT Expr
    | Expr LE Expr
    | Expr EQ Expr
    | Expr NE Expr
    | LPAR Expr RPAR %PAR
    | MINUS Expr %UMINUS
    | NOT Expr
```

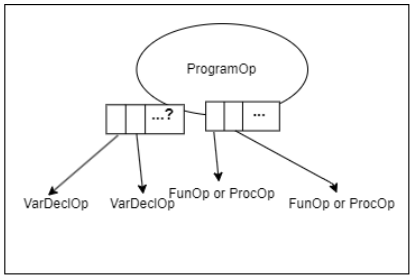
3.2 Tabella delle precedenze

La priorità della seguente tabella viene specificata come nell'ordine fornito da Java CUP, riga più in basso equivale a priorità più alta.

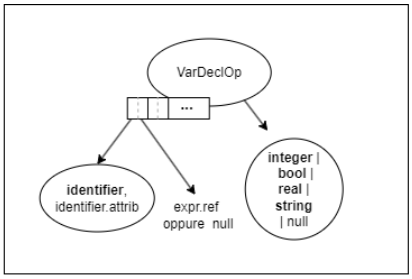
Token	Associatività
ID VAR	SINISTRA
COMMA	SINISTRA
ASSIGN	DESTRA
OR	SINISTRA
AND	SINISTRA
EQ NE	SINISTRA
LE GE GT LT	SINISTRA
PLUS MIUS	SINISTRA
TIMES DIV	SINISTRA
NOT UMINUS	DESTRA
PAR	SINISTRA

3.3 Abstract Syntax Tree

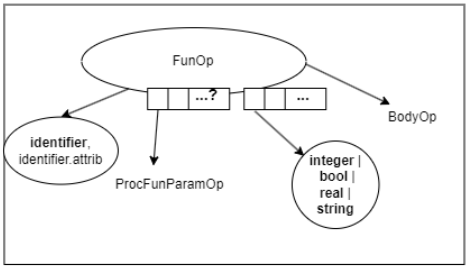
Nodo ProgramOp



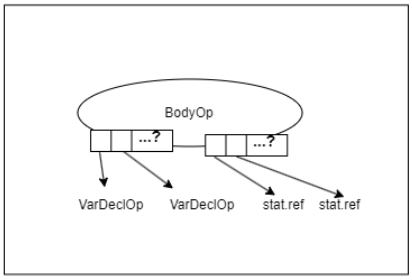
Nodo VarDeclOp



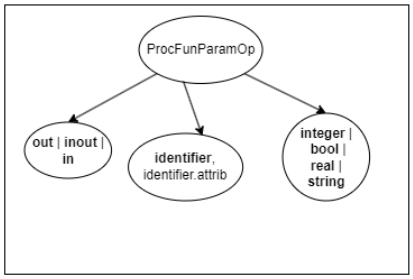
Nodo FunOp



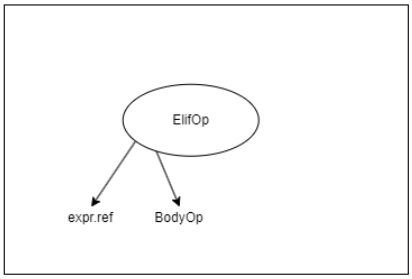
Nodo BodyOp



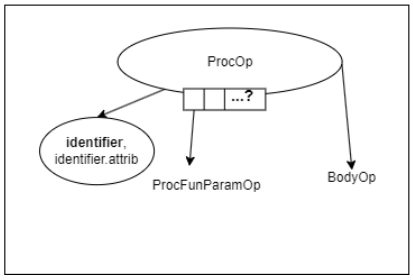
Nodo ProcFunParamOp



Nodo ElifOp

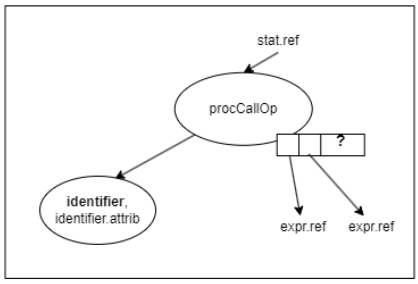


Nodo ProcOp

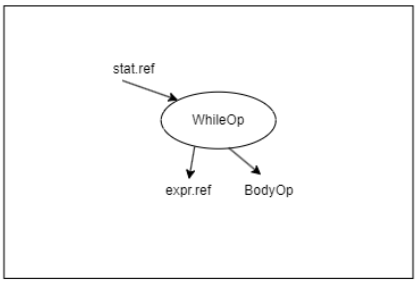


Nodo Stat

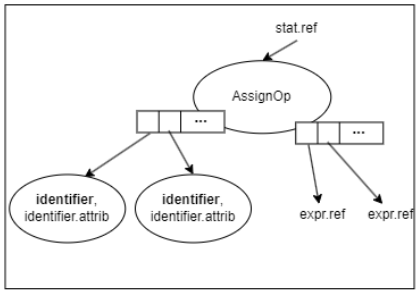
Nodo ProcCallOp



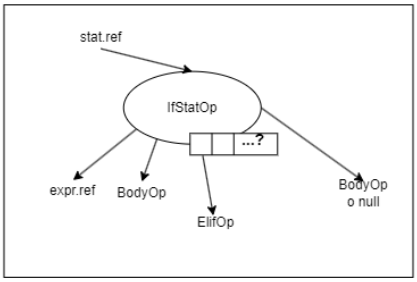
Nodo WhileOp



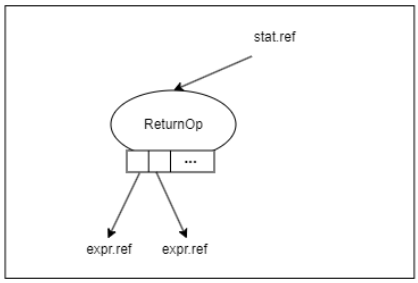
Nodo AssignOp



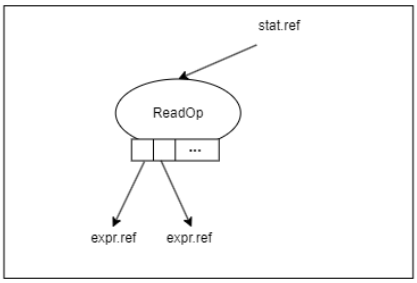
Nodo IfStatOp



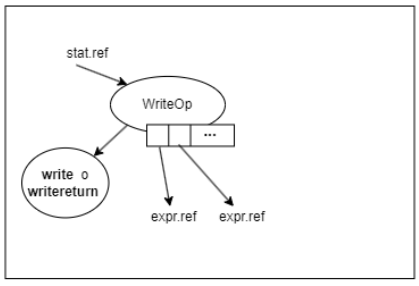
Nodo ReturnOp

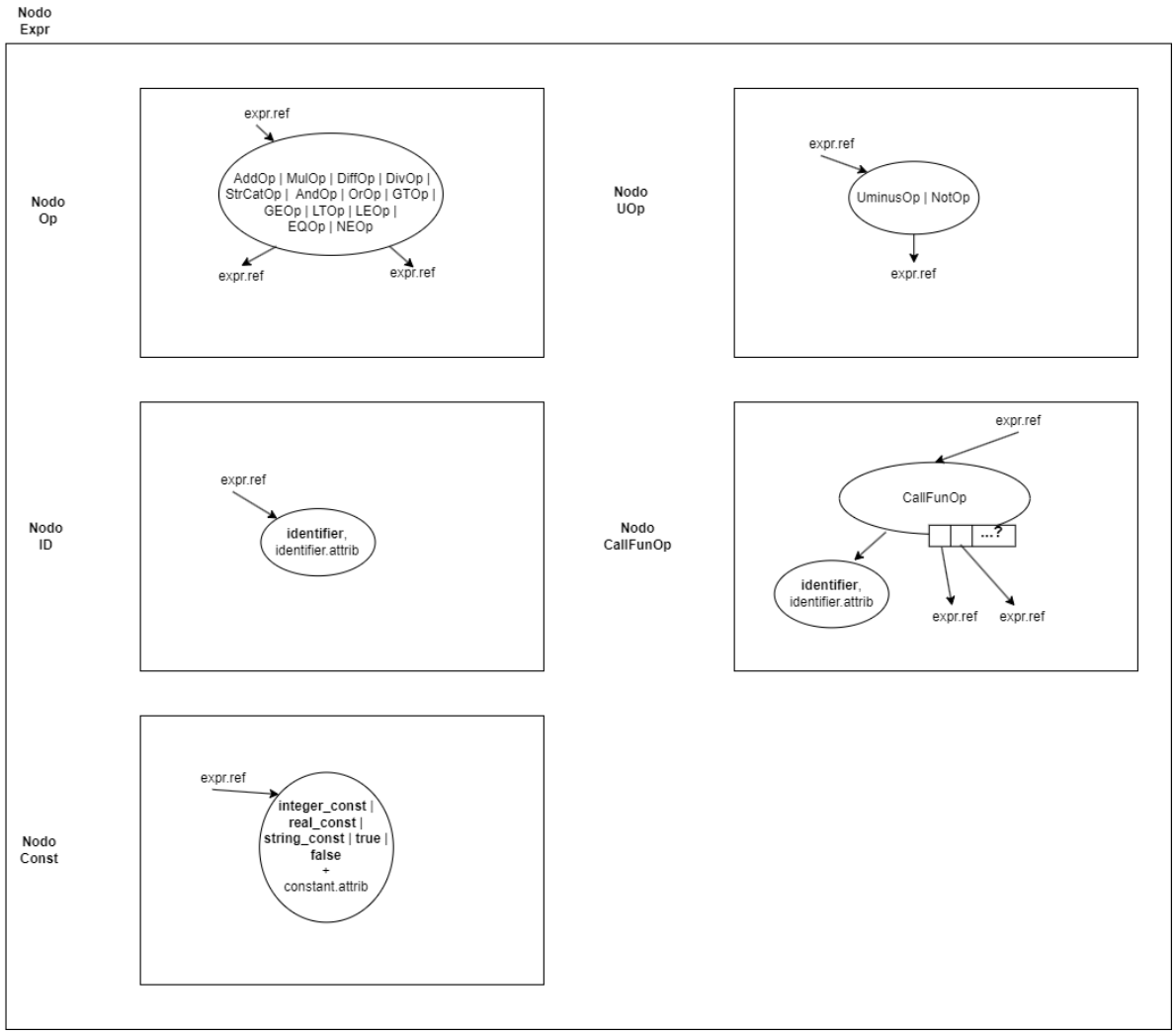


Nodo ReadOp



Nodo WriteOp




Nodo
Expr
Nodo
Op
expr.ref
AddOp | MulOp | DiffOp | DivOp |
StrCatOp | AndOp | OrOp | GTOp |
GEOp | LTOp | LEOp |
EQOp | NEOp
expr.ref
expr.ref
Nodo
UOp
expr.ref
UminusOp | NotOp
expr.ref
Nodo
ID
expr.ref
identifier,
identifier.attrib
Nodo
CallFunOp
expr.ref
CallFunOp
...?
identifier,
identifier.attrib
expr.ref
expr.ref
Nodo
Const
expr.ref
integer_const |
real_const |
string_const | true |
false
+
constant.attrib

4 Specifiche Semantiche

Identificatore

$$\frac{\Gamma(id)=\tau}{\Gamma\vdash id:\tau}$$

Costanti

$\Gamma \vdash$ real_const : real
$\Gamma \vdash$ integer_const : integer
$\Gamma \vdash$ string_const : string
$\Gamma \vdash$ true : boolean
$\Gamma \vdash$ false : boolean

Lista di istruzioni

$$\frac{\Gamma\vdash\ stmt_1 : notype\ \Gamma\vdash\ stmt_2 : notype}{\Gamma\vdash\ stmt_1,\ stmt_2 : notype}$$

Chiamata a funzione

$$\frac{\Gamma\vdash\ f : \tau_1\times...\times\tau_n -> \sigma_1...\sigma_m\ \Gamma\vdash e_i : \tau_i^{i\in 1...n}}{\Gamma\vdash f(e_1,...,e_n) : \sigma_1...\sigma_m}$$

Chiamata a procedura

$$\frac{\Gamma\vdash\ f : \tau_1\times...\times\tau_n -> notype\ \Gamma\vdash e_i : \tau_i^{i\in 1...n}}{\Gamma\vdash f(e_1,...,e_n) : notype}$$

Assegnazione

$$\frac{\Gamma(id_i) : \tau_i^{i\in 1...n}\ \Gamma\vdash e_i : \tau_i^{i\in 1...n}}{\Gamma\vdash id_1,...,id_n \wedge = e_1,...,e_n : notype}$$

Dichiarazione-Istruzione

$$\frac{\Gamma[id\rightarrow\tau]\vdash stmt : notype}{\Gamma\vdash\tau\ id;\ stmt : notype}$$

While

$$\frac{\Gamma\vdash e : boolean\ \Gamma\vdash body : notype}{\Gamma\vdash \textbf{while}\ e\ \textbf{do}\ body\ \textbf{endwhile} : notype}$$

If-elseif-else

$$\frac{\Gamma\vdash e_1 : boolean\ \Gamma\vdash body_1 : notype\ \Gamma\vdash e_2,\ ...,\ e_n : boolean\ \Gamma\vdash body_2,\ ...,\ body_n : notype\ body_{n+1} : notype}{\Gamma\vdash \textbf{if}\ e_1\ \textbf{then}\ body_1\ \textbf{elseif}\ e_2\ \textbf{then}\ body_2\ ...\ \textbf{elseif}\ e_n\ \textbf{then}\ body_n\ \textbf{else}\ body_{n+1}\ \textbf{endif} : notype}$$

Write

$$\frac{\Gamma\vdash e : \tau}{\Gamma\vdash --> e : notype}$$

Read

$$\frac{\Gamma(id{=}\tau)\ \Gamma{\vdash}e : \tau}{\Gamma{\vdash}{<}{-}{-}\ id\ e : notype}$$

Return

$$\frac{\Gamma{\vdash}e_i : \tau_i^{\,i\in 1\ldots n}}{\Gamma{\vdash}return\ e_1, \ldots, e_n : notype}$$

Operatori Unari

$$\frac{\Gamma{\vdash}e : \tau_1\ optype1(op_1, \tau_1) = \tau}{\Gamma{\vdash}op_1\ e : \tau}$$

op1	operando	risultato
MINUS	integer	integer
MINUS	real	real
NOT	boolean	boolean

Operatori Binari

$$\frac{\Gamma{\vdash}e_1 : \tau_1\ \Gamma{\vdash}e_2 : \tau_2\ optype2(op_2, \tau_1, \tau_2) = \tau}{\Gamma{\vdash}e_1\ op_2\ e_2 : \tau}$$

op2	operando1	operando2	risultato
DIV	integer	integer	real
ADD	string	integer	string
ADD	integer	string	string
ADD	string	real	string
ADD	real	string	string
ADD	string	boolean	string
ADD	boolean	string	string
ADD, MINUS, TIMES	integer	integer	integer
ADD, MINUS, TIMES, DIV	integer	real	real
ADD, MINUS, TIMES, DIV	real	integer	real
AND, OR	boolean	boolean	boolean
EQ, NE	integer	integer	boolean
EQ, NE	real	integer	boolean
EQ, NE	integer	real	boolean
EQ, NE	real	real	boolean
EQ, NE	string	string	boolean
EQ, NE	boolean	boolean	boolean
LT, LE, GT, GE	integer	integer	integer
LT, LE, GT, GE	real	integer	integer
LT, LE, GT, GE	integer	real	integer
LT, LE, GT, GE	real	real	integer

5 Test

Test	Descrizione
TestCubo	Pass
TestCuboError	Errore: endvar mancante
TestFibonacci	Pass
TestFibonacciError	Errore: manca il return
TestTabelline	Pass
TestTabellineError	Errore: il while non viene chiuso
TestConvertitore	Pass
TestConvertitoreError	Errore: gradi è di tipo integer, ma viene utilizzato come se fosse real
TestContaspazi	Pass
TestContaspaziError	Errore: c non è stato dichiarato
Sample	Pass
ProgramEs5	Pass

6 Modifiche Aggiuntive

- All'interno delle funzioni viene effettuato il controllo del return, se è presente un blocco if, in tutti i body del blocco if, ovvero anche gli elseif ed else. Ovviamente il compilatore non darà errore se è stato inserito un return fuori dal blocco if. Questa modifica funziona anche in presenza di più blocchi if adiacenti o innestati. Esempi di codice corretti e scorretti:
  - Corretto

```
func funzione(num : integer)->integer:
    if num>0 then
        num^=3;
    endif;
    return num;
endfunc
```

  - Corretto

```
func funzione(num : integer)->integer:
    if num>0 then
        num^=3;
        return num;
    endif;
endfunc
```

  - Corretto

```
func funzione(num : integer)->integer:
    if num>0 then
        num^=3;
```

```
        return num;
    elseif num<3 then
        num^=0;
        return num;
    else
        return 0;
    endif;
endfunc
```

– Non Corretto: manca return nell'else

```
func funzione(num : integer)->integer:
    if num>0 then
        num^=3;
        return num;
    else
        num^=0;
    endif;
endfunc
```

– Non Corretto: manca return nell'elseif

```
func funzione(num : integer)->integer:
    if num>0 then
        num^=3;
        return num;
    elseif num<3 then
        num^=0;
    endif;
endfunc
```

– Non Corretto: manca return nell'else

```
func funzione(num : integer)->integer:
    if num>0 then
        num^=3;
        return num;
    elseif num<3 then
        num^=0;
        return num;
    else
        num^=3;
    endif;
endfunc
```