Documentazione di Toy2

Implementazione del Compilatore Toy2

Benedetto Scala
Leopoldo Todisco

Università degli Studi di Salerno
Dipartimento di Informatica

Contents

Chapter 1

Specifiche Lessicali

1.1 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	< −−

Table 1.1: Tabella dei token e dei relativi pattern

Token	**Pattern**
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]+)?

Table 1.2: Continuo della tabella dei token e pattern

Chapter 2

Specifiche Sintattiche

2.1 Grammatica

Rispetto alla grammatica fornitaci è stato necessario includere due nuovi non-terminali: IterWithoutProcedure e IOArgsExpr. IterWithoutProcedure è identico a Iter ma al suo interno non ha produzioni con Procedure, mentre IOArgsExpr serve per le operazioni I/O

2.1.1 Grammatica

Program ::= Iter Procedure Iter

IterWithoutProcedure ::= VarDecls IterWithoutProcedure
| Function IterWithoutProcedure
| /* 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 */

IOArgsExpr ::= ID
| IOArgsExpr PLUS IOArgsExpr
| 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
| MINUS Expr %prec UMINUS
| NOT Expr

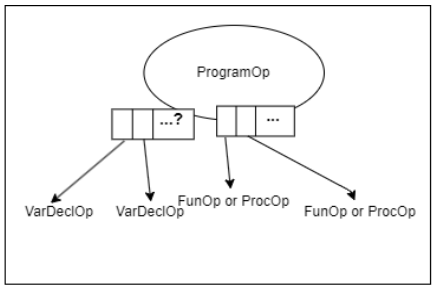
2.1.2 Tabella delle precedenze

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

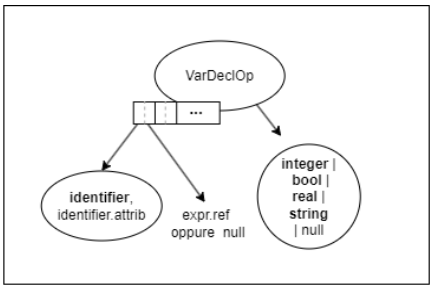
Token	**Associatività**
OR	SINISTRA
AND	SINISTRA
NOT	SINISTRA
EQ NE LE GE GT LT	NONASSOC
PLUS MINUS	SINISTRA
TIMES DIV	SINISTRA

2.1.3 Abstract Syntax Tree

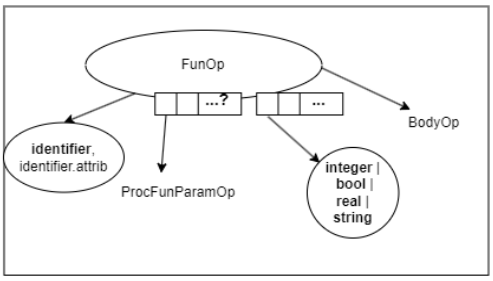
Nodo ProgramOp



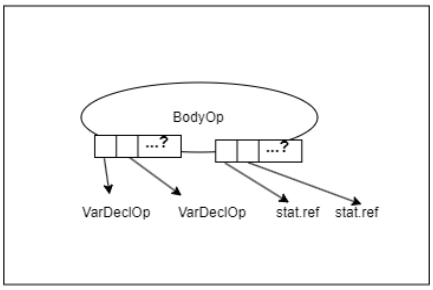
Nodo VarDeclOp



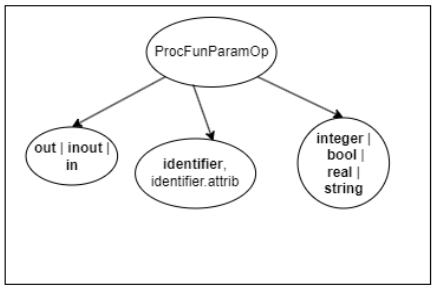
Nodo FunOp



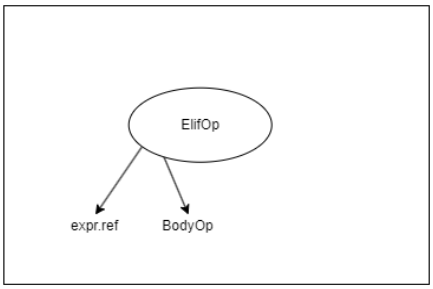
Nodo BodyOp



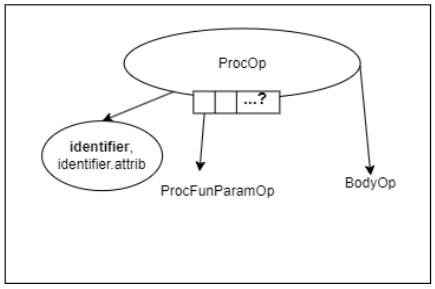
Nodo ProcFunParamOp



Nodo ElifOp

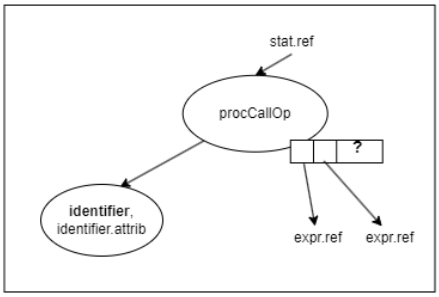


Nodo ProcOp

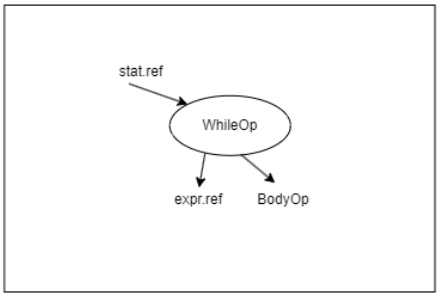


Nodo
Stat

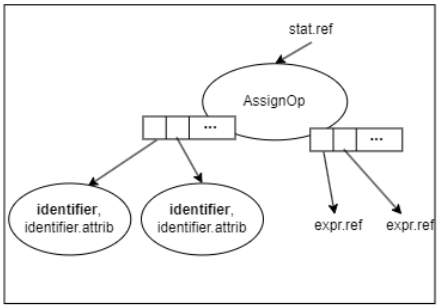
Nodo
ProcCallOp



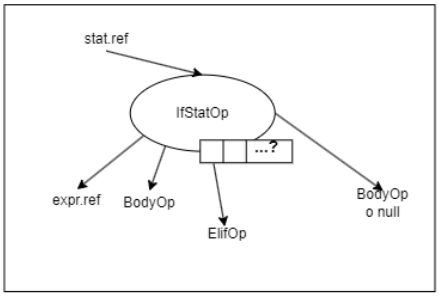
Nodo
WhileOp



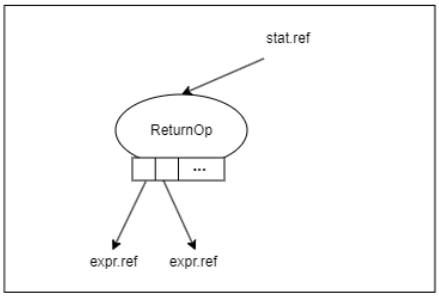
Nodo
AssignOp



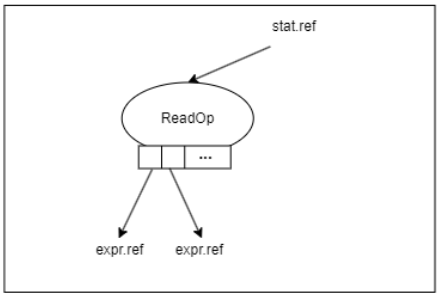
Nodo
IfStatOp



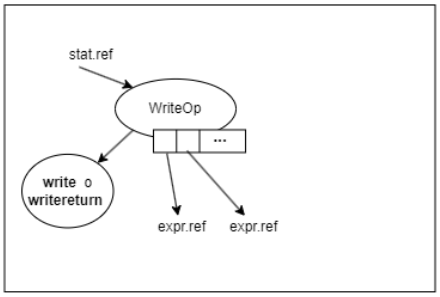
Nodo
ReturnOp

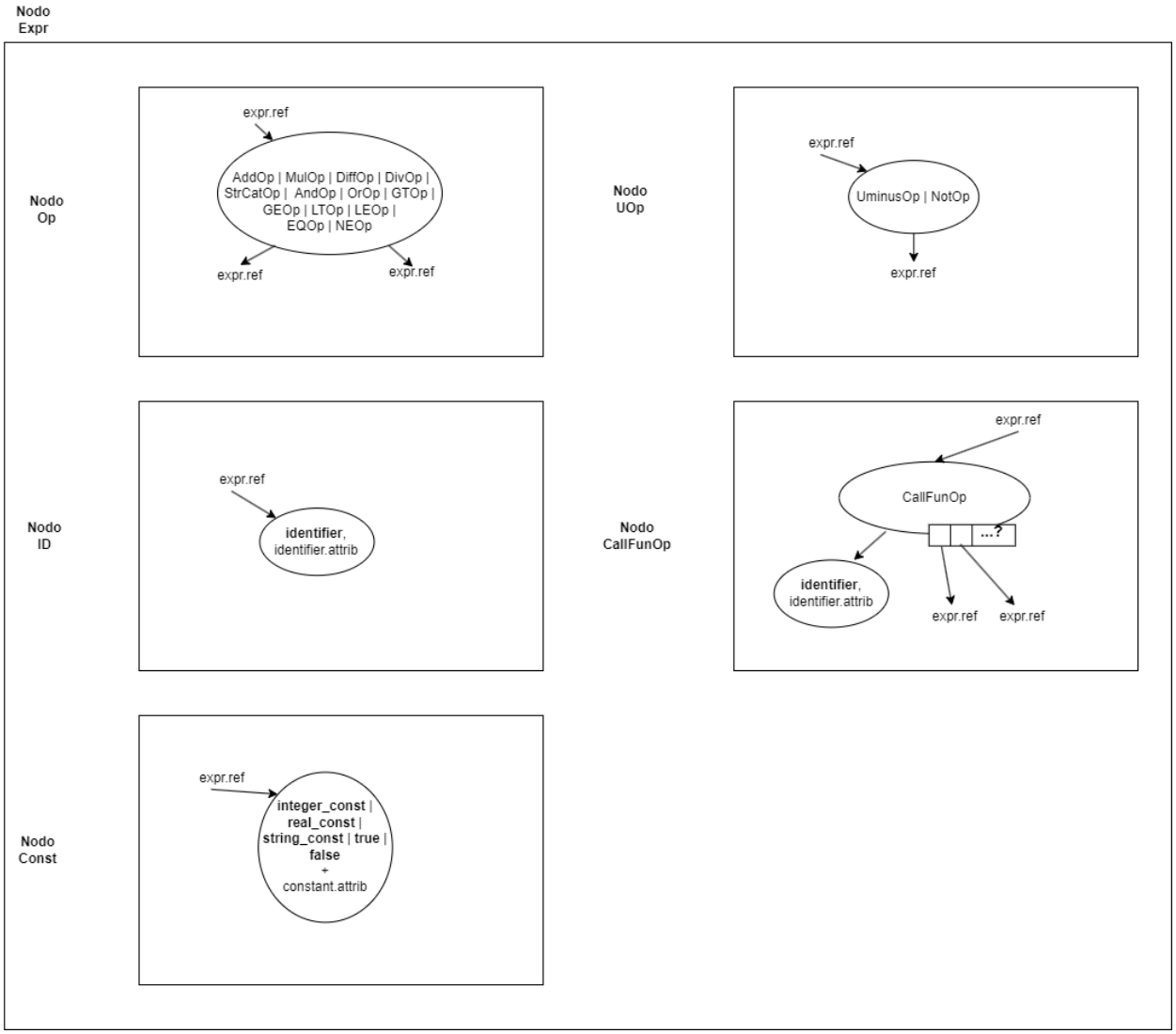


Nodo
ReadOp



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

Chapter 3

Specifiche Semantiche

Identificatore

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

Costanti

$\Gamma \vdash \text{real\_const} : \text{real}$

$\Gamma \vdash \text{integer\_const} : \text{integer}$

$\Gamma \vdash \text{string\_const} : \text{string}$

$\Gamma \vdash \text{true} : \text{boolean}$

$\Gamma \vdash \text{false} : \text{boolean}$

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

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

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

Chapter 4

Test forniti

Nella tabella sottostante sono elencati tutti i test che sono stati forniti e eseguiti. I risultati includono sia i casi in cui i test sono stati superati con successo che quelli in cui si sono verificati errori, in questi casi abbiamo aggiunto il messaggio dell'eccezione.

Test	**Descrizione**
valid1	Pass
valid2	Pass
valid3	Pass
valid4	Pass
invalid_bad_funcall	Errore: Parametri dichiarati nella chiamata di funzione non matchano quelli utilizzati
invalid_bad_proc_decl	Errore: hai dichiarato una procedura con un return
invalid_bad_return_type	Errore: I tipi dei parametri usati nel return non matchano con quelli usati nella funzione
invalid_bad_write_argument	Syntax error
invalid_fun_redeclaration	Il simbolo 'main' è dichiarato più di una volta nello scope
invalid_fun_out_argument	Il test in questione è in realtà corretto, sebbene segnato come invalido
invalid_no_out_param	Il numero di out e di ref non matcha
invalid_no_var_declaration	id non dichiarato: result
invalid_num_param	il numero di parametri in procedura non matcha
invalid_out_expression	il numero di out e ref non matcha
invalid_param_type	Non puoi usare out quando fai il cast da integer a real
invalid_return	Non hai un returrn nella funzione: fib
invalid_return_mult_assign	Numero di elementi a destra e sinistra di assign non è uguale I tipi dei parametri usati nel return non matchano con quelli usati nella funzione, tipi nel return [REAL, REAL, REAL, REAL] tipi nella dichiarazione[REAL, REAL, REAL]
invalid_var_mult_assign	Numero di elementi a destra e sinistra di assign non è uguale
invalid_var_mult_assign2	Nell'avvenire dell'inizializzazione il numero delle costanti deve essere pari al numero degli id
invalid_var_redeclaration	Il simbolo 'n' è dichiarato più di una volta nello scope

Table 4.1: tabella dei test

Si noti che:

- nel test *invalid_no_out_argument*, segnato come invalid, non vi era l'errore atteso, ossia che un parametro out non fosse passato come riferimento. Abbiamo introdotto noi l'errore.
- nel test *valid2* alla linea 1 del file valid2_in.txt c'era *tmoltiplicazione* anzichè *moltiplicazione*, sebbene il test funzionasse ugualmente in C, vi erano delle differenze nella CI/CD tra ciò che veniva dato in output e l'oracolo, pertanto si è deciso di sostituire tmoltiplicazione con moltiplicazione;
- nel test *valid2* alla linea 35 del file valid_2.txt nello statement writereturn si stampava due volte *res3* anzichè stampare *res4*. Anche in questo caso il test passava regolarmente, tuttavia vi erano delle differenze con l'oracolo. Per questo motivo si è deciso di sostituire *res3* con *res4*

Test Aggiuntivi

Test	**Descrizione**
fibonacci	Pass
sampl2	Pass
convertitore	Pass

Chapter 5

Modifiche aggiuntive

Per quanto concerne ulteriori modifiche, nel Lexer abbiamo aggiunto delle parole riservate come:

- daRestituire
- r_[0-9]+

tali keyword, infatti, sono utili per la traduzione del programma in C. In particolare daRestituire si usa nella traduzione in C per gestire i tipi di ritorno multipli, quindi è l'identificatore per una struct quando questa deve essere restituita in una funzione con più tipi di ritorno. La regex r_[0-9]+ si usa per tenere traccia delle struct dopo una funcall. Infatti il numero dopo r_ indica il numero di chiamate effettuate.

Inoltre nel nostro compilatore è stato anche eseguito un controllo relativo ai return quando si ha un if-else. Come si può vedere nell'esempio di seguito, sebbene non ci sia un return statement alla fine, il codice viene compilato poichè sia il body dell'if che il body dell'else hanno un return statement.

```
func funzione(a:integer)->integer :
    if a>8 then
        return 8;
    else
        return 10;
    endif;
endfunc
```

Inoltre, nella repository è presente una branch chiamata "OrderOfVariable" [1] nella quale è presente una prima versione del compilatore (incompleta) in cui si poteva usare una variabile se e solo se questa era stata dichiarata nelle linee precedenti di codice. Ciò è stato fatto grazie alla classe DefaultMutableTreeNode che è stata anche usata per produrre l'albero sintattico. Nel metodo visit dello ScopeCheckingVisitor si iterava su tutti i figli di un body e si visitava l'albero al contrario,

[1] https://shorturl.at/nxyBV

in questo modo se, in uno statement, si usava una variabile che non era presente nella tabella dei simboli si lanciava un'eccezione. Nella repository è anche presente una branch chiamata "CodeGeneratorVisitor" nella quale è presente una versione del compilatore (anche questa incompleta) che consentiva di usare funzioni con multipli tipi di ritorno anche nei parametri di altre funzioni.

Tuttavia, tali versioni sono state state abbandonate e lasciate incomplete perchè non compatibili con il linguaggio Toy2 con le specifiche del 2 gennaio 2024.