UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Software Engennering and IT Management

COMPILATORI
Documentazione per NewLang

Studentesse:

Mariarosaria Esposito
0522501095

Francesca Perillo
0522501096

Docente del corso:

Gennaro Costagliola

A.A. 2022/2023

Capitolo 1

Introduzione

Nel presente documento verranno elencate tutte le regole di tipo del linguaggio NewLang. Questo documento è di supporto alla documentazione fornita nel corso di Compilatori A.A. 2022/2023. Nel **Capitolo 2.** verranno riportate le tabelle di compatibilità per gli operatori binari e unari. Nel **Capitolo 3.** verranno introdotte le regole di inferenza, con relata spiegazione per quelle aggiuntive a quelle fornitoci dal docente.

N.B. All'interno della quinta esercitazione è presente un file readme.md dove vengono evidenziate:

- una piccola panoramica della struttura del progetto;
- i cambiamenti alla grammatica;
- introduzione alle classi main;
- i file di test;
- i lessemi riservati.

Nel presente documento verranno approfonditi alcuni aspetti della struttura del progetto, specificando il contenuto della cartella */utility/*.. per una completa comprensione della gestione dello scoping e dell'implementazione delle regole di tipo.

Capitolo 2

Implementazione scoping e regole di tipo

2.1 src/visitor/utility

La gestione della tabella dei simboli è gestita dalla classe **SymbolTable** e vede l'utilizzo di un HashMap, dove la chiave è il nome della tabelle stessa; mentre il valore corrisponde al tipo **Element**. Quest'ultimo può a sua volta essere un metodo o una variabile, a seconda dell'oggetto che viene inserito nella tabella dei simboli. Per gestire e identificare il tipo di elemento aggiunto abbiamo nello specifico due classi:

- **ElementMethod**, all'interno della quale viene definito il costruttore per un generico metodo. Abbiamo quindi l'id del metodo, il nome dei parametri con annessi tipi, le modalità del parametro (ricordiamo che un parametro in NewLang può essere anche *out*) e il tipo di ritorno;
- **ElementVar**, all'interno della quale viene definito il costruttore di un generica variabile. Abbiamo quindi il nome e il tipo della variabile. E' presente anche un tipo boolean *isOut* che evidenzia se l'elemento var che si sta considerando sia o meno di out.

Entrambe estendono Element.

Il type environment è invece gestito dalla classe **TypeEnvironment**, che fa uso di un da uno *Stack* di *SymbolTable*. Questa classe contiene metodi per la gestione dello scope:

- *newScope()* - viene aggiunta una nuova tabella dei simboli al Type Environment con l'ausilio del metodo push;
- *getScope()* - viene ritornata l'ultima tabella dei simboli (la più vicina);
- *removeScope()* - viene rimosso lo scope con l'ausilio del metodo pop;
- *addElement(Element e)* - viene aggiunto un elemento nello scope corrente, a seconda che questo sia una variabile o un metodo;
- *lookup(String name)* $\rightarrow$ *Element* - metodo che fa il lookup di un elemento con nome *name* e, nel caso in cui lo trova, ritorna l'*Element* corrispondente;
- *lookupBool(String name)* $\rightarrow$ *Boolean* - metodo che fa il lookup di un elemento con nome *name* e, nel caso in cui lo trova, ritorna true. Ritorna false altrimenti;

Occorre notare che *NewLang* permette l'utilizzo di una *variabile* e di una *funzione* anche prima della loro dichiarazione. Per permettere ciò sono state rispettivamente utilizzate due diverse strutture dati:

- una *HashMap<String,String>*, nel caso di una variabile. La chiave è il nome della variabile stessa; mentre il valore è il suo Tipo.

- un *ArrayList<String>*, nel caso di una funzione. La stringa rappresenta il nome della funzione.

Nel momento in cui una variabile viene utilizzata viene fatto un primo check facendo la *.lookup()*, se non è stata ancora dichiarata allora viene aggiunta all'*HashMap<String,String>* e viene rimossa solo nel momento in cui viene effettivamente dichiarata. Per verificare che tutte le variabili utilizzate siano effettivamente state anche dichiarate, un ultimo controllo viene effettuato alla fine dell'esecuzione. Se l'*HashMap<String,String>* allora vuol dire che tutte le variabili sono state dichiarate; altrimenti l'esecuzione terminerà con un errore. Per quanto invece riguarda le funzioni, la gestione è simile.

Inoltre, per la gestione della tabella di compatibilità è stata implementata una apposita classe java **CompatibilityTables**. Questa contiene due oggetti *enum*:

- un primo che identifica tutti i tipi che uno (o due) operatori possono avere. Tra questi ritroviamo *integer*, *real*, *bool*, *string* e un tipo *erroType* per la gestione dei messaggi onError da parte del **VisitorSemantic**;
- un secondo che identifica il tipo di operazione. Queste verranno evidenziate nel dettaglio in seguito.

Di seguito vengono riportate le tabelle di compatibilità degli operatori binari 2.1 e unari 2.2. I primi vengono identificati come:

- **aritmetical operation** - rappresentano le operazioni *plus*, *minus*, *times* e *pow*;
- **div** - rappresenta la divisione binaria;
- **string concat** - rappresenta la concatenazione fra stringhe;
- **logical operator** - che rappresenta le operazioni logiche per *and* e *or*;
- **comparison operator** - che rappresenta le operazioni di confronto *lt*, *le*, *eq*, *ne*, *gt*, *ge*.

Per gli operatori unari si seguono le direttive delle lezioni. Per completezza è riportata la tabella.

Kind	op1	op2	risultato
arithmetic operation	integer	integer	integer
	integer	real (float)	real (float)
	real (float)	integer	real (float)
	real (float)	real (float)	real (float)
div	integer	integer	real (float)
	integer	real (float)	real (float)
	real (float)	integer	real (float)
	real (float)	real (float)	real (float)
logical operator	bool	bool	bool
string concat	string	string	string
	string	integer	string
	string	real	string
	string	character	string
	integer	string	string
	real	string	string
	character	string	string
comparison operation	integer	integer	bool
	integer	real (float)	bool
	real (float)	integer	bool
	real (float)	real (float)	bool

Tabella 2.1: Operazioni binarie (optype2).

Kind	op1	risultato
unminus	integer real (float)	integer real (float)
not	bool	bool

Tabella 2.2: Operazioni unarie (optype1).

Capitolo 3

Regole di inferenza

Tutte le regole di inferenza fornite dal docente sono state implementate. Di seguito vengono riportate le regole di inferenza utilizzate.

3.1 Identifier

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

3.2 Costanti

$$\Gamma \vdash int\_const : integer$$

$$\Gamma \vdash string\_const : string$$

$$\Gamma \vdash true : boolean$$

$$\Gamma \vdash true : boolean$$

3.3 Lista di istruzioni

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

3.4 Chiamata a funzione con o senza tipo di ritorno (espressione o istruzione) senza controllo del parametro out

$$\frac{\Gamma \vdash f : \tau_1 x ... x \tau_n \implies \tau \Gamma \vdash stmt_2 : notype}{\Gamma \vdash stmt_1; stmt_2 : notype}$$

3.5 Assegnazione

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

3.6 Blocco dichiarazione-istruzione

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

3.7 Istruzione while

$$\frac{\Gamma \vdash e : boolean \quad \Gamma \vdash body : notype}{\Gamma \vdash \textbf{while} e \textbf{loop} body : notype}$$

3.8 Istruzione for

$$\frac{\Gamma \vdash e_1 : int\_const \quad \Gamma \vdash e_2 : int\_const \quad \Gamma[id \mapsto integer] \vdash body : notype}{\Gamma \vdash \textbf{for} \quad id << e_1 \quad \textbf{to} \quad e_2 \quad \textbf{loop} \quad body : notype}$$

3.9 if-then

$$\frac{\Gamma \vdash e : boolean \quad \Gamma \vdash body : notype}{\Gamma \vdash \textbf{if} \quad e \quad \textbf{then} \quad body : notype}$$

3.10 Operatori unari

$$\frac{\Gamma \vdash e : \tau_1 \quad optype1(op_1, \tau_1) = \tau}{\Gamma \vdash op1 \quad e : \tau}$$

3.11 Operatori binari

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

3.12 Return

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \textbf{return} \quad e : \tau}$$