Relazione Progetto LC1 - Gruppo 12 - parte 3

Collevati Michele Procentese Lorenzo
Riccio Francesco

9 aprile 2016

Indice

1 Descrizione generale della soluzione realizzata

A eccezione di determinate cose illustrate nei seguenti paragrafi, abbiamo rispettato la consegna e le specifiche ufficiali del linguaggio di programmazione *Blue*.

Abbiamo inizialmente definito la grammatica del nostro linguaggio (che abbiamo denominato *Blu*) in un file *BNFC*, dopodichè abbiamo usato i seguenti file generati a partire da quest'ultimo (il lexer, il parser, la sintassi astratta e la monade che implementa l'errore) adattandoli alle nostre esigenze, e abbiamo scritto un typechecker e un sistema di pretty printing.

1.1 Assunzioni e scelte fatte

- I commenti sono solo inline e iniziano con `--`

- Un programma è costituito da 3 blocchi, nessuno dei quali deve necessariamente essere dichiarato, e devono apparire nel seguente ordine
  - blocco dichiarazioni costanti
  - blocco dichiarazioni variabili
  - blocco dichiarazioni funzioni
- Il corpo di una funzione è costituito dagli stessi blocchi elencati nel punto precedente, oltre a un successivo blocco che è introdotto da `do` che contiene gli statement
- A certi elementi della sintassi di *Blue* abbiamo imposto la presenza di terminatori. In particolare
  - La chiamata a funzione, il *break*, il *continue*, il *goto* e l'assegnamento sono terminati da `;`
  - La dichiarazione del blocco delle costanti (che in *Blue* inizia con `const`) è terminata da `end const`
  - La dichiarazione del blocco delle variabili (che in *Blue* inizia con `var`) è terminata da `end var`
  - La dichiarazione del blocco delle funzioni (che in *Blue* inizia con `routines`) è terminata da `end routines`
  - La dichiarazione di una funzione è terminata da `end` seguito dal nome della stessa
  - Un costrutto while, for, if termina con, rispettivamente, `end while`, `end for`, `end if`
- Utilizzo della keyword `label` che precede ogni dichiarazione di label per il *goto*
- Uso dei costrutti per assegnamento con operatore. Dei seguenti specifichiamo il significato
  - `^:=` assegnamento con elevamento a potenza
  - `&:=` assegnamento con *and* logico
  - `|:=` assegnamento con *or* logico
- Per dichiarare un puntatore si usa il costrutto `Pt <type>`.
- Abbiamo scelto di usare $ come operatore di dereferenziazione di un puntatore (nel linguaggio *C* è `*`).

- In un singolo comando di dichiarazione di variabile non è possibile dichiararne più di una.
- Sono ammessi solo i seguenti tipi di assegnamento
  - Una l-espressione le a sinistra dell'operatore e una r-espressione re a destra, a condizione che qualora re sia una chiamata a funzione, essa abbia esattamente 1 parametro di output
  - n l-espressioni a sinistra dell'operatore e chiamata a una qualche funzione f a destra, a condizione che f abbia esattamente n parametri di output
- la modalità di passaggio dei parametri di default è quella per valore, perciò essa non va specificata nella dichiarazione dei parametri formali di una funzione
- In uno scope envinronment sono presenti 3 categorie di identificatori: qualli di variabili/costanti, quelli di funzione, quelli di label. All'interno di ciascuna di tali categorie non ci devono essere omonimie, ma possono esserci tra elementi di categorie diverse.
- Non è permessa la mutua ricorsione nel corpo delle funzioni.
- Si fornisce la possibilità di dichiarare la dimensione dell'array.

1.2 Tecniche impiegate

Siamo partiti da una grammatica per bnfc. Dopo la compilazione abbiamo modificato i file generati del lexer, del parser e dell'albero di sintassi astratta. Abbiamo scritto il prettyPrinter usando il modulo Text.PrettyPrint. Per la maggior parte delle tecniche impiegate sono state prese del libro: "Aarne Ranta, Implementing Programming Languages. An Introduction to Compilers and Interpreters".

1.3 Cosa non è stato fatto

- Non è possibile per il sistema di stampa degli errori stabilire la posizione nel sorgente di un dato astratto di tipo `ArrayElems` (una rappresentazione dell'array)
- Non vengono effettuati i controlli di semantica statica relativi alla modalità di passaggio dei parametri alle funzioni

1.4 Errori commessi e soluzioni adottate

- Sono stati commessi degli errori nella definizione della grammatica *BNFC* (file *Blu.cf*), che sono però stati corretti nella successiva fase di adattamento (attraverso modifica) dei sorgenti prodotti da *BNFC*.

2 Type System

2.1 Dichiarazioni

2.1.1 Semplice

$$\frac{\tau \in \{int, char, real, bool, str, array, pt\}}{\Gamma}$$

2.1.2 Assegnamento

$$\frac{\Gamma \vdash_E e : (\tau) \qquad \tau_1, \tau \in \{int, char, real, bool, str, array, pt\} \qquad \tau_1 \leq \tau}{\Gamma \vdash_D x : \tau_1 := e \therefore (x : \tau_1)}$$

2.1.3 Funzione

$$\frac{\Gamma, \{[ip_i : \tau_{ii}], [op_i : \tau_{o1} \ldots op_{oi}] \in basicType, f([\tau_{ii}]) \rightarrow ([\tau_{oi}])\} \vdash_S body}{\Gamma \vdash_D f([\tau_{ii}]) \rightarrow ([\tau_{oi}])\ is \text{ const vars funsc̣o } body \therefore (f : ([\tau_i]) \rightarrow [\tau_o])}$$

2.2 Body

$$\frac{\Gamma \vdash_{RE} a : (\tau) \qquad \tau \in \{int, char, float, bool, str\}}{\Gamma \vdash_{RE} a : (\tau)}$$

2.3 R-Expr

2.3.1 Letterali

$$\frac{\tau \in \{int, char, real, bool, str, array, pt\}}{\Gamma \vdash_{RE} ident : \tau}$$

2.3.2 Array di letterali

$$\frac{\Gamma \vdash [e : \tau] \qquad \tau = max([\tau])}{\Gamma \vdash_{RE} [e] : Array(\tau)}$$

2.3.3 Operazione relazionale

$$\frac{\Gamma \vdash e_1 : \tau_1, e_2 : \tau_2 \qquad relOp \in \{=, <>, <, <=, >, >=\} \qquad max(\tau_1, \tau_2) \neq Nothing}{\Gamma \vdash_{RE} e_1\ relOp\ e_2 : bool}$$

2.3.4 Operazione booleana

$$\frac{\Gamma \vdash e_1 : bool, e_2 : bool \qquad boolOp \in \{\&\&, ||\}}{\Gamma \vdash_{RE} e_1\ boolOp\ e_2 : bool}$$

2.3.5 Operazione aritmetica

$$\frac{\Gamma \vdash e_1 : \tau_1, e_2 : \tau_2 \qquad arithOp \in \{+, -, *, /\} \qquad \tau = max(\tau_1, \tau_2) < Real}{\Gamma \vdash_{RE} e_1\ arithOp\ e_2 : \tau}$$

2.3.6 Operazione modulo

$$\frac{\Gamma \vdash e_1 : \tau_1, e_2 : \tau_2 \qquad \tau = max(\tau_1, \tau_2) < Int}{\Gamma \vdash_{RE} e_1\ \%\ e_2 : \tau}$$

2.3.7 Operazione aritmetica unaria

$$\frac{\Gamma \vdash e : \tau \qquad \tau < Real}{\Gamma \vdash_{RE} +e : \tau,\ -e : \tau}$$

2.3.8 Operazione booleana unaria

$$\frac{\Gamma \vdash e : bool}{\Gamma \vdash_{RE} !e : bool}$$

2.3.9 RE a LE

$$\frac{\Gamma \vdash_{LE} e : \tau}{\Gamma \vdash_{RE} e : \tau}$$

2.3.10 Chiamata di funzione

$$\frac{\Gamma \vdash_{RE} f([\tau_i]) \rightarrow ([\tau_o]) \qquad \Gamma \vdash_{RE} [\alpha : \tau_i]}{\Gamma \vdash_{RE} f([\alpha]) : ([\tau_o])}$$

2.3.11 Indirizzo di

$$\frac{\Gamma \vdash_{LE} e : \tau}{\Gamma \vdash_{RE} \&e : (\tau)}$$

2.4 L-Expr

2.4.1 Id

$$\frac{}{\Gamma, id : \tau \vdash_{LE} id : \tau}$$

2.4.2 Elemento di array

$$\frac{\Gamma \vdash_{LE} a : Array(\tau) \qquad \Gamma \vdash_{RE} i : (\tau') \qquad \tau' \leq Int}{\Gamma \vdash_{LE} a[i] : \tau}$$

2.4.3 Dereferenziazione

$$\frac{\Gamma \vdash_{RE} e : \&(\tau)}{\Gamma \vdash_{LE} \$e : \tau}$$

2.5 Statements

2.5.1 Assegnamento

$$\frac{\Gamma \vdash_{LE} [e : \tau_l] \qquad \Gamma \vdash_{LE} [a : \tau_r] \qquad \tau_l \geq \tau_r}{\Gamma \vdash_{S} [e] := [a]}$$

2.5.2 Assegnamento con operazione aritmetica

$$\frac{\Gamma \vdash_{LE} e_1 : \tau_1, e_2 : \tau_2 \qquad op \in \{*, +, /, -\} \qquad \tau_1 \geq \tau_2}{\Gamma \vdash_{S} e_1 op := e_2}$$

2.5.3 Assegnamento con operazione booleana

$$\frac{\Gamma \vdash_{LE} e_1 : bool, e_2 : bool \qquad op \in \{\&, |\}}{\Gamma \vdash_{S} e_1 op := e_2}$$

2.5.4 Chiamata di funzione

$$\frac{\Gamma \vdash_{LE} f([e : \tau_{li}]) \rightarrow () \qquad \Gamma \vdash_{RE} [\alpha : \tau_{ri}] \qquad \tau_{li} \geq \tau_{ri}}{\Gamma \vdash_{S} f([\alpha])}$$

2.5.5 While-Do

$$\frac{\Gamma \vdash_{RE} e : bool \qquad \Gamma \vdash_S [stmt]}{\Gamma \vdash_S while\ (e)\ do\ [stm]\ end\ while}$$

2.5.6 Do-While

$$\frac{\Gamma \vdash_{RE} e : bool \qquad \Gamma \vdash_S [stmt]}{\Gamma \vdash_S do\ [stm]\ while\ (e)}$$

2.5.7 For

$$\frac{\Gamma \vdash_{LE} init : \tau \qquad \Gamma \vdash_{RE} guard : bool \qquad \Gamma \vdash_{LE} stop : \tau \qquad \Gamma \vdash_S [stmt]}{\Gamma \vdash_S for\ (init,\ guard,\ stop)\ do\ [stmt]\ end\ for}$$

2.5.8 If {[ElseIF]} {Else}

$$\frac{\Gamma \vdash_{RE} e : bool \qquad \Gamma \vdash_S [stmt]}{\Gamma \vdash_S if\ (e)\ do\ [stmt]\ end\ if}$$

$$\frac{\Gamma \vdash_{RE} e : bool \qquad \Gamma \vdash_S [stmt]}{\Gamma \vdash_S elseif\ (e)\ do\ [stmt]} (se\ dopo\ un\ if,\ o\ un\ elseif)$$

$$\frac{\Gamma \vdash_S [stmt]}{\Gamma \vdash_S else\ (e)\ do\ [stmt]} (se\ dopo\ un\ if,\ o\ un\ elseif)$$

2.5.9 Label

$$\frac{\Gamma \vdash l : Label \notin \Gamma}{\Gamma \vdash_S label\ l :}$$

2.5.10 Label

$$\frac{\Gamma \vdash_{Label} l}{\Gamma \vdash_S goto\ l}$$

2.5.11 Break

$$\frac{}{\Gamma \vdash_S break} (se\ dentro\ un\ ciclo)$$

2.6 Dichiarazioni

2.6.1 Semplice

$$\frac{\tau \in \{int, real, bool, string, char, array, pt\}}{\Gamma \vdash_D x : \tau \therefore (x : \tau)}$$

2.6.2 Assegnamento

$$\frac{\Gamma \vdash_E e : \tau \qquad \tau_1, \tau \in \{int, real, bool, string, char, array, pt\} \qquad \tau_1 > \tau}{\Gamma \vdash_D x : \tau := e \therefore (x : \tau_1)}$$

2.6.3 Funione

$$\frac{\Gamma, [ip : \tau_i], [op : \tau_o], f([tau_i) \to ([\tau_o]) \vdash_S body}{\Gamma \vdash_D f([ip]) \to ([op]) body \therefore (f : ([\tau_i]) \to ([\tau_o]))}$$

2.7 Corpo delle funzioni

$$\frac{\Gamma \vdash_D consts \therefore F}{\Gamma \vdash_D consts}$$

$$\frac{\Gamma \vdash_D vars \therefore F}{\Gamma \vdash_D vars}$$

$$\frac{\Gamma \vdash_D function(consts\ vars\ functions\ [stmt]) \therefore F}{\Gamma \vdash_D function}$$

3 File sorgenti, compilazione ed esecuzione

3.1 File sorgenti

- *Blu.cf*: file contenente la grammatica bnfc
- *LexBlu.hs*: il lexer
- *ParBlu.hs*: il parser
- *Abs.hs*: l'albero di sintassi astratta
- *ErrM.hs*: la monade Error
- *TestBlu.hs*: il file contentente il main

- *TypeChecker.hs*: l'implementazione del file system
- il *Makefile*: standard mkefile

3.2 Istruzioni compilazione, esecuzione ed effettuazione test

Si fornisce un makefile con i seguenti comandi:

- *make build*: genera il file eseguibile per eseguire i test
- *make demo*: lancia la batteria dei test
- *make clean*: elimina i file generati durante la compilazione eccetto l'eseguibile