Progetto di Linguaggi e Compilatori 1 – Parte 3
A.A 2015-16

GRUPPO 7

Graca Sanela 114201
Markovski Vlatko 112122
Michelli Luca 109532

Il lexer ed il parser sono stati generati tramite il BNFC.

Nel linguaggio costruito a partire dalla stessa sintassi concreta di Ruby abbiamo fatto delle scelte in contrasto con quelle del linguaggio. Forniamo le nostre variazioni e le relative motivazioni:
- **Variabili**: in Ruby le variabili non hanno il tipo esplicito. Nella nostra sintassi le variabili vengono categorizzate in variabili di istanza e variabili locali. Le variabili di istanza vengono dichiarate con '@nomevariabile' e vengono definite al di fuori delle funzioni, come previsto da Ruby. In aggiunta possono essere definite nei blocchi. Le variabili locali sono quelle che vengono usate solo nei corpi delle funzioni e esistono solo all'interno di esse.
- **Blocchi**: il blocco in Ruby è un modo per raggruppare gli statement e viene invocato da una funzione. Per soddisfare le richieste del testo, abbiamo fatto in modo che in un blocco si possano dichiarare sia le variabili di istanza che le funzioni, e in aggiunta anche altri blocchi. Siccome i blocchi in Ruby hanno un nome (Ident) per poterli invocare dal metodo, abbiamo deciso di mantenere questa sintassi (nonostante il blocco non venga invocato) seguito dalle parentesi grafe che racchiudono il blocco.
- **Funzioni**: Ruby ha soltanto i metodi (funzioni) contententi le espressioni che restituiscono un valore e vengono definite con la keyword 'def'. La nostra modifica delle funzioni consiste nel fatto che le funzioni possono sia restituire un valore che non restituire nulla (in questo caso noi le consideriamo procedure, ma non le distinguiamo dalle funzioni stesse). Per questo motivo abbiamo aggiunto un identificatore di tipo nella definizione della funzione, che in caso di funzioni senza ritorno, è il tipo 'nil' (corrispondente a 'void'). Inoltre, in caso di una funzione senza gli argomenti devono essere presenti le parentesi '()', che in Ruby non è richiesto.
- **Parametri formali delle funzioni**: in Ruby i parametri formali non vengono dichiarati con i tipi espliciti e le modalità di passaggio, la cosa che noi abbiamo aggiunto come richiesto dal testo.
- **Tipi**: i tipi ammessi sono quelli standard (interi, booleani, real (double), caratteri, stringhe, array e puntatori). Per i **puntatori** abbiamo usato una sintassi simile a C e inoltre abbiamo aggiunto un altro tipo **range**, che in Ruby è una classe che permette di stabilire un intervallo tra due interi inclusi separati da due punti (int .. int). Il range ad esempio viene usato negli array, che possono essere definiti sia con l'elenco degli elementi (es. a=[1,2,3]) oppure specificando l'intervallo (es. a=[1..3]). In array è comunque possibile usare qualsiasi tipo di elementi, anche misto (es. a=[1,2,"stringa",'c',3]. In aggiunta, il tipo 'interi' è compatibile con il tipo 'real', il che non vale per il viceversa.
- Nei comandi per il controllo di sequenza semplici, if-else ed elseif, come in quelli di iterazione indeterminata, il ciclo for ed il ciclo while, abbiamo seguito la sintassi concreta di Ruby. Abbiamo aggiunto però le istruzioni **break** e **continue**, su cui eseguiamo il controllo che possano venire eseguite soltanto nel corpo dei cicli.
- **I messaggi di errore** risultano corretti ed efficienti, ma non specificano la riga e la colonna in cui si è manifestato l'errore.

- Nel progetto è incluso il Makefile che genera i file necessari all'esecuzione lanciando 'make', invece lanciando 'make demo' viene eseguito il codice di prova (file Test.hs) che prende in input il file Prova.txt e stampa il risultato in modo ordinato per mezzo del PrittyPrinter.hs

Type system

Dichiarazione di una variabile senza tipo: $\frac{}{\Gamma \vdash x}$

Assegnamento: $\frac{\Gamma \vdash RE : \tau' \qquad \tau' \leq \tau, \qquad \tau \in \{int, real, boolean, char, array, string\}}{\Gamma \vdash \tau\ x = RE : (x : \tau)}$

Dichiarazione di un array : $\frac{\Gamma \vdash ELE : \{ x_1 : \tau_1 , \ldots , x_n : \tau_n \}}{\Gamma \vdash array\ x = ELE : array}$

Puntatore: $\frac{}{\Gamma \vdash \tau * x : (x : \tau)}$

Puntatore con assegnamento: $\frac{\Gamma \vdash RE : \tau' \qquad \tau' = \tau}{\Gamma \vdash \tau * x = \& RE, : (x : \tau)}$

Dichiarazione di funzione:

$$\frac{\Gamma , PAR : \{ p_1 : \tau_1 , \ldots , p_n : \tau_n \} , RETURN : \tau_r \vdash BODY : STM \qquad \tau_r = \tau}{\Gamma \vdash \tau\ f\ PAR\ BODY : (RETURN : \tau_r)}$$

While: $\frac{\Gamma \vdash GUARDIA : bool \quad \Gamma \vdash BODY : STMT}{\Gamma \vdash WHILE\ DO\ GUARDIA\ BODY\ END}$

If: $\frac{\Gamma \vdash GUARDIA : bool \qquad \Gamma \vdash BODY : STMT}{\Gamma \vdash IF\ GUARDIA\ BODY}$

If-else:

$$\frac{\Gamma \vdash GUARDIA : bool \quad \Gamma \vdash BODY1 : STMT \quad \Gamma \vdash BODY2 : STMT}{\Gamma \vdash IF\ GUARDIA\ THEN\ BODY1\ [Elsif]\ ELSE\ BODY2\ END}$$

Elsif: $\frac{\Gamma \vdash GUARDIA : bool \quad \Gamma \vdash BODY : STMT}{ELSIF\ GUARDIA\ THEN\ BODY}$

For: $\frac{\Gamma \vdash VAR : int \qquad RANGE:Range \qquad \Gamma \vdash BODY : STMT}{FOR\ VAR\ IN\ RANGE\ DO\ BODY\ END}$

Range: $\dfrac{\Gamma \vdash VAR1: int \qquad \Gamma \vdash VAR2: int}{VAR1..VAR2}$

Break: $\dfrac{}{\Gamma , \{ break : nil \} \vdash break}$

Continue: $\dfrac{}{\Gamma , \{ continue: nil \} \vdash continue}$

Operazioni aritmetiche unarie prec :

$$\frac{\Gamma \vdash LE: \tau \ \tau \in \{ int , real \} \ OP \in \{+ + , -- , - \}}{\Gamma \vdash LE = OP \ \ LE: \tau}$$

Operazioni aritmetiche unarie post:

$$\frac{\Gamma \vdash LE: \tau \ \tau \in \{ int , real \} \ OP \in \{+ + , -- \}}{\Gamma \vdash LE = LE \ \ OP : \tau}$$

Operazioni aritmetiche binarie :

$$\frac{\Gamma \vdash RE1: \tau \qquad \Gamma \vdash RE2: \tau ' \ \tau \leq \tau ' \quad \tau , \tau ' \in \{ int , real \} \quad OP \in \{+ , - , / , * , ** , \% \}}{\Gamma \vdash RE1 \ OP \ RE2: \tau '}$$

Operazioni unarie booleane: $\dfrac{\Gamma \vdash RE : bool \ OP \in \{ ! \}}{\Gamma \vdash OP \ RE : bool}$

Operazioni binarie booleane: $\dfrac{\Gamma \vdash RE1: bool \ \Gamma \vdash RE2: bool \ OP \in \{ \&\& , || \}}{\Gamma \vdash RE1 \ OP \ RE2: bool}$

Operazioni binarie relazionali:

$$\frac{\Gamma \vdash RE1: \tau \ \Gamma \vdash RE2: \tau ' \ \tau \leq \tau ' \vee \tau ' \leq \tau \ \tau , \tau ' \in \{ int, real, char, string, bool \} \ OP \in \{ ==, !=, <, \leq, > , \geq \}}{\Gamma \vdash RE1 \ OP \ RE2: bool}$$