UNIVERSITÁ DI UDINE

MAGISTRALE INFORMATICA

ANNO ACCADEMICO 2016-2017

Relazione del progetto di

Linguaggi e Compilatori

Studenti:
MARCO DE BORTOLI
MICHELE COLLEVATI
VALENTINO PICOTTI

Mail:
debortoli.marco.1@spes.uniud.it
collevati.michele@spes.uniud.it
picotti.valentino@spes.uniud.it

Indice

Makefile

Il progetto è stato sviluppato con `GHC 8.4.3`, in particolare è stato utilizzato il resolver `lts-12.7` del tool `stack`.

Di seguito i comandi per l'utilizzo del Makefile:

- *make*: per compilare i sorgenti e creare il file eseguibile
- *make demo*: per eseguire la batteria dei test
- *make distclean*: per eliminare i file generati dalla compilazione dei sorgenti

Scelte sintattiche e semantiche

La sintassi concreta segue fedelmente quella di `Ruby`, con l'aggiunta dei tipi nelle dichiarazioni e degli operatori **address-of** (`&`) e **dereferencing** (`*`). Per non causare ambiguità lessicale fra moltiplicazioni, dereferencing, esponenziazione e dichiarazione di puntatore, l'elevamento a potenza è stato introdotto con il simbolo `??` (in `Ruby` è `**`).

Le variabili devono essere inizializzate al momento della dichiarazione. Per dichiarare un tipo puntatore si utilizza il simbolo `*`, e le dichiarazioni vanno lette da destra a sinistra, eccezione fatta per il tipo array che va letto da sinistra a destra nel caso in cui si dichiarino array di array. Per un esempio di dichiarazione di tipo composto si veda Sezione 4.2.

La grammatica contiene un solo shift-reduce, che ora andremo ad analizzare. La situazione che si presenta è la seguente:

LExpr2 => '(' LExpr . ')'

il conflitto sorge dal fatto che vi sono due possibilità: la prima consiste nello shiftare il token ')' ed entrare così nello stato:

LExpr2 => '(' LExpr ')' .

Mentre la seconda possibilità consiste nel ridurre la LExpr ad RExpr utilizzando la seguente regola:

RExpr => LExpr .

Il conflitto viene risolto in modo corretto dato che Happy seleziona automaticamente l'azione di shift, producendo l'effetto desiderato esposto nella prima delle due possibilità.

Il linguaggio offre la possibilità di separare tra loro le istruzioni tramite un `;` oppure andando a capo.

Riguardo ai cicli, si supporta la scrittura inline a patto che il corpo sia preceduto dalla keyword `do` (altrimenti non necessaria) ed ogni istruzione semplice termini con un `;`. Lo stesso discorso vale per le istruzioni condizionali, come `if` ed `unless`, con la differenza che la keyword è `then` invece di `do`. Fanno eccezione il ciclo `loop`, per il quale è strettamente necessario racchiudere il corpo tra parentesi graffe o tra le keyword `do-end`, e il ciclo `do-while`, per il quale è necessario racchiudere il corpo tra le keyword `begin-end`, in coerenza con la sintassi concreta di ruby.

Il costrutto *if-then-else* per la categoria sintattica delle espressioni è stato introdotto con la sintassi concreta di Ruby del *ternary operator* (`guard ? rexpr : rexpr`).

Il comando `try` *Statements* `catch` *Statements* segue la sintassi concreta di Ruby: `begin` *Statements* `rescue` *Statements* `end`.

2.1 Dichiarazioni e visibilità

Il nostro linguaggio permette di dichiarare variabili e funzioni all'interno di un qualsiasi blocco; un nome può essere introdotto nell'ambiente locale in un qualsiasi momento, senza che sia necessario relegare tutte le dichiarazioni in testa al

blocco. I nomi di variabili e funzioni vivono in ambienti distinti, permettendo ad uno stesso nome di denotare sia una variabile che una funzione (il contesto di utilizzo disambiguerà l'oggetto denotato).

I nomi di variabile e funzione seguono le regole di scoping statico. Un nome di variabile è visibile nell'ambiente locale del blocco solo dopo la sua dichiarazione, mentre un nome di funzione risulta visibile nell'intero blocco in cui è dichiarato (quindi anche prima della sua dichiarazione), così da permettere la mutua ricorsione. Un nome di funzione può oscurare anche il nome della funzione nel quale è dichiarato, se vi coincide. Nello scope globale, le dichiarazioni di variabili precedono ogni dichiarazione di funzione, indipendentemente dal punto in cui vengono dichiarate nel sorgente.

Nel costrutto di iterazione determinata `For`, la variabile di iterazione non viene considerata come una dichiarazione implicita locale del corpo, ma deve essere stata precedentemente dichiarata.

Dichiarazione di funzione

In una dichiarazione di funzione, l'unica limitazione posta al tipo di ritorno riguarda gli array, i quali non sono ammessi come tipo di ritorno.

Per quanto riguarda le modalità di passaggio dei parametri, sono ammesse le seguenti modalità, con le rispettive limitazioni:

- Value (`val`): modalità di default. Non è possibile passare per valore oggetti di dimensione variabile (es. stringhe).
- Reference (`ref`): in presenza di un passaggio per riferimento, non è ammessa alcuna coercizione sul parametro attuale, il cui tipo, quindi, deve obbligatoriamente coincidere con quello del parametro formale.
- Result (`res`): non è possibile dichiarare un parametro con modalità result e tipo array perché controllare che ogni cella dell'array venga inizializzata risulterebbe in un controllo di semantica statica gravoso. Inoltre non è ammessa alcuna coercizione sul parametro attuale, il cui tipo deve necessariamente coincidere con quello del formale.
- Value-Result (`valres`): non è ammessa alcuna coercizione sul parametro attuale, il cui tipo deve necessariamente coincidere con quello del formale. Inoltre non è possibile passare per value-result oggetti di dimensione variabile (es. stringhe).
- Const (`const`): si richiede che il parametro attuale sia una espressione dotata di l-valore. Per dati di grandi dimensioni (stringhe ed array) è stato implementato mediante un riferimento, mentre per dati di piccole dimensioni è stato implementato come il passaggio per valore.

Descrizione della soluzione

La soluzione realizzata comprende i seguenti file:

- `Lexer.x` e `Parser.y`
- `Annotations.hs`: contiene il dato Haskell `GenericAnnotated` utilizzato per annotare l'intero albero di sintassi astratta costruito in fase di parsing.
- `SourceLocation.hs`: definisce i tipi di dato che ci permettono di tenere traccia delle posizioni dei costrutti del linguaggio nel codice sorgente.
- `PrettyPrinting.hs` esporta il modulo `Text.PrettyPrint` del pacchetto Haskell `pretty` e definisce la classe `PrPrint`.
- `AST.hs` è dove abbiamo definito la sintassi astratta.
- `Types.hs` contiene i tipi del nostro linguaggio e le funzioni per manipolarli.
- `SymbolTable.hs` definisce le strutture dati utilizzate in fase di controllo della semantica statica e di generazione di codice per tenere traccia dei nomi del programma e le rispettive informazioni.
- `Errors.hs` definisce gli errori di semantica statica.
- `TypeChecker.hs` contiene il modulo responsabile di tutti i controlli di semantica statica.
- `TAC.hs` si occupa della generazione di codice.
- `Compiler.hs` definisce gli entry-point del front-end.
- `Main.hs`

La monade nella quale eseguiamo i controlli statici è definita nel file `TypeChecker.hs` mediante l'uso dei monad-transformers forniti dal pacchetto Haskell `mtl`. Utilizziamo le funzionalità di una State monad (transformer `StateT`) per tener traccia dell'environment e mantenere un log degli errori, mentre utilizziamo una Except monad (transformer `ExceptT`) per segnalare gli errori che poi saranno aggiunti al log. Nel file `Errors.hs` è possibile trovare la lista di tutti gli errori statici che segnaliamo in questa fase della compilazione.

Il type-checker ha il compito di annotare tutte le espressioni con il tipo inferito, generare eventuali coercizioni nell'AST e marcare tutte le espressioni costanti come tali così che, in un secondo momento, possano essere valutate a compile time ove necessario.

Nel file `TAC.hs` è definita la monade utilizzata in generazione di codice. Anche in questo caso sfruttiamo i monad-transformer: la State monad tiene traccia dell'environment corrente e del TAC generato fino ad ora, nonché delle informazioni necessarie a generare identificativi univoci per label e temporanei; la Reader monad (transformer `ReaderT`) simula le label ereditate `BTrue`, `BFalse`, `SNext`, così come presentate a lezione. Nel medesimo file è presente il data-type con cui codifichiamo il TAC.

In fase di generazione di codice, è stata implementata la tecnica "fall-through" per evitare `goto` ridondanti nella valutazione delle espressioni booleane.

Le espressioni booleane nelle guardie vengono valutate con short-cut, come richiesto dalla consegna. In altri contesti, si è scelto di supportare il medesimo comportamento rifacendosi alla soluzione generata per le guardie.

L'albero delle espressioni viene visitato in depth-first order, di conseguenza il codice intermedio generato valuta le espressioni da sinistra a destra. Inoltre, anche l'ordine della valutazione degli argomenti avviene da sinistra a destra.

Infine, oltre all'istruzione TAC `on_exception_goto` *Label*, è stata introdotta l'istruzione `end_catch` per segnalare la mancanza di un exception-handler per le istruzioni a seguire.

Type System

I tipi del linguaggio sono stati organizzati in un ordine parziale con la semantica che $t_1 \sqsubseteq t_2$ se e soltanto se t_1 è coercizzabile a t_2. All'interno del file `Types.hs`, si dichiara il tipo di dato `TypeSpec`, che rappresenta i tipi del nostro linguaggio, e lo si rende istanza della classe `PartialOrd` del pacchetto `lattices`. Oltre ai tipi presenti nel linguaggio, a `TypeSpec` sono stati aggiunti i costruttori:

- `UnTypeable` a indicare che due tipi non sono compatibili
- `UnTyped` a indicare un tipo non noto

così da avere il join-semilattice di Fig. 1.

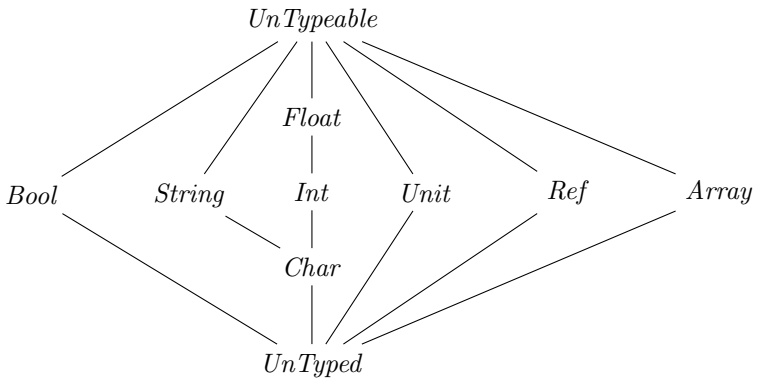

Figura 1: Compatibilità fra tipi

4.1 Inference rules (Type System)

Il nostro environment è composto dall'environment dei nomi di variabile, Γ, e dall'environment dei nomi di funzione, Δ. Entrambi gli environment sono una lista di ambienti locali (che a loro volta sono liste di associazioni nome-tipo), in cui solo l'ambiente locale attuale può essere esteso con nuovi nomi. Nel descrivere le regole di inferenza adottiamo la sintassi di Cardelli e facciamo uso delle seguenti asserzioni, con il rispettivo significato:

$\Gamma \vdash \diamond$	Γ è un environment ben formato per le variabili
$\Delta \vdash \diamond$	Δ è un environment ben formato per le funzioni
$\Gamma \vdash \tau$	τ è un tipo ben formato in Γ
$\Gamma \vdash_{ST} C$	C è un comando ben formato in Γ
$\Gamma \vdash_{RE} E : \tau$	E è una r-expression ben formata di tipo τ in Γ
$\Gamma \vdash_{LE} E : \tau$	E è una l-expression ben formata di tipo τ in Γ
$\Gamma \vdash D \therefore S$	D è una dichiarazione ben formata con signature S in Γ
$\Delta \vdash F \therefore S$	F è una dichiarazione ben formata con signature S in Δ

$$\text{(Env } \varnothing)\ \frac{}{[\varnothing], [\varnothing] \vdash \diamond} \qquad \text{(Env Insert Var)}\ \frac{\Delta, \Gamma \vdash \tau \qquad I \notin dom(\Gamma_1)}{\Delta, \Gamma[[I : \tau :: \Gamma_1] :: \cdots] \vdash \diamond}$$

$$\text{(Env Insert Fun)}\ \frac{\Delta, \Gamma \vdash \tau \qquad I \notin dom(\Delta_1)}{\Delta[[I : \tau :: \Delta_1] :: \cdots], \Gamma \vdash \diamond}$$

$$\text{(Type } Bool)\ \frac{\Delta, \Gamma \vdash \diamond}{\Delta, \Gamma \vdash Bool} \qquad \text{(Type } Int)\ \frac{\Delta, \Gamma \vdash \diamond}{\Delta, \Gamma \vdash Int} \qquad \text{(Type } Float)\ \frac{\Delta, \Gamma \vdash \diamond}{\Delta, \Gamma \vdash Float}$$

$$\text{(Type } Char)\ \frac{\Delta, \Gamma \vdash \diamond}{\Delta, \Gamma \vdash Char} \qquad \text{(Type } String)\ \frac{\Delta, \Gamma \vdash \diamond}{\Delta, \Gamma \vdash String} \qquad \text{(Type } Unit)\ \frac{\Delta, \Gamma \vdash \diamond}{\Delta, \Gamma \vdash Unit}$$

$$\text{(Type } RefType)\ \frac{\Delta, \Gamma \vdash \tau}{\Delta, \Gamma \vdash RefType\ \tau} \qquad \text{(Type } ArrayType)\ \frac{\Delta, \Gamma \vdash \tau \qquad \Delta, \Gamma \vdash R : Int}{\Delta, \Gamma \vdash ArrayType\ R\ \tau}$$

$$\text{(Type } FunType)\ \frac{\Delta, \Gamma \vdash \tau_1, \ldots, \tau_n, \tau_r \qquad \tau_r \neq ArrayType}{\Delta, \Gamma \vdash (\tau_1 \times \cdots \times \tau_n) \to \tau_r}$$

Tabella 1: Environment e tipi ben fondati

$$\text{(LExpr Identifier 1)}\ \frac{\Delta, \Gamma[\cdots :: [\Gamma_1, I : \tau, \Gamma_2] :: \cdots] \vdash \diamond}{\Delta, \Gamma \vdash_{LE} I : \tau}$$

$$\text{(LExpr Identifier 2)}\ \frac{\Delta, \Gamma[\cdots :: [\Gamma_1, I : \tau, \Gamma_2] :: \cdots :: [\Gamma_3, I : \tau', \Gamma_4] :: \cdots] \vdash \diamond}{\Delta, \Gamma \vdash_{LE} I : \tau}$$

$$\text{(LExpr Pre-Incr)}\ \frac{\Delta, \Gamma \vdash_{LE} L : \tau \qquad \tau \in \{Int, Float\}}{\Delta, \Gamma \vdash_{RE} \texttt{++}L : \tau}$$

$$\text{(LExpr ArrayEl)}\ \frac{\Delta, \Gamma \vdash_{LE} L : ArrayType\ \tau \qquad \Delta, \Gamma \vdash_{RE} R : Int}{\Delta, \Gamma \vdash_{LE} L[R] : \tau}$$

$$\text{(LExpr Deref)}\ \frac{\Delta, \Gamma \vdash_{RE} R : RefType\ \tau}{\Delta, \Gamma \vdash_{LE} \texttt{Deref}\ R : \tau}$$

$$\text{(LExpr Brackets)}\ \frac{\Delta, \Gamma \vdash_{LE} L : \tau}{\Delta, \Gamma \vdash_{LE} \texttt{(}L\texttt{)} : \tau}$$

Tabella 2: L-Expression

$$\text{(RExpr LExpr)}\frac{\Delta, \Gamma \vdash_{LE} L : \tau}{\Delta, \Gamma \vdash_{RE} L : \tau} \quad \text{(RExpr Reference)}\frac{\Delta, \Gamma \vdash_{LE} L : \tau}{\Delta, \Gamma \vdash_{RE} \texttt{Ref}\ L : RefType\ \tau}$$

$$\text{(RExpr Literal)}\frac{\tau \in \{Int, Bool, Float, Char, String\}}{\Delta, \Gamma \vdash_{RE} \texttt{Const}\ lit(\tau)}$$

$$\text{(RExpr ArrayLiteral)}\frac{\Delta, \Gamma \vdash_{RE} R_1 : \tau, \ldots, R_n : \tau}{\Delta, \Gamma \vdash_{RE} \texttt{Const}\ [R_1, \ldots, R_n] : ArrayType\ n\ \tau}$$

$$\text{(RExpr Ternary Operator)}\frac{\Delta, \Gamma \vdash_{RE} R : Bool, R_1 : \tau, R_2 : \tau}{\Delta, \Gamma \vdash_{RE} R\ \texttt{?}\ R_1\ \texttt{:}\ R_2 : \tau}$$

$$\text{(RExpr Brackets)}\frac{\Delta, \Gamma \vdash_{RE} R : \tau}{\Delta, \Gamma \vdash_{RE} \texttt{(}R\texttt{)} : \tau} \quad \text{(RExpr Coercion)}\frac{\Delta, \Gamma \vdash_{RE} R : \tau' \qquad \tau' \sqsubseteq \tau}{\Delta, \Gamma \vdash_{RE} \texttt{Coercion}\ \tau\ R : \tau}$$

$$\text{(RExpr Plus)}\frac{\Delta, \Gamma \vdash_{RE} R : \tau \qquad \tau \in \{Int, Float\}}{\Delta, \Gamma \vdash_{RE} \texttt{+}\ R : \tau} \quad \text{(RExpr Minus)}\frac{\Delta, \Gamma \vdash_{RE} R : \tau \qquad \tau \in \{Int, Float\}}{\Delta, \Gamma \vdash_{RE} \texttt{-}\ R : \tau}$$

$$\text{(RExpr Neg)}\frac{\Delta, \Gamma \vdash_{RE} R : Bool}{\Delta, \Gamma \vdash_{RE} \texttt{not}\ R : Bool} \quad \text{(RExpr BitComplement)}\frac{\Delta, \Gamma \vdash_{RE} R : Int}{\Delta, \Gamma \vdash_{RE} \sim R : Int}$$

$$\text{(RExpr FCall)}\frac{\Delta[\cdots :: [\Delta_1, f : (\tau_1 \times \cdots \times \tau_n) \to \tau_r, \Delta_2] :: \cdots], \Gamma \vdash \diamond \qquad \Delta, \Gamma \vdash_{RE} R_1 : \tau_1, \ldots, R_n : \tau_n}{\Delta, \Gamma \vdash_{RE} f(R_1, \ldots, R_n) : \tau_r}$$

Tabella 3: R-Expression

$$\text{(RExpr Sum)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Int \qquad \Delta,\Gamma \vdash_{RE} R_2 : Int}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{+}\ R_2 : Int}$$

$$\text{(RExpr Sum)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Float \qquad \Delta,\Gamma \vdash_{RE} R_2 : Float}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{+}\ R_2 : Float}$$

$$\text{(RExpr Difference)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Int \qquad \Delta,\Gamma \vdash_{RE} R_2 : Int}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{-}\ R_2 : Int}$$

$$\text{(RExpr Difference)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Float \qquad \Delta,\Gamma \vdash_{RE} R_2 : Float}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{-}\ R_2 : Float}$$

$$\text{(RExpr Mul)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Int \qquad \Delta,\Gamma \vdash_{RE} R_2 : Int}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{*}\ R_2 : Int}$$

$$\text{(RExpr Mul)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Float \qquad \Delta,\Gamma \vdash_{RE} R_2 : Float}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{*}\ R_2 : Float}$$

$$\text{(RExpr Div)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Int \qquad \Delta,\Gamma \vdash_{RE} R_2 : Int}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{/}\ R_2 : Int}$$

$$\text{(RExpr Div)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Float \qquad \Delta,\Gamma \vdash_{RE} R_2 : Float}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{/}\ R_2 : Float}$$

$$\text{(RExpr Mod)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Int \qquad \Delta,\Gamma \vdash_{RE} R_2 : Int}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{\%}\ R_2 : Int}$$

$$\text{(RExpr Exp)}\frac{\Delta,\Gamma \vdash_{RE} R_1 : Float \qquad \Delta,\Gamma \vdash_{RE} R_2 : Float}{\Delta,\Gamma \vdash_{RE} R_1 \texttt{??}\ R_2 : Float}$$

Tabella 4: Infix operations 1

$$\text{(RExpr And)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : Bool \quad \Delta, \Gamma \vdash_{RE} R_2 : Bool}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{\&\&}\ R_2 : Bool}$$

$$\text{(RExpr Or)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : Bool \quad \Delta, \Gamma \vdash_{RE} R_2 : Bool}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{||}\ R_2 : Bool}$$

$$\text{(RExpr Eq/Neq)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : \tau \quad \Delta, \Gamma \vdash_{RE} R_2 : \tau \quad \tau \neq ArrayType}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{==}\ R_2 : Bool}$$

$$\text{(RExpr LtE/Lt/GtE/Gt)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : \tau \quad \Delta, \Gamma \vdash_{RE} R_2 : \tau \quad \tau \in \{Int, Float\}}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{<=}\ R_2 : Bool}$$

$$\text{(RExpr BitAnd)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : Int \quad \Delta, \Gamma \vdash_{RE} R_2 : Int}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{\&}\ R_2 : Int}$$

$$\text{(RExpr BitOr)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : Int \quad \Delta, \Gamma \vdash_{RE} R_2 : Int}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{|}\ R_2 : Int}$$

$$\text{(RExpr BitXOr)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : Int \quad \Delta, \Gamma \vdash_{RE} R_2 : Int}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{\^{}}\ R_2 : Int}$$

$$\text{(RExpr LShift)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : Int \quad \Delta, \Gamma \vdash_{RE} R_2 : Int}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{<<}\ R_2 : Int}$$

$$\text{(RExpr RShift)}\ \frac{\Delta, \Gamma \vdash_{RE} R_1 : Int \quad \Delta, \Gamma \vdash_{RE} R_2 : Int}{\Delta, \Gamma \vdash_{RE} R_1 \texttt{>>}\ R_2 : Int}$$

Tabella 5: Infix operations 2

$$\text{(List of Stmt 1)}\ \frac{\Delta \vdash F \therefore (f : A) \qquad \Delta[[f : A :: \Delta_1] :: \cdots], \Gamma \vdash [C_1, \ldots, C_i, C_{i+2}, \ldots, C_n]}{\Delta[\Delta_1 :: \cdots], \Gamma \vdash_{ST} [C_1, \ldots, C_i, F, C_{i+2}, \ldots, C_n]}$$

$$\text{(List of Stmt 2)}\ \frac{\Delta, \Gamma \vdash D \therefore (I : A) \qquad \Delta, \Gamma[[I : A :: \Gamma_1] :: \cdots] \vdash [C_1, \ldots, C_n]}{\Delta, \Gamma[\Gamma_1 :: \cdots] \vdash_{ST} [D, C_1, \ldots, C_n]}$$

$$\text{(List of Stmt 3)}\ \frac{\Delta, \Gamma \vdash C_1 \qquad \Delta, \Gamma \vdash [C_2, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} [C_1, C_2, \ldots, C_n]}$$

$$\text{(List of Stmt 4)}\ \frac{}{\Delta, \Gamma \vdash_{ST} [\,]}$$

Tabella 6: List of Statements

$$\text{(Stmt Assign)}\ \frac{\Delta, \Gamma \vdash_{LE} L : \tau \quad \Delta, \Gamma \vdash_{RE} R : \tau' \quad \tau' \sqsubseteq \tau \quad \tau \notin \{ArrayType\}}{\Delta, \Gamma \vdash_{ST} L \texttt{ = } R}$$

$$\text{(Stmt Assign Op)}\ \frac{\Delta, \Gamma \vdash_{LE} L : \tau \quad \Delta, \Gamma \vdash_{RE} R : \tau'}{\Delta, \Gamma \vdash_{ST} L \texttt{ `op'= } R}$$

with τ and τ' respecting `` `op' `` and Assign type rules.

$$\text{(Stmt RexprStmt)}\ \frac{\Delta, \Gamma \vdash_{RE} R : \tau}{\Delta, \Gamma \vdash_{ST} R} \qquad \text{(Stmt Decl)}\ \frac{\Delta, \Gamma \vdash D \therefore S}{\Delta, \Gamma \vdash_{ST} D \therefore S}$$

$$\text{(Stmt Block)}\ \frac{\varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{begin } [C_1, \ldots, C_n] \texttt{ end}}$$

$$\text{(Stmt try-catch)}\ \frac{\varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1^1, \ldots, C_n^1] \quad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1^2, \ldots, C_m^2]}{\Delta, \Gamma \vdash_{ST} \texttt{begin } [C_1^1, \ldots, C_n^1] \texttt{ rescue } [C_1^2, \ldots, C_m^2] \texttt{ end}}$$

$$\text{(Stmt Break)}\ \frac{}{\Delta, \Gamma \vdash_{ST} \texttt{break}} \qquad \text{(Stmt Continue)}\ \frac{}{\Delta, \Gamma \vdash_{ST} \texttt{next}}$$

$$\text{(Stmt Return 1)}\ \frac{\tau_r = Unit}{\Delta, \Gamma \vdash_{ST} \texttt{return}}$$

with τ_r return type of the function

$$\text{(Stmt Return 2)}\ \frac{\Delta, \Gamma \vdash_{RE} R : \tau \quad \tau \sqsubseteq \tau_r \quad \tau_r \neq Unit}{\Delta, \Gamma \vdash_{ST} \texttt{return } R}$$

with τ_r return type of the function

Tabella 7: Statements

$$\text{(Stmt If)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \quad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n] \quad \Delta, \Gamma \vdash [\texttt{elsif} \ \ldots]}{\Delta, \Gamma \vdash_{ST} \texttt{if}\ R\ \texttt{then}\ [C_1, \ldots, C_n]\ [\texttt{elsif} \ \ldots]\ \texttt{end}}$$

$$\text{(Stmt If-Else)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \quad \varnothing :: \Delta, \varnothing :: \Gamma, \vdash_{ST} [C_1^1, \ldots, C_n^1] \quad \Delta, \Gamma \vdash [\texttt{elsif} \ \ldots] \quad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1^2, \ldots, C_m^2]}{\Delta, \Gamma \vdash_{ST} \texttt{if}\ R\ \texttt{then}\ [C_1^1, \ldots, C_n^1]\ [\texttt{elsif} \ \ldots]\ \texttt{else}\ [C_1^2, \ldots, C_m^2]\ \texttt{end}}$$

$$\text{(Stmt ElseIf)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \quad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{elsif}\ R\ \texttt{then}\ [C_1, \ldots, C_n]}$$

$$\text{(Stmt [ElseIf])}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \quad \Delta, \Gamma \vdash_{ST} \texttt{elsif}_1 \ldots \quad \Delta, \Gamma \vdash_{ST} [\texttt{elsif}_2 \ldots, \ldots, \texttt{elsif}_n \ldots]}{\Delta, \Gamma \vdash_{ST} [\texttt{elsif}_1 \ldots, \texttt{elsif}_2 \ldots, \ldots, \texttt{elsif}_n \ldots]}$$

$$\text{(Stmt Unless)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \quad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{unless}\ R\ \texttt{then}\ [C_1, \ldots, C_n]\ \texttt{end}}$$

$$\text{(Stmt UnlessElse)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \quad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1^1, \ldots, C_n^1] \quad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1^2, \ldots, C_m^2]}{\Delta, \Gamma \vdash_{ST} \texttt{unless}\ R\ \texttt{then}\ [C_1^2, \ldots, C_n^2]\ \texttt{else}\ [C_1^2, \ldots, C_m^2]\ \texttt{end}}$$

Tabella 8: Selection Statements

$$\text{(Stmt While)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \qquad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{while}\ R\ \texttt{do}\ [C_1, \ldots, C_n]\ \texttt{end}}$$

$$\text{(Stmt DoWhile)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \qquad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{begin}\ [C_1, \ldots, C_n]\ \texttt{end while}\ R}$$

$$\text{(Stmt Until)}\ \frac{\Delta, \Gamma \vdash_{RE} R : Bool \qquad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{until}\ R\ \texttt{do}\ [C_1, \ldots, C_n]\ \texttt{end}}$$

$$\text{(Stmt Loop)}\ \frac{\varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{loop do}\ [C_1, \ldots, C_n]\ \texttt{end}}$$

$$\text{(Stmt For)}\ \frac{\Delta, \Gamma \vdash_{RE} x : Int, R_1 : Int, R_2 : Int \qquad \varnothing :: \Delta, \varnothing :: \Gamma \vdash_{ST} [C_1, \ldots, C_n]}{\Delta, \Gamma \vdash_{ST} \texttt{For}\ x\ \texttt{in}\ R_1 \texttt{..} R_2\ \texttt{do}\ [C_1, \ldots, C_n]\ \texttt{end}}$$

Tabella 9: Iteration Statements

$$\text{(Var Decl)}\ \frac{\Delta, \Gamma \vdash_{RE} R : \tau' \qquad \tau \in \{Int, Float, Bool, Char, String, ArrayType, RefType\} \qquad \tau' \sqsubseteq \tau}{\Delta, \Gamma \vdash (\tau\ x\ =\ R) \therefore (x : \tau)}$$

$$\text{(Fun Decl)}\ \frac{\Delta[\varnothing :: [f : (\tau_1 \times \cdots \times \tau_n) \to \tau_r :: \Delta_1] :: \cdots], [\alpha_1 : \tau_1, \ldots, \alpha_n : \tau_n] :: \Gamma \vdash_{ST} [C_1, \ldots, C_m] \qquad \Delta, \Gamma \vdash (\tau_1 \times \cdots \times \tau_n) \to \tau_r}{\Delta[\Delta_1 :: \cdots], \Gamma \vdash \texttt{def}\ \tau_r\ f(\tau_1\ \alpha_1, \ldots, \tau_n\ \alpha_n)\ [C_1, \ldots, C_m]\ \texttt{end} \therefore (f : (\tau_1 \times \cdots \times \tau_n) \to \tau_r)}$$

Tabella 10: Declarations

4.2 Array

Si permette solamente la dichiarazione di array omogenei, cioè contenenti elementi tutti di uno stesso tipo. Inoltre, si deve sempre fornire esplicitamente le dimensioni dell'array, per le quali è richiesto essere note staticamente. Ad esempio:

```
int [10]*[5][2] a
```

è la dichiarazione di un array a di 5 elementi, ciascuno dei quali è un array di 2 puntatori ad array di interi di dimensione 10.

Si permette l'inizializzazione delle variabili di tipo array solo al momento della dichiarazione, a patto che le dimensioni dell'array literal coincidano con quelle dell'array. Ad esempio:

```
char [2][2] a = [['a','b'],['c','d']]
```

è la dichiarazione ed inizializzazione di un array a con un array literal. In questo caso il TypeChecker verifica la correttezza dell'inizializzazione confrontando i tipi e le dimensioni dei due array.