


POLITECNICO DI TORINO

**Laboratorio di Compilatori**  
Corso di Linguaggi e Traduttori

Esercitazione 5



Stefano Scanzio  
mail: [stefano.scanzio@polito.it](mailto:stefano.scanzio@polito.it)  
sito: <http://www.skenz.it/traduttori>

a.a 2009 / 2010

## Controllo dei tipi

---

- Type expressions
- Symbol tables
- Implementazione di un type-checker
  - strutture dati
  - grammatica
  - semantica

<http://www.skenz.it/traduttori> Esercitazione 5 3

## Type Checking

---

- Type Checking è il processo di verifica dei vincoli sui tipi:
  - Può essere eseguito a tempo di compilazione (check statico) o a tempo di esecuzione (check dinamico)
  - Il check dinamico è spesso utilizzato nei linguaggi interpretati, mentre nei linguaggi compilati viene utilizzato un check statico
  - Il checking statico è uno dei principali task semantici eseguiti da un compilatore
- Esempio di check statico
 

```
int a;
float b;

a = 2.5;
b = 2.5 + 'a';
```

<http://www.skenz.it/traduttori> Esercitazione 5 4

## I Tipi

---

- *Tipi base*
  - I linguaggi di programmazione includono tipi base per rappresentare:
    - numeri (int, float), caratteri, booleani
- *Tipi composti*
  - I programmatori hanno bisogno di strutture astratte più complesse dei tipi base definiti nel linguaggio
    - ▲ Al fine di gestire liste, grafi, alberi, tabelle, etc
  - I linguaggi di programmazione permettono di combinare e aggregare tipi base al fine di creare tipi aggregati.
- Un sistema di tipi consiste in un insieme di tipi base e di costruttori di tipo
  - array, function, pointer, product, struct
- Usando i tipi base e i costruttori di tipi ogni dichiarazione, funzione e espressione in un programma può essere rappresentata come una *type expression*

<http://www.skenz.it/traduttori> Esercitazione 5 5

## Type-expressions

---

- In generale, i tipi possono essere:
  - primitivi (int, float, char)
  - costruiti (struct, union)
- Una type-expression è formata da un tipo primitivo, oppure è un costruttore di tipo applicato ad una type-expression.
- I tipi primitivi sono tutti quelli necessari al linguaggio (int, float, char,...) più i due tipi speciali:
  - void : denota l'assenza di un tipo,
  - type\_error : indica un errore rilevato durante il controllo dei tipi.

<http://www.skenz.it/traduttori> Esercitazione 5 6

## Costruttori di tipo

---

- Array: `array( I , T )`
  - I : dimensione dell'array
  - T : type expression
- Puntatori: `pointer( T )`
- Prodotto: `T1 X T2`
- Strutture: `struct( T )`
- Esempi:

Dichiarazione: <pre>char v[10]</pre>	Type expression: <pre>array(10,char)</pre>
<pre>struct {   int i;   char s[5]; }</pre>	<pre>struct(int x array(5,char))</pre>

<http://www.skenz.it/traduttori> Esercitazione 5 7

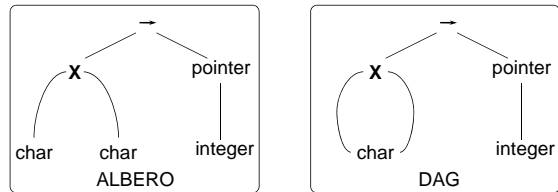
### Costruttori di tipo

- Una funzione mappa un elemento del proprio dominio in un elemento del range.
- Funzioni:  $T_1 \rightarrow T_2$ 
  - $T_1$ : tipo del dominio,
  - $T_2$ : tipo del range.
- La funzione  $int* f(char a, char b)$  viene rappresentata dalla seguente type expression:  
 $(char \times char) \rightarrow pointer(int)$

### Grafo dei tipi

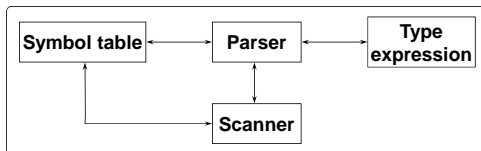
- Un modo efficace di rappresentare le type expression è l'uso di grafi (alberi o DAG).

$(char \times char) \rightarrow pointer(int)$



### Type checker

- Un type-checker è formato da un insieme di moduli interoperanti:
  - scanner: riconosce il lessico,
  - parser: verifica la sintassi ed aggiunge la semantica,
  - gestore di type-expression,
  - gestore di symbol table.



### Symbol table

- Le tabelle dei simboli associano valori a nomi per rendere accessibili informazioni semantiche, legate ad un identificatore, al di fuori del contesto in cui esso è stato dichiarato.
- Le informazioni associate a ciascun nome vengono utilizzate per verificare il corretto uso semantico degli identificatori di un programma.
- Tali informazioni possono essere aggiornate dinamicamente via via che nuove caratteristiche del nome sono dichiarate nel programma o dedotte dal compilatore.

### Symbol table

- Le informazioni che vengono memorizzate all'interno di una tabella dei simboli vengono dette *entry*.
- Ogni operazione di inserimento definisce una *chiave* (solitamente una stringa) tramite la quale poter, successivamente, reperire le informazioni.
- In generale, le informazioni memorizzate in una symbol table non sono omogenee, perciò conviene memorizzare dei puntatori invece delle informazioni stesse.
- Un traduttore utilizza diverse tabelle dei simboli per memorizzare informazioni diverse o appartenenti a contesti diversi.

### Symbol table: implementazione

- Da un punto di vista implementativo una tabella può essere realizzata con una delle seguenti tecniche
  - Liste disordinate
  - Liste ordinate
  - Alberi binari
  - Tabelle hash
  - BTree ...
- La scelta dipende dal numero di simboli da memorizzare, dalle prestazioni che si intendono ottenere, dalla complessità del codice che si intende produrre.

## Symbol table: Implementazione in Java con HashMap

```
import java.util.HashMap;

// Inizializzo la symbol table
HashMap symTable = new HashMap();

// Inserisco delle entry nella symbol table: int a; float b;
symTable.put("a","int");           // "a" chiave, "int" valore associato
symTable.put("b","float");

// Ricavo il valore associato alla chiave "a"
String tipo = (String) symTable.get("a");
System.out.println(tipo);

// Cancello la entry associata alla chiave "a" dalla symbol table
symTable.remove("a");

// Elimino tutte le entry della symbol table
symTable.clear();
```

http://www.skenz.it/traduttori

Esercitazione 5

18

## Type expression

- La rappresentazione naturale delle type expression tramite alberi di tipi può essere trasformata in una rappresentazione interna (Una classe).
- La gestione delle type expression richiede
  - la definizione della struttura dati dei nodi del grafo,
  - la definizione delle primitive per operare sui nodi.
- I nodi devono essere in grado di rappresentare i diversi costruttori di tipo ed i tipi di base.
- Le primitive servono per nascondere la rappresentazione interna dei nodi e consentire all'utente di scrivere del codice il più semplice possibile.

http://www.skenz.it/traduttori

Esercitazione 5

19

## Type expression: implementazione

- Ogni nodo di un grafo dei tipi contiene:
  - un tag, che rappresenta il tipo di nodo;
  - una serie di vari campi dipendenti dal tipo di dato da memorizzare

```
public class te_node {
    public int tag; // BASE, ARRAY, POINTER,...
    public int size; // Numero di el. in array
    public int code; // Tipo base: INT, CHAR, FLOAT,...
    //Solo per struct
    public String name; // Nome della struttura

    // Figlio sinistro e destro del nodo
    private te_node left, right;
}
```

http://www.skenz.it/traduttori

Esercitazione 5

20

## Type expression: implementazione

- Il modulo di gestione delle TE deve offrire le seguenti primitive:

```
public class te_node {
    public int tag;
    ...
    public static te_node te_make_base(int code);
    public static te_node te_make_pointer(te_node base);
    public static te_node te_make_array(int size, te_node base);
    public static te_node te_make_product(te_node l, te_node r);

    //Solo per struct
    public static te_node te_make_name(String name);
    public static void te_cons_struct(te_node str, te_node flds);
    public static te_node te_make_fwdstruct(String name);
    public static te_node te_make_struct(te_node flds, String n);
    public static te_node te_make_function(te_node d, te_node r);
}
```

http://www.skenz.it/traduttori

Esercitazione 5

21

## Type checker: semantica

- Oltre alle funzioni offerte dal modulo di gestione delle type expression, è opportuno fornire alcune primitive per l'accesso alle tabelle dei simboli.
  - Solo nel caso si voglia riconoscere anche il costrutto struct
 

```
int add_type(String name, te_node type);
te_node type_lookup(String nome);
```
  - Sempre
 

```
int add_var(String name, te_node type);
```
- Oltre a semplificare la scrittura delle azioni semantiche, permettono di nascondere i dettagli implementativi delle tabelle.

http://www.skenz.it/traduttori

Esercitazione 5

22

## Type checker: semantica

```
Decl ::= T Vlist;
T ::= TYPE:t; { : RESULT=(te_node)t; : }

Vlist ::= V:t { : RESULT=(te_node)t; : }

|Vlist ' ' { : RESULT=(te_node)stack[top-1]; : } V:t
{ : RESULT=(te_node)t; : }
;

V ::= Ptr ID:a Ary:t { : add_var(a,t);
RESULT=(te_node)stack[top-3]; : }
;

Ptr ::= /* empty */ { : RESULT=(te_node)stack[top]; : }
| Ptr*p { : RESULT=te_make_pointer(p); : }
;

Ary ::= /* empty */ { : RESULT=(te_node)stack[top-1]; : }
| Ary:a [T NUM:b ] { : RESULT=te_make_array(b,a); : }
;
```

http://www.skenz.it/traduttori

Esercitazione 5

23

**Type checker: grammatica completa**

```
S ::= /* empty */
    | S Decl ';'
;
Decl ::= T Vlist
      | TYPEDEF T Vlist
;
T ::= TYPE
   | STRUCT ID '{' SFL '}'
   | STRUCT '{' SFL '}'
   | STRUCT ID
;
SFL ::= Field
     | SFL Field
;
Field ::= T Vlist
;
Vlist ::= V
        | Vlist ',' V
;
V ::= Ptr ID Ary
;
Ptr ::= /* empty */
      | Ptr '*'
;
Ary ::= /* empty */
      | Ary '[' NUM ']'
;
```

<http://www.skenz.it/traduttori>

Esercitazione 5

24