Laboratorio di Compilatori

30 Maggio 2014

Esercizio I

Sviluppare un parser e un type-checker per il linguaggio definito dalla seguente grammatica:

```
S --> id = E ; | if ( E ) S | while ( E ) S | S S
E --> E || E | E && E | E rel E | E + E | (E)
    |   id | true | false
```

Il parser deve costruire alberi sintattici. Il type-checker deve controllare che i tipi delle espressioni siano quelli corretti e in caso negativo deve restituire un messaggio di errore.

Esercizio II

La *liveness analysis* consiste nell' analizzare le variabili usate(registri immaginari) nel codice prodotto e registrare il loro campo di azione, cioè per quali istruzioni il loro valore deve essere *vivo*. A questo scopo come prima cosa si costruisce il grafo del flusso di dati. Ogni istruzione viene messa in un nodo e le due istruzioni successive vengono collegate da un arco diretto. Usando questo grafo si fa la liveness analysis vera e propria, registrando per ogni nodo l'insieme di variabili che sono vive all'entrata ed all'uscita. Queste informazioni servono alla fase successiva, l'**Allocazione dei Registri**. Lalgoritmo che crea il grafo diretto delle istruzioni prende in ingresso una lista di istruzioni e restituisce il grafo corrispondente, ed opera in due fasi. In una prima fase vengono creati gli archi tra i nodi che non rappresentano istruzioni di salto. Nella seconda fase vengono collegati i nodi delle istruzioni di salto con le loro destinazioni.

Esercizio II (ctd.)

Fondamentale per la creazione del grafo è la costruzione dei *Basic Blocks*.
Implementare (in Java) un algoritmo **BasicBlocks** che trova i nodi leader. Essi possono essere:

- Nodi senza archi entranti
- Nodi che sono obbiettivi di salto condizionato o incondizionato.
- Nodi che hanno un arco entrante.