Descrizione del linguaggio Scala40

sssbbf

1 Soluzioni non standard

1.1 Soluzioni non standard - TAC

BOZZA - non riletta

Per la gestione della creazione e dell'ordine di esecuzione delle istruzioni TAC durante la creazione di quest'ultimo, è stata utilizzata la monade `State a`. Questa permette di trattare l'inserimento di nuove istruzioni, la creazione di nuove etichette e di nuovi temporanei, come fossero un cambio di stato in una computazione. La monade `State` utilizza due funzioni per operare sullo stato corrente: `get`, per ottenerlo, e `put` per aggiornarlo.

La monade `State a` utilizzata durante la creazione del Three Address Code opera su stati formati da tuple di quattro elementi:

- un intero che indica l'indice dell'ultimo temporaneo creato;
- un intero che indica l'ultima etichetta creata;
- la lista di istruzioni TAC che compongono il TAC in output;
- una lista di liste di istruzioni TAC. Ogni lista contiene un insieme di istruzioni di una singola funzione.

Lo stato viene modificato nel momento in cui si crea una nuova etichetta e/o un nuovo temporaneo (per evitare duplicazioni tra le istruzioni) (funzioni `newLabel` e `newTemp`, rispettivamente), quando una nuova istruzione TAC di una funzione viene inserita nella lista delle istruzioni di quella funzione (tramite `out`) e, nel momento in cui tutti gli statetement di una funzione sono stati tradotti in istruzione TAC, quando l'insieme delle istruzione di quelle funzioni vengono aggiunte in testa all'insieme delle istruzioni TAC principale (quelle che compongono il TAC in output), tramite la funzione `pushCurrentStream`.

2 Descrizione soluzione

2.1 Descrizione soluzione - TAC

BOZZA

Il Three Address Code (TAC) viene costruito a partire dall'albero annotato risultante dall'analisi di semantica statica (Type Checking). Per la costruzione viene fatto largo uso della monade State, descritta nella sezione precedente. In particolare lo stato ha quattro campi: il numero di temporanei e di etichette finora utilizzati, la lista di istruizioni che compongono il TAC e una lista di liste di istruzioni (utili per la gestione degli scope e delle funzioni innestate). Lo stato, come detto, è stato rinominato in `TacState`.

Nel TAC gli indirizzi utilizzati nelle istruzioni sono:

- Letterali: per valori costanti di tipo base.
- Indirizzi di variabile e di funzioni: hanno la seguente forma,

  `ident@loc`

  dove `ident` è il loro identificatore e `loc` la locazione di dichiarazione.
- Temporanei: utilizzati per identificare espressioni.

Inoltre, nel TAC sono presenti le etichette che vengono utilizzate per indicare una locazione univoca nel flusso di esecuzione. Nel caso delle funzioni, come etichetta, viene utilizzato il loro indirizzo.

La costruzione del TAC parte dalla funzione

```
genProg ::  Program -> Bool -> TacState ()
```

che ha i seguenti compiti:

- Inserire un'etichetta in fondo a tutte le istruzioni (e il relativo `Goto` dopo le dichiarazioni globali) se durante l'analisi di semantica statica non è stato trovato un metodo main da cui far partire la computazione. In alternativa, viene inserita un'istruzione `Goto` all'etichetta del main, se esso è stato trovato;
- iniziare la discesa nell'albero annotato per poter costruire le singole istruzioni, tramite la funzione

  ```
  genDecls ::  [Declaration] -> TacState ()
  ```

La funzione `genDecls` prende in input una lista di dichiarazioni e le scorre producendo le relative istruzioni TAC. Questo avviene tramite la funzione

```
genDecl ::  Declaration -> TacState ()
```

la quale divide le dichiarazioni nei quattro casi possibili:

- Dichiarazione di variabile con assegnamento: viene creato un nuovo indirizzo a partire dall'identificatore della variabile e dalla sua locazione di dichiarazione. Gli viene assegnato un valore tramite la funzione `genExpAssign` che permette di evitare la creazione di temporanei inutili quando ci troviamo nel caso di un'espressione semplice (es. $x = 3$, $*y = x$, $x = y + z$ ecc.).

Tipo	Valore di default
Int	0
Float	0.0
Bool	False
Pointer	Null
Array	-

Table 1: Valori di default per dichiarazione senza assegnamento

- Dichiarazione di variabile: come nel caso precente costruiamo un indirizzo a partire da identificatore e locazione, e gli assegnamo un valore di default () tramite la funzione `buildDefaultValue`.
- Dichiarazione di funzione/procedura: in questo caso viene creata una nuova etichetta per la funzione. Viene chiamata la funzione `genBlock` che (tramite `genStms`) controlla la presenta dell'istruzione `return` (nel caso delle procedure verrà aggiunto un'istruzione di return vuoto) e costruisce tutte le istruzioni TAC relative al corpo della funzione. Al termine di questo processo le istruzioni TAC che sono state create vengono estratte tramite operazione di pop dalla lista di liste di istruzioni dello stato e inserite in testa al codice globale del TAC. Questo per far si che una funzione dichiarata nel corpo di un'altra venga inserita sopra quest'ultima a livello di Three Address Code, e non al suo interno.

  Ogni funzione è costituita da un insieme di statement (contenuti in un blocco). La funzione che si occupa di percorrere tutti gli statement di un blocco è

  ```
  genStms ::  [Stm] -> TacState Bool
  ```

  che ritorna `True` se l'ultimo statement è un return (per evitare che nel TAC compaiano due istruzioni `return` consecutive). Nel contempo vengono esaminati gli statement singolarmente tramite la funzione

  ```
  genStm ::  Stm -> TacState ()
  ```

  Quest'ultima prende in esame tutti i possibili statement:

  - Dichiarazione: viene riutilizzata `genDecl` per la gestione delle dichiarazioni interne al blocco.
  - Blocchi interni: viene riutilizzata la funzione `genBlock`.
  - Assegnamenti: un assegnamento ha la seguente forma:

    ```
    Lexpression = RExpression
    ```

Le `LExpression` comprendono: accessi ad array, variabili puntatori e identificatori di variabile.La loro gestione avviene tramite la funzione `genLexp`. Le `RExpression` oltre che le `LExpression`, comprendono le espressioni.

- Chiamata di procedura: vengono create le istruzioni che indicano i parametri passati alla funzione (tramite la funzione `genParams` che crea istruzioni `Param addr`) e viene creata un'istruzione `Call` del TAC.
- Return: se l'istruzione `return` ritorna un'espressione essa viene gestita tramite `genExp`, che in caso di espressioni complesse (non binaria tra due identificatori o unaria) restituisce un temporaneo.
- If e While: in entrambi i casi la gestione delle etichette viene gestita tramite la funzione

  ```
  genCondition ::  Exp -> Label -> Label -> TacState ()
  ```

  che, presa l'espressione della condizione e le due etichette che indicano dove spostarsi in caso essa sia vera o meno, gestitisce la creazione dei `Goto` ed esegue il controllo del flusso.

La generazione delle istruzioni riguardanti le espressioni viene gestita tramite la funzione

```
genExp ::  Exp -> TacState Addr
```

che ritorna l'indirizzo TAC dell'espressione: un letterale nel caso di espressioni composte da un solo letterale o un nuovo temporaneo nel caso di dereferenziazioni, LExpression, accessi ad array o espressioni simili a quelle descritte nel caso dell'assegnamento (gestite tramite la funzione `genExpAssign`).

Terminate tutte le dichiarazioni presenti nello scope globale, vengono inserite in testa alla lista di istruzioni TAC, tutte le dichiarazioni di variabile globale seguite dall'istruzione `Goto` alla funzione `main` se presente, oppure all'etichetta `l0` in fondo a tutte le istruzioni (evitando così che tutte le altre funzioni vengano eseguite sequenzialmente).