Laboratorio: Attività Preparatoria e Primi Esercizi

Sommario: 15 Marzo, 2019

- Scelta Degli Strumenti: Il Linguaggio di Sviluppo
  - Uso del Linguaggio C in:
    Realizzazione di Sintassi Astratta
  - Meccanismi Necessari e Risorse del C:
    Tipi Unione e Costrutto union del C
  - Studio di Caso:
    Applichiamo alla Sintassi Astratta del λ-Calcolo

Scelta Degli Strumenti: Il Linguaggio di Sviluppo

- **Linguaggio di Sviluppo**.
  Quello in cui scriveremo il nostro Interprete

  - La sua scelta dipende da tanti fattori:
  - Incluso che tale Linguaggio ci sia imposto: OCaml per noi.
  - Espressività.
    Rimane un fattore di primaria importanza e condiziona:
    - I vincoli di uso da imporre sul sistema prodotto
    - La leggibilità del codice
    - La modifica e correggibilità di malfunzionamenti
  - Oggi useremo il Linguaggio C per
    - Realizzare la SA di un LP
    - Questo è un passo cruciale dello Sviluppo di un Interprete
    - Infatti lo AST su cui opera l'interprete è scritto utilizzando la SA

Realizzare la SA di un LP

- Una SA introduce Tipi di Alberi e
- Tipi che sono l'unione di alcuni di tali Tipi;

Esempio

Sintassi Astratta: *Una grammatica su* $\texttt{Tree}^{\texttt{T}}$ *per le Espressioni*

$$\texttt{G}^{\texttt{A}} = (\{\texttt{E}\}, \{\texttt{N}, +, -, *\}, \texttt{E}, \texttt{R}_{\texttt{G}^{\texttt{A}}})$$

$$\texttt{R}_{\texttt{G}^{\texttt{A}}} = \{\texttt{E} ::= [+]\ \texttt{E}\ \texttt{E},\ \texttt{E} ::= [*]\ \texttt{E}\ \texttt{E},\ \texttt{E} = [-]\texttt{E},\ \texttt{E} = [\texttt{N}]\}$$

- Nell'esempio sopra il tipo `E` è un unione di 4 distinti Tipi di Albero;
- I quattro Tipi di Albero sono introdotti in modo anonimo;
- in C, Conviene introdurli con un Tipo esplicito;

Tipi Unione in C

- Tipi che sono l'unione di alcuni di tali Tipi

Esempio

Sintassi Astratta: *Una grammatica su* $\texttt{Tree}^{\texttt{T}}$ *per le Espressioni*

$$\texttt{G}^{\texttt{A}} = (\{\texttt{E A P O K}\}, \{\texttt{N}, +, -, *\}, \texttt{E}, \texttt{R}_{\texttt{G}^{\texttt{A}}})$$

$$\texttt{R}_{\texttt{G}^{\texttt{A}}} = \{\texttt{A} ::= [+]\ \texttt{E}\ \texttt{E},\ \texttt{P} ::= [*]\ \texttt{E}\ \texttt{E},\ \texttt{O} = [-]\texttt{E},\ \texttt{K} = [\texttt{N}]$$
$$\texttt{E} ::= \texttt{A} \mid \texttt{P} \mid \texttt{O} \mid \texttt{K}\}$$

- Nell'esercizio sulla Deriva, abbiamo visto come utilizzare costrutti **struct**, costrutti **pointer** e costrutti **alloc** per definire operazioni in grado di creare tutti gli alberi di Tipo `A`, `P`, `O`, `K` a patto di disporre del (degli alberi di) Tipo `E` utilizzato nella costruzione i tali alberi
- Dobbiamo usare il costrutto **union** per esprimere Tipi unione di Tipi.

Il Costrutto union in C

Presentazione presa da [1]

- La sintassi delle union è come quella delle struct, ma i membri delle union condividono la memoria
- Un tipo union definisce un insieme di valori alternativi che possono essere contenuti in una porzione di memoria condivisa come quella delle struct, ma i membri delle union condividono la memoria
- Il programmatore è responsabile della corretta **interpretazione** dei valori memorizzati
- ... seguono esempi di uso ...(vedi testo in nota, pagg. 360-366)

Aggiungiamo che:

- Possiamo creare valori di tale Tipo usando allocazione statica o come per struct, dinamica utilizzando pointer e m-c-alloc (vedi Listing oneUnion.c)
- Come controllare la corretta **Interpretazione** ?

[1] A. Kelley e I Pohl, C: Didattica e Programmazione , Addison-Wesley, Milano, 1996, I^{a} edizione

Tipi Unione di Tipi: Perchè , Dove, Quando ci servono

- Consideriamo la struttura dei termini definiti da R_{G^A} per il Tipo `E`.
- La definizione di `E` afferma che ogni termine di Tipo `A` è anche termine di `E` e analogamente afferma, per i Tipi `P`, `O`, `K`.
- Se ora esaminiamo la struttura dei termini di Tipo `A` vediamo che questi hanno come sotto-termini una qualunque coppia di termini di `E`.
- Ed è proprio nell'esprimere i termini dei sottoTipi `A` , `P` ed `O` che nasce la necessità di esprimere il Tipo `E` ed esprimerlo come unione di di Tipi.

Esempio

Introduciamo una union per esprimere una struttura a condivisione di memoria adatta ai Termini di tipo `E`.

```
union Estruct{
    Atype a;
    Ptype p;
    Otype o;
    Ktype k;
};
```

- *La definizione assume che* `Atype`, `Ptype`, `Otype` e `Ktype` *siano i nomi usati per i Tipi* `A`, `P`, `O`, `K` *nelle definizioni date in C.*
- *Questa struttura potrà essere allocata staticamente o dinamicamente ed*
- *utilizzata per contenere un Termine di tipo* `E` *inserito selezionando il campo previsto per il sotto Tipo effettivo del Termine scelto.*

- L'esempio mostra la struttura a condivisione di memoria per contenere ogni possibile Termine di Tipo `E`.
- È sufficiente tale struttura per calcolare con i Termini di `E`?

union in C: Controllare l'**Interpretazione** della memoria condivisa.

- È sufficiente tale struttura per calcolare con i Termini di `E`?
- La risposta è: NO
- Infatti. La si supponga sufficiente.
  Si abbia allora, una struttura Estruct contenente un Termine di Tipo `E`.
  Ed ora si vogliano selezionare ed esaminare i due sotto Termini di Tipo `E` del Termine contenuto.
- Come dovremmo procedere?
  Non abbiamo modo di procedere in modo corretto.
  Infatti, come essere certi che il Termine contenuto sia proprio di sotto Tipo `A`, `P`? E se fosse di tipo `k`?
- Dobbiamo introdurre un meccanismo di controllo del Tipo del valore correntemente inserito.

Esempio

Accoppiamo Estruct con un flag che univocamente ricordi il sotto Tipo dell'ultimo Termine inserito.

```
typedef enum {A, P, O, K} EKinds;
struct ETerm{
    EKinds flag;
    union Estruct term;
};
```

- *Interrogandoci sul valore di flag possiamo sapere con certezza quale Tipo di Termine è contenuto.*
- *Ma, ricordiamo che dobbiamo gestire tutto noi.*
- *Il Linguaggio C offre solo i costrutti struct, union, malloc: I tipi Unione li emuliamo.*

Completiamo l'infrastruttura con le definizioni dei Tipi

- Ora siamo in grado di fornire le definizioni di tutti i Tipi che dobbiamo trattare, ovvero: I sotto Tipi `Atype, Ptype, Otype e Ktype`,

Esempio

Introduciamo un Tipo con appropriata struttura struct per ciascun sotto Tipo: Vediamo `Atype`.

```
typedef struct Astruct{;
        struct ETerm * term1;
        struct ETerm * term2;
} Atype, ;
```

- *Notare i Tipi dei sotto Termini indicati come* `struct ETerm*`.
- `struct ETerm*` *ci dice come dovremo costruire tali sotto Termini.*

- e il Tipo unione di Tipi: `Etype`.

Esempio

Finalmente possiamo introdurre la definizione del Tipo `Etype`

```
typedef struct ETerm * Etype;
```

- *E come possiamo vedere il Tipo* `Etype` *è proprio quello dato dall'espressione di tipo* `struct ETerm*` *sopra utilizzata per il Tipo dei componenti dei Termini* `Atype` *(e degli altri, omessi).*

Sintassi Astratta in C

Esercizio (semplificato)

Si consideri la Sintassi Astratta sotto, per il Lambda Calcolo:

`Λ ::= Var | Const | [$] Var Λ | [@] Λ Λ,`
`Var ::= [X] Ide, Const ::= [K] Ide`

dove usiamo $ per l'astrazione, @ per l'applicazione e `K` *per i simboli di costante. Infine* `Ide` *è una lessicale.*
Si chiede di fornire in C strutture, tipi ed operazioni per la costruzione, l'accesso e la presentazione degli AST (Abstract Syntax Tree) dei termini del Linguaggio del Lambda Calcolo.
In particolare, le operazioni richieste devono inludere:

Costruttori: `mkConst, mkAbs`
Predicati di Controllo: `isConst, isAbs`
Selettori di Componente: `getIde, getTem`
Presentatori di Valori:

Esercizio (Completo)

Aggiungere le seguenti operazioni:

Costruttori: `mkVar, mkApp`
Predicati di Controllo: `isVar, isApp`
Selettori di Componente: `getFun, getArg`
Presentatori di Valori: `printAST`

Si mostri poi, come è costruito il termine $[\lambda - ([x], [\lambda - ([y], [@ - ([x], [y])])])$ *e, utilizzando printAST, se ne mostri la presentazione fornita dal sistema definito.*