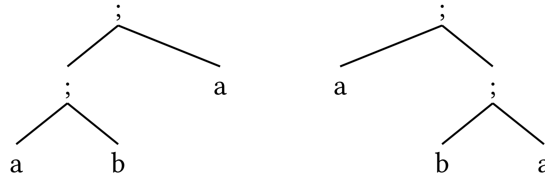
Soluzione Linguaggi Totale

2024-02-06

NOTA: Questa è una soluzione proposta da me, non è detto che sia giusta. Se trovate errori segnalateli o meglio correggeteli direttamente :)

1. La grammatica è ambigua perché ammette due alberi di derivazione per la stringa $a; b; a$. Infatti:



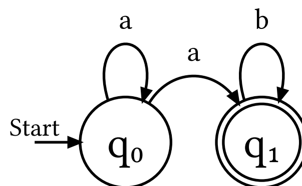
   Per le regole di SOS forse bastano quelle che ha definito il prof sui lucidi:

$$\frac{<\text{c0},\sigma> \underset{c}{\rightarrow} <\text{c0'},\sigma'>}{<\text{c0;c1},\sigma> \underset{c}{\rightarrow} <\text{c0';c1},\sigma'>}$$

$$\frac{<\text{c0},\sigma> \underset{c}{\rightarrow} \sigma'}{<\text{c0;c1},\sigma> \underset{c}{\rightarrow} <\text{c1},\sigma'>}$$

2. Il DFA ottenuto dai ha stati $Q_1 \times Q_2$ e stati finali $F_1 \times F_2$. La funzione di transizione è interessante: $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$. Se si nota bene, la funzione di transizione è l'insieme dei risultati delle due funzioni di transizione degli automi originali. È come se partissimo per entrambi gli automi dal loro stato iniziale e mentre provimo a riconoscere una stringa ci moviamo contemporaneamente su entrambi. La conclusione è che, essendo entrambi DFA e deterministici, anche seguirli in "parallelo" risulterebbe in un automa deterministico, e quindi un DFA. Il linguaggio riconosciuto è l'unico l'inguaggio che riconoscono entrambi, perché per andare in uno stato di errore basta che ci vada solo uno dei due. Quindi il linguaggio riconosciuto è l'intersezione dei linguaggi originali: $L_1 \cap L_2$

3. Costruiamo l'automa:



   Non è deterministico, ma non ci preoccupiamo delle cosa in quanto non è richiesto che sia un DFA. La grammatica è la seguente:

$$S \rightarrow \text{AB}$$
$$A \rightarrow \text{aA} \mid a$$
$$B \rightarrow \text{bB} \mid \varepsilon$$

4. Per costruire il parser $LL(1)$ che riconosce il linguaggio $L = \{a^n c b^n \mid n \geq 1\}$ trovo prima la grammatica:

$$S \rightarrow \text{aSb} \mid \text{acb}$$

   Per renderla $LL(1)$ fattorizzo:

$$S \to \mathrm{aA}$$
$$A \to \mathrm{Sb} \mid \mathrm{cb}$$

Ora elimino la ricorsione sinitra non immediata:

$$S \to \mathrm{aA}$$
$$A \to \mathrm{aAb} \mid \mathrm{cb}$$

Trovo poi i *first* e i *follow*:

	First	**Follow**
S	a	$
A	a, c	b, $

Costruisco la tabella di parsing $LL(1)$:

	a	b	c	$
S	$S \to \mathrm{aA}$			
A	$S \to \mathrm{aAb}$		cb	

Mostriamo ora il funzionamento sull'input *acb*:

acb$	S$
acb$	aA$
cb$	ab$
$	$

5. Quando nomi diversi denotano lo stesso oggetto si parla di *aliasing*. Si può pensare quindi di avere un oggetto in memoria e più variabili che puntano a tale oggeto. La risposta per la domanda *"È possibile inserire nel compilatore di un ipotetico linguaggio un controllo che permetta di identificare tutte le situazioni di aliasin?"* è **dipende**. Infatti dipende dalla libertà che abbiamo nel linguaggio stesso. Ovvero se il linguaggio non ci permette di fare tutto quello che vogliamo (come Rust), allora è possibile creare un compilare che possa controllare l'aliasing. Invece in un linguaggio come C, che permette di fare tutto, non è possibile controllare l'aliasing a tempo di compilazione perché si potrebbe sempre scrivere un indirizzo di memoria esplicito dentro un puntatore e fino al momento dell'esecuzione non si saprebbe a cosa punta.

6. Per fare gli esercizi con il passaggio per nome basta ricopiarsi il codice della funzione con un minimo di accortezze:

```
{
  int x = 2;
  int y = 5;
  int z = 10;
  void pippo (name int v, name int w) {
    int x = 1000;
    w = v;
    v = v + w + (z++) + z;
    z = 1000;
  }
  { int x = 20;
    int y = 50;
    int z = 100;
    //pippo (x, y );
    // Ricopio il codice stando attento
```

```
    int x_int = 1000; // La x è quella interna alla funzione. Gli cambio nome
    y = x;
    v = x + x + (z++) + z; // La z è quella in questo blocco per lo scoping dinamico
    // L'istruzione precedente sarebbe undefined behavior in C. Assumiamo che si
    // valuti strattamente da sinistra a destra
    // v = 20 + 20 + 100 + 101 = 241
    z = 1000;
    write (x, y, z); // 241, 20, 1000
  }
  write (x, y, z)  // 2, 5, 10
}
```

Quindi la stampa finale è: **241, 20, 1000, 2, 5, 10**

7. Non so bene come rappresentare la cosa in typst. Facciamo istruzione per istruzione:

```
class C { C next ; }
C p = new C(); // nuovo OGG ( OGG1 )
// Creo l'oggetto in memoria e gli assegno un lock randominco (parto da 1)
// Poi faccio puntare p all'oggetto e assegno a p.key il valore del lock
// p -> OGG1
// OGG1.lock = 1
// p.key = 1

for ( int i = 0, i < 2, i ++ ){
  // ------ PRIMA ITERAZIONE -------
  {
    C q = new C(); // nuovo OGG2
    // Come sopra, chiamiamo questo oggetto OGG2
    // q -> OGG2
    // OGG2.lock = 2
    // q.key = 2

    q.next = new C(); // nuovo OGG3
    // q.next -> OGG3
    // OGG3.lock = 3
    // (q.next).key = 3

    p.next = q.next ;
    // p.next -> OGG3
    // (p.next).key = 3
  }

  // ------ SECONDA ITERAZIONE -------
  {
    C q = new C(); // nuovo OGG4
    // q -> OGG4
    // OGG4.lock = 4
    // q.key = 4

    q.next = new C(); // nuovo OGG5
    // q.next -> OGG5
    // OGG5.lock = 5
    // (q.next).key = 5

    p.next = q.next ;
    // p.next -> OGG5
    // (p.next).key = 5
```

```
  }
}
// Finito il for saranno rimasti in memoria gli oggetti OGG{1..5} ma solo
// OGG1 e OGG5 saranno raggiungibili rispettivamente da p e p.next
// p -> OGG1
// OGG1.lock = 1
// p.key = 1
// p.next -> OGG5
// (p.next).key = 5
```

8. Dal testo sappiamo che $B <: A$ e $C <: A$. Le scritture e letture su array sono covarianti.
   - I1 ERRATA: Non c'è relazione tra i tipi
   - I2 ERRATA: Come la precedente
   - I3 GIUSTA: `c` è sottotipo di `a` e quindi possiamo fare l'assegnamento
   - I4 GIUSTA: `bb[0]` restituisce un tipo `B` che può essere assegnato ad un tipo `A`
   - I5 GIUSTA: L'array di tipo `B` va bene per un array di tipo `A`
   - I6 ERRATA: L'array di tipo `A` non va bene per un array di tipo `C`
   - I7 GIUSTA: Banale
   - I8 ERRATA: Non possiamo assegnare un array ad un tipo non array
   - I9 GIUSTA: Possiamo assegnare un tipo `C` ad un tipo `A`.
   - I10 ERRATA: Non possiamo usare un oggetto `A` al posto di `B`
   - I11 ERRATA: Non possiamo assegnare un oggetto di tipo `B` ad uno di tipo `C`