Compiladors: Examen de teoria

20 de juny de 2016

Duració de l'examen: 3 hores

Publicació de les notes: 27 de juny
Revisió de notes: 28 de juny (10:00-11:00)

1 Analitzador sintàctic (3.5 punts)

Escriure analitzadors recursius descendents per a les tres gramàtiques següents:

1. $A \rightarrow$ `'+'` AA | `'-'` AA | `'a'`
2. $A \rightarrow$ `'0'` A `'1'` | `'0'` `'1'`
3. $A \rightarrow$ `'a'` (`'0'` | `'1'`)$*$ A `'b'` | ϵ

Podeu suposar que cada gramàtica té la regla $S \rightarrow A\$$, on S és el símbol inicial i `$` és el símbol que indica el final de l'entrada. La funció que correspon al símbol S és la següent:

```
void S() {
  A();
  if (Token != '$') SyntaxError();
}
```

Per a cada gramàtica cal escriure el codi de la funció `A()`. En cas de produir-se un error, cal cridar la funció `SyntaxError()`. Es pot suposar que la variable global `Token` conté el símbol terminal actual de l'entrada. La funció `match(T)` comprova que `Token` és igual a `T` i avança al token següent. En el cas que no sigui igual, genera un error sintàctic.

En el cas que alguna de les gramàtiques no sigui LL(1), caldria retocar-la i fer-la LL(1). Si cal introduir algun nou símbol no terminal, també caldrà escriure el codi de la seva funció.

Resposta:

La segona gramàtica no és LL(1). Es pot fer LL(1) de la manera següent: $A \rightarrow$ `'0'` $A?$ `'1'`.
O també factoritzant:

$$\begin{aligned} A &\rightarrow \text{'0'}\ B \\ B &\rightarrow A\ \text{'1'} \mid \text{'1'} \end{aligned}$$

Els codis de les funcions podrien ser els següents:

(1)

```
void A() {
  if (Token == '+') {
    match('+'); A(); A();
  } else if (Token == '-') {
    match('-'); A(); A();
  } else if (Token == 'a') {
    match('a');
  } else {
    SyntaxError();
  }
}
```

(2)

```
void A() {
  match('0');
  if (Token == '0') A();
  match('1');
}
```

(3)

```
void A() {
  if (Token == 'a') {
    match('a');
    while (Token == '0' or Token == '1') {
      match(Token);
    }
    A();
    match('b');
  }
}
```

Per la segona versió de la gramàtica 2 (factoritzant), tindríem:

```
void A() {
  match('0'); B();
}
```

```
void B() {
  if (Token == '0') {
    A(); match('1');
  } else if (Token == '1') match('1');
  else SyntaxError();
}
```

2 Gramàtiques d'atributs (3 punts)

Considereu la gramàtica d'atributs següent:

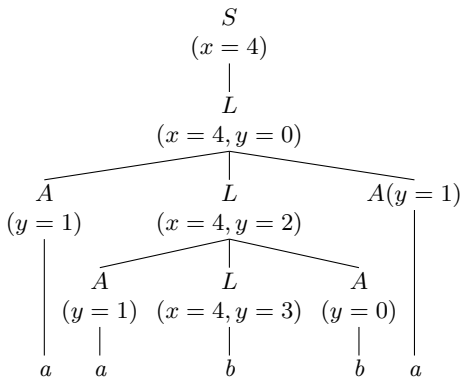
$$
\begin{array}{lcll}
S & \to & L & \{L.y = 0; S.x = L.x\} \\
L & \to & A_1 L_1 A_2 & \{L_1.y = A_1.y + L.y + A_2.y; L.x = L_1.x\} \\
A & \to & a & \{A.y = 1\} \\
A & \to & b & \{A.y = 0\} \\
L & \to & a & \{L.x = L.y\} \\
L & \to & b & \{L.x = L.y + 1\}
\end{array}
$$

1. Dibuxeu l'arbre sintàctic per l'entrada *aabba*, indicant a cada node el valor dels atributs x i y.
2. Descriviu quin és el valor de $S.x$ per a qualsevol entrada en general.
3. Proposeu una gramàtica d'atributs equivalent més senzilla que calculi $S.x$.

Resposta: Aquesta gramàtica genera paraules amb un nombre imparell de símbols. Si representem com $|a|$ el nombre d'as de l'entrada, llavors $S.x$ té el valor següent:

$$
S.x = \begin{cases} |a|-1 & \text{si el símbol central és una } a \\ |a|+1 & \text{si el símbol central és una } b \end{cases}
$$

Per l'entrada *aabba*, l'arbre sintàctic seria el següent:



Una gramàtica alternativa podria ser la següent:

$$
\begin{array}{lcll}
S & \to & L & \{S.x = L.x\} \\
L & \to & A_1 L_1 A_2 & \{L.x = A_1.x + L_1.x + A_2.x\} \\
A & \to & a & \{A.x = 1\} \\
A & \to & b & \{A.x = 0\} \\
L & \to & a & \{L.x = 0\} \\
L & \to & b & \{L.x = 1\}
\end{array}
$$

3 Generació i optimització de codi (3.5 punts)

Donat el codi següent que calcula el segon valor més petit d'un vector:

```
int segon(int v[], int n) {
  int x = v[0];
  int y = v[1];
  if (y < x) { int z = x; x = y; y = z; }
  int i = 1;
  while (i < n - 1) {
    ++i;
    if (v[i] < y) {
      y = v[i];
      if (y < x) { int z = x; x = y; y = z; }
    }
  }
  return y;
}
```

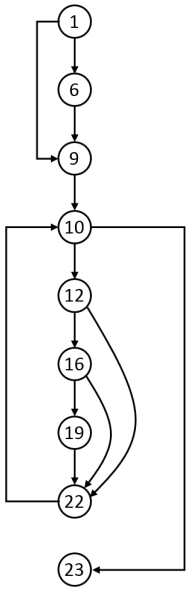
1. Genereu codi de tres adreces sense optimitzar, numerant totes les instruccions consecutivament: `(1), (2), (3), ...` Identifiqueu els *leaders* dels blocs bàsics en el mateix codi. Suposeu que un `int` ocupa 4 bytes.
2. Dibuixeu el graf de blocs bàsics, posant com etiqueta de cada node el número del *leader* del bloc bàsic.
3. Dibuixeu l'arbre de dominadors dels blocs bàsics.
4. Escriviu el codi després d'optimitzar. Expliqueu cadascuna de les optimitzacions realitzades en el codi.

Resposta:

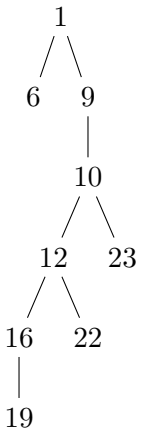
Codi no optimitzat

```
(1)     t1 = 4*0
(2)     x = v[t1]
(3)     t2 = 4*1
(4)     y = v[t2]
(5)     ifFalse y < x goto L1
(6)     z = x
(7)     x = y
(8)     y = z
(9) L1: i = 1
(10) L2: t3 = n - 1
(11)    ifFalse i < t3 goto L4
(12)    i = i + 1
(13)    t4 = i*4
(14)    t5 = v[t4]
(15)    ifFalse t5 < y goto L3
(16)    t6 = i*4
(17)    y = v[t6]
(18)    ifFalse y < x goto L3
(19)    z = x
(20)    x = y
(21)    y = z
(22) L3: goto L2
(23) L4: return y
```

Graf de blocs bàsics



Arbre de dominadors



Codi optimitzat (1)

```
    x = v[0]
    y = v[4]
    ifFalse y < x goto L1
    z = x
    x = y
    y = z
L1: i = 1
L2: t3 = n - 1
    ifFalse i < t3 goto L4
    i = i + 1
    t4 = i*4
    t5 = v[t4]
    ifFalse t5 < y goto L2
    y = v[t4]
    ifFalse y < x goto L2
    z = x
    x = y
    y = z
L3: goto L2
L4: return y
```

Codi optimitzat (2)

```
    x = v[0]
    y = v[4]
    ifFalse y < x goto L1
    z = x
    x = y
    y = z
L1: i = 1
L2: t3 = n - 1
    ifFalse i < t3 goto L4
    i = i + 1
    t4 = i*4
    t5 = v[t4]
    ifFalse t5 < y goto L2
    y = t5
    ifFalse t5 < x goto L2
    z = x
    x = t5
    y = z
L3: goto L2
L4: return y
```

Codi optimitzat (3)

```
    x = v[0]
    y = v[4]
    ifFalse y < x goto L1
    z = x
    x = y
    y = z
L1: i = 1
    t3 = n - 1
    t4 = 4
L2: ifFalse i < t3 goto L4
    i = i + 1
    t4 = t4 + 4
    t5 = v[t4]
    ifFalse t5 < y goto L2
    y = t5
    ifFalse t5 < x goto L2
    z = x
    x = t5
    y = z
L3: goto L2
L4: return y
```

En el primer codi optimitzat s'han propagat constants i s'han detectat expressions comunes (`i*4`). També s'han optimizat salts (`goto L2`). En el segon codi s'ha detectat l'expressió comuna `v[t4]` i s'ha propagat la còpia `y=t5`. Finalment, en el tercer codi s'ha detectat l'invariant `t3=n-1` i la variable d'inducció `t4`.