Compiladors: Examen de teoria

18 de juny de 2013

Duració de l'examen: 2.5 hores

1 Anàlisi sintàctica (2.5 punts)

Donada la següent gramàtica:

$$\begin{array}{lrcl} (1) & S' & \rightarrow & S\$ \\ (2) & S & \rightarrow & AA \\ (3) & A & \rightarrow & aA \\ (4) & A & \rightarrow & b \end{array}$$

- Descriure el llenguatge que genera.

  Resposta: El llenguatge generat és a^*ba^*b.

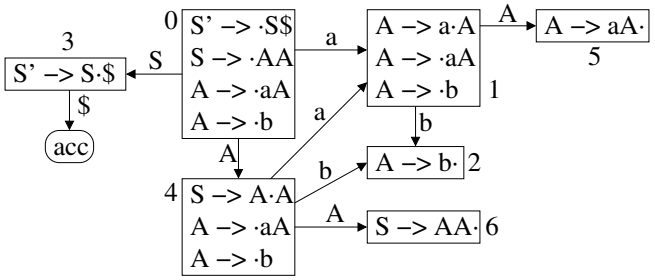
- Calcular *First* i *Follow* pels símbols S i A.

  Resposta:

	First	*Follow*
S	$\{a, b\}$	$\{\$\}$
A	$\{a, b\}$	$\{a, b, \$\}$

- Generar l'autòmat i la taula LR(1).

  Resposta:



	a	b	$	S	A
0	s1	s2		3	4
1	s1	s2			5
2	r4	r4	r4		
3			acc		
4	s1	s2			6
5	r3	r3	r3		
6			r2		

2 Generació de codi (2.5 punts)

Volem generar codi per a l'expressió condicional de C++ utilitzant l'avaluació d'expressions booleanes amb *backpatching*: $B\ ?\ E_1 : E_2$.

Es demana:

- Escriure el codi generat per la següent instrucció (utilitzant backpatching):

$$x = i < n \texttt{ and not } (\text{valid } \texttt{and } i > j)\ ?\ y : z + 1;$$

Resposta:

```
            ifFalse i < n goto Lfalse
            ifFalse valid goto Ltrue
            ifFalse i > j goto Lfalse
Ltrue:      t = y
            goto Lend
Lfalse:     t = z+1
Lend:       x = t
```

- Dissenyar la funció de generació de codi per a les expressions condicionals:

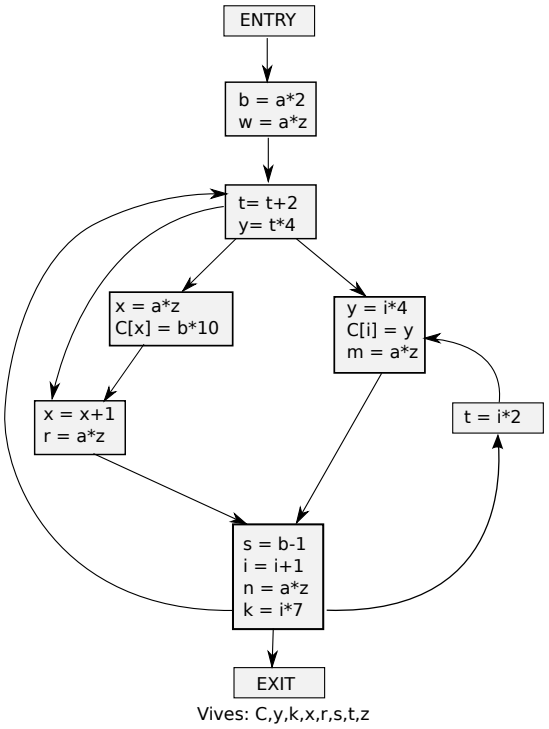
$$\texttt{EvalExpr } (E \rightarrow B\ ?\ E_1 : E_2)$$

Resposta:

```
EvalExpr (E -> B ? E1 : E2)
    Ltrue = new Label(); Lfalse = new Label(); Lend = new Label();
    E.addr = new Temp();
    EvalBoolExpr(B, Ltrue, Lfalse); EvalExpr(E1); EvalExpr(E2);
    E.code =     B.code ||
             "Ltrue:" ||
                 E1.code ||
                 "E.addr = E1.addr" ||
                 "goto Lend" ||
             "Lfalse:" ||
                 E2.code ||
                 "E.addr = E2.addr" ||
             "Lend:";
```

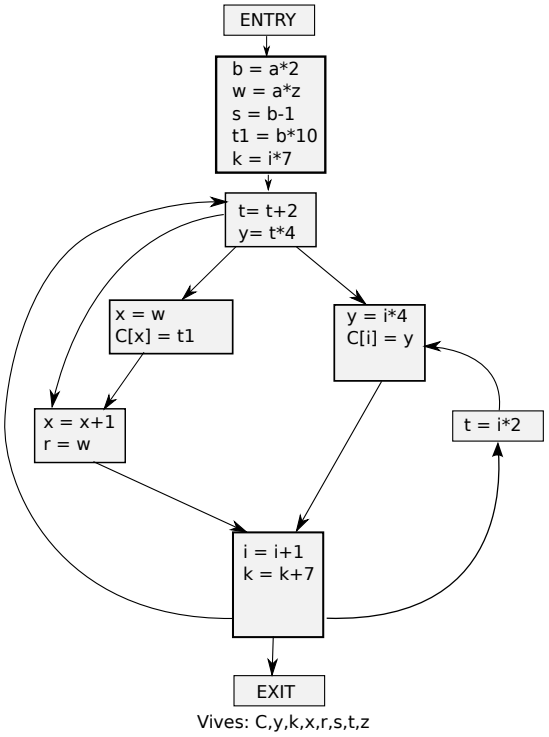
3 Optimització (2.5 punts)

Considerem el següent diagrama de blocs bàsics:



Suposem que totes les variables han estat inicialitzades abans del bloc d'entrada i que les variables vives en el bloc de sortida són $\{C, y, k, x, r, s, t, z\}$. Es demana aplicar totes les optimitzacions possibles explicant el raonament utilitzat per a cada optimització.

Resposta: El codi que resulta després de fer totes les optimitzacions és:



On s'han aplicat:

- Detecció d'invariants: `S=b-1`, `t1=b*10`
- Eliminació d'expressions comunes: `a*z`
- Eliminació de codi mort: `m = a*z`, `n = a*z`
- Variables d'inducció: `i,k`

4 Intèrpret (2.5 punts)

Volem dissenyar un intèrpret per descriure i calcular rutes de mercaderies. Les rutes es poden definir literalment:

```
Ruta1 := Tarragona Barcelona 100
Ruta2 := Girona Barcelona 105
Ruta3 := Barcelona Igualada 70
Ruta4 := Igualada Lleida 95
Ruta5 := Lleida LaSeuDUrgell 130
Ruta6 := LaSeuDUrgell Andorra 20
Ruta7 := Barcelona Berga 110
Ruta8 := Berga Puigcerda 51
Ruta9 := Puigcerda LaSeuDUrgell 48
```

És a dir, hi ha una carretera que uneix Tarragona i Barcelona amb una distància de 100 kms, una carretera que uneix Girona i Barcelona amb una distància de 105 kms, etc. La definició de rutes és unidireccional, és a dir, la Ruta1 només defineix el trajecte de Tarragona a Barcelona, però no el de Barcelona a Tarragona (per això caldria definir una altra ruta).

També es poden definir rutes a partir de composar-les via dos operadors: `'+'` representa la concatenació de rutes, mentre que `'|'` representa l'alternativa de rutes. Aleshores, rutes possibles a partir de les rutes definides abans podrien ser:

```
Ruta10 := Ruta3 + Ruta4
Ruta11 := Ruta1 + (Ruta10 + Ruta5 | Ruta7 + Ruta8 + Ruta9)
```

És a dir, la Ruta10 defineix un trajecte entre Barcelona i Lleida, passant per Igualada. La Ruta11 defineix dos possibles trajectes per anar de Tarragona a La Seu d'Urgell. L'operador `'+'` (`CONCAT`) té mes prioritat que l'operador `'|'` (`ALTER`). Com es pot veure a l'exemple (Ruta11), es pot canviar la prioritat amb l'ús de parèntesi.

1) Descriure una gramàtica que defineixi el llenguatge de les rutes, suposant que els tokens `ID`, `ASSIGN`, `INTCONST`, `CONCAT`, `ALTER`, `LPAR` i `RPAR` ja han estat definits prèviament.

Resposta:

$$
\begin{aligned}
\texttt{rutes} &\rightarrow (\texttt{ruta})* \\
\texttt{ruta} &\rightarrow \texttt{ID ASSIGN (directa} \mid \texttt{exp)} \\
\texttt{directa} &\rightarrow \texttt{ID ID INTCONST} \\
\texttt{exp} &\rightarrow \texttt{expconc (ALTER expconc)}* \\
\texttt{expconc} &\rightarrow \texttt{atom (CONCAT atom)}* \\
\texttt{atom} &\rightarrow \texttt{ID} \mid \texttt{LPAR exp RPAR}
\end{aligned}
$$

2) Suposem que cada node de l'AST conté els atributs `tipus`, `origen` i `destí`. L'atribut `tipus` determina el tipus del node (`DIRECTA` per definicions de rutes entre dues ciutats, `ID` per rutes ja definides, i `CONCAT`

i `ALTER` per rutes composades), mentre que els atributs `origen` i `destí` determinen, donada una definició de ruta (per exemple la Ruta1 o la Ruta10 de l'arbre anterior), les ciutats origen i destí.

El node de tipus `DIRECTA` té tres fills, que representen l'origen, destí i distància. El node de tipus `ID` no té cap fill i només inclou el nom de la ruta que representa en el seu camp de text. Tots els nodes de l'AST tenen l'atribut `text` de tipus string al que s'hi pot accedir amb la funció `getText()`. Els nodes de tipus `CONCAT` i `ALTER` tindran tants fills com termes tingui l'expressió que representen. Per accedir als fills d'un node `n` es poden fer servir les funcions `n.getChildCount()` i `n.getChild(i)`.

Es demana completar el procediment `CalculaOrigenDesti` que hi ha a continuació per a calcular els atributs `origen` i `destí` en els nodes corresponents de l'AST. Aquest ha de comprovar dos propietats principals de tota ruta: a) només hi ha un origen i un destí, i b) la ruta està ben formada: en concatenar dues rutes, la ciutat destí de la primera ha de ser igual a la ciutat origen de la segona. Quan algun dels requeriments no es dona, s'emetrà un missatge d'error.

```
void CalculaOrigenDesti(AST a) {
   if (a.tipus == DIRECTA) {
      a.origen = a.getChild(0).getText();
      a.desti = a.getChild(1).getText();
   }
   else if (a.tipus == ID)  {
     ...
   }
   ...
```

Es pot suposar que es disposa d'una funció que permet accedir a l'arrel de la definició d'una ruta a partir del seu nom:

```
AST TrobaRuta(String nom)
```

Resposta:

En aquesta solució, per claretat, es recalculen els origens i destins de les rutes cada vegada que són utilitzades. Es podrien marcar inicialment per evitar recalcular-ho.

```
void CalculaOrigenDesti(AST a) {
   if (a.tipus == DIRECTA) {
      a.origen = a.getChild(0).getText();
      a.desti = a.getChild(1).getText();
   }
   else if (a.tipus == ID) {
      AST b = TrobaRuta(a.getText());
      CalculaOrigenDesti(b);
      a.origen = b.origen;
      a.desti = b.desti;
   }
   else if (a.tipus == ALTER) {
      CalculaOrigenDesti(a.getChild(0));
      a.origen = a.getChild(0).origen;
      a.desti = a.getChild(a,0).desti;
      for(int i = 1; i < a.getChildCount(); ++i) {
         CalculaOrigenDesti(a.getChild(i));
         if (a.getChild(i).origen != a.origen or a.getChild(i).desti != a.desti)
            Error("Node | amb discrepancies origen-desti")
      }
   }
   else { // a.tipus == CONCAT
      CalculaOrigenDesti(a.getChild(0));
      a.origen = a.getChild(0).origen;
```

```
      a.desti = a.getChild(a,0).desti;
      for(int i = 1; i < a.getChildCount(); ++i) {
         CalculaOrigenDesti(a.getChild(i));
         if (a.getChild(i).origen != a.desti)
            Error("Node + amb discrepancies origen-desti")
         a.desti = a.getChild(i).desti
      }
   }
}
```

3) Suposem que els nodes de l'AST tenen addicionalment un nou atribut: `distMin`, que donada una ruta (que pot ser composada o no), conté la distància mínimia entre les ciutats origen i destí de la ruta. Suposarem que totes les distàncies estan representades com a nombres enters positius. A partir d'aquesta modificació de l'AST, dissenyar la funció següent:

```
int DistanciaMinima(AST a)
```

que donat un node de l'AST, trobi la distància corresponent al trajecte de distància mínima entre les ciutats que connecta i ho assigni a l'atribut `distMin` del node. La funció retorna el valor assignat.

```
int DistanciaMinima(AST a) {
     if (a.tipus == DIRECTA) a.distMin = Integer.parseInt(a.getChild(2).getText());
     else if ...
     ...
     ...
     return a.distMin;
}
```

Resposta:

```
int DistanciaMinima(AST a) {
   if (a.tipus == DIRECTA)  a.distMin = Integer.parseInt(a.getChild(2).getText());
   else if (a.tipus == ID) a.distMin = DistanciaMínima(TrobaRuta(a.getText()));
   else if (a.tipus == CONCAT) {
      // Suma de distancies
      a.distMin = 0;
      for (int i = 0; i < a.getChildCount(); ++i) {
         a.distMin += DistanciaMinima(a.getChild(i))
      }
   }
   else { // a.tipus == ALTER
      // Distancia minima
      a.distMin = DistanciaMinima(a.getChild(0))
      for (int i = 1; i < a.getChildCount(); ++i) {
         a.distMin = min(a.distMin, DistanciaMinima(a.getChild(i)));
      }
   }
   return a.distMin;
}
```