Compiladors

(Examen final, 19 de juny de 2020)

1 Parsing (1/3)

Considereu la gramàtica següent:

$$
\begin{array}{lcl}
S & \rightarrow & A\$ \\
A & \rightarrow & B \\
A & \rightarrow & aAc \\
B & \rightarrow & \\
B & \rightarrow & BbB
\end{array}
$$

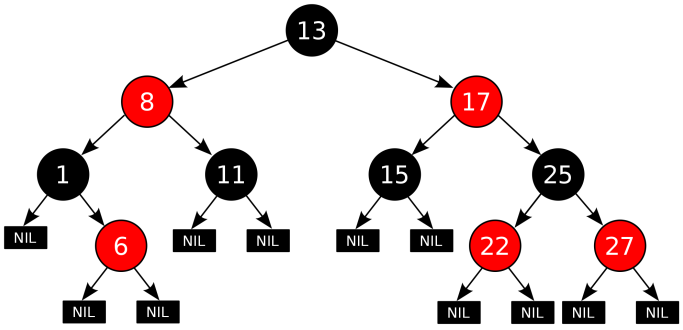
Es demana:

- Calcular *Nullable*, *First* i *Follow* dels símbols no terminals.
- Generar un analitzador SLR(1): autòmat i taula de parsing.
- En cas que l'analitzador tingui conflictes, identificar els conflictes, redissenyar la gramàtica i tornar a generar l'analitzador.

Nota: En cas que la primera gramàtica tingui conflictes, no cal desenvolupar l'analitzador sintàctic completament. Només cal desenvolupar-ho fins el punt on sigui evident el conflicte o donar una justificació formal del per què existeix el conflicte.

2 Gramàtiques d'atributs (1/3)

Tenim un llenguatge per descriure Red-black trees (RBT) de nombres enters positius, tal com es mostra a la figura. En el llenguatge utilitzat, les etiquetes `'b'` i `'r'` representen el color de cada node.




```
(b13
  (r8 (b1 () r6) b11)
  (r17 b15 (b25 r22 (r27)))
)
```

Per a representar l'arbre fem servir l'estructura següent:

```
struct Node {
  int   value;  // Valor del node (enter positiu)
  Node* left;   // Arbre dret (nullptr si no hi ha arbre dret)
  Node* right;  // Arbre esquerra (nullptr si no hi ha arbre esquerra)
  bool  black;  // color del node: negre (cert) o vermell (fals)
  bool  rbt;    // indica si es un red-black tree
  bool  bst;    // indica si es un arbre binari de cerca
  // ... altres atributs
};

using Tree = Node*;
```

Recordem la propietat fonamental d'un arbre binari de cerca (BST):

> Per a cada node de l'arbre, tots els valors del subarbre esquerre han de ser menors que el valor del node i tots els valors del subarbre dret han de ser majors que el valor del node.

Per tal que un BST sigui un RBT cal que també es compleixin les propietats següents:

- Cada node ha de tenir només un color: vermell o negre.
- Totes les fulles (NIL) són negres.
- Si un node és vermell, els seus fills són negres.
- Tots els camins que van d'un node a les fulles descendents (NIL) atravessen el mateix nombre de nodes negres.

Nota: considerarem que les fulles (NIL) estan representades pels `nullptr` dels nodes.

Per exemple, l'arbre de la figura és un RBT. En canvi, l'arbre descrit a continuació no ho és:

```
(b2 b1 (r5 r4 ()))
```

En aquest exercici es pot estendre l'estructura `Node` per tal d'acollir informació necessaria per a resoldre el problema demanat. Podem suposar que l'analitzador lèxic ens proporciona dos tokens (`color` i num) on `color.val` és de tipus `char` i pot valer `'b'` o `'r'` i `num.val` és de tipus `int` i conté el valor del node representat.
Podeu suposar que a la gràmatica d'atributs hi ha una regla del tipus:

```
N → color num
      {N.tree = new Node {num.val, nullptr, nullptr, color.val == 'b', ...}}
```

Es demana:

- Afegir els atributs necessaris a l'estructura `Node` per resoldre les preguntes d'aquest problema. Especificar clarament el significat de cada atribut.
- Dissenyar una gramàtica d'atributs que accepti el llenguatge descrit anteriorment, construeixi l'arbre i calculi la informació necessària per saber si l'arbre és un RBT i un BST.

El valor dels atributs auxiliars és irrellevant una vegada construït l'arbre. Cal que els atributs `bst` i `rbt` siguin consistents a tots els nodes de l'arbre. Cal pensar que un arbre podria ser un BST però no ser un RBT. En canvi, al revés no pot passar (tot RBT és BST). És a dir, `RBT` $\implies$ `BST`.
Es recomana dissenyar la funció

```
void check(Tree T);
```

que actualitza els atributs de l'arbre i que podrà ser utilitzada a les accions semàntiques de la gramàtica.
Importat: es valorarà la senzillesa de la gramàtica i de la funció `check`. Cal utilitzar un estil de programació tant senzill i elegant com sigui possible.

3 Generació de codi (1/3)

Considereu l'estructura de dades i el codi següents:

```
struct info {
    double x;
    int v;
    int list[10];
};
info A[100];

A[i].v = A[i+3].list[A[j].v] - 1;
```

Suposeu que les mides d'`int` i `double` són 4 i 8 bytes, respectivament.

- Dibuixeu l'AST del tipus de la variable `A`. Annoteu les mides i els offsets (en bytes) associats a cada node de l'AST. Podeu fer servir l'estil dels exemples que hi ha als apuntes o els exercicis de la col.lecció de problemes.
- Suposem que l'adreça base d'`A` és 1000.
  - A quina adreça de memòria es trobaria A[4].v?
  - A quina adreça es trobaria? A[4].list[2]?

  Indiqueu els càlculs que heu fet per arribar a les solucions.
- Dibuixeu l'AST de l'assignació.
- Escriviu el codi generat (no optimitzat) corresponent a l'AST de l'assignació. Poseu a cada node de l'AST els valors `addr` i `offset` que s'anirien propagant a mesura que es genera el codi (mireu els exemples de les pàgines 21 i 26 del capítol de *Intermediate Code Generation*). No cal que anoteu el codi a l'AST. Només cal que doneu el codi final.