Parseo y Generación de Código – 2do semestre 2016
Licenciatura en Informática con Orientación en Desarrollo de Software
Universidad Nacional de Quilmes

Primer parcial

Nota: este parcial es a libro abierto. Se permite tener cualquier material manuscrito o impreso, pero no se permite el uso de dispositivos electrónicos. El parcial dura 3 horas y se califica con una nota numérica de 1 a 10. Se requiere ≥ 4 en ambos parciales para aprobar la materia. Para promocionar se requiere nota ≥ 6 en ambos parciales y promedio ≥ 7.

Ejercicio 1. Dada la expresión regular $R = a^*(a|b)$ en el alfabeto $\{a, b\}$.

a. Aplicar la construcción de Thompson para obtener un AFN que acepte el lenguaje $L(R)$.

b. Convertir el AFN en un AFD usando la construcción de subconjuntos.

Indicar claramente los estados iniciales, finales y transiciones de cada autómata. Recordar que **todos** los estados de un AFD deben tener transiciones para **todos** los símbolos del alfabeto.

Ejercicio 2. Considerar la gramática $G = (\Sigma, \{E\}, E, P)$ de expresiones booleanas en el alfabeto $\Sigma = \{\mathsf{True}, \mathsf{False}, \mathsf{and}, \mathsf{not}, (,)\}$, en la que el not es un operador **sufijo**, con las siguientes producciones:

$$E \quad \rightarrow \quad \mathsf{True} \mid \mathsf{False} \mid E \ \mathsf{and} \ E \mid E \ \mathsf{not} \mid (E)$$

a. Mostrar que G es ambigua.

b. Dar una gramática G_1 no ambigua que genere el mismo lenguaje que G.

c. Eliminar la recursión a izquierda de la gramática G_1 y exhibir la gramática G_2 obtenida.

Ejercicio 3. Dada la gramática $G = (\{S, A, B, X\}, \{0, 1\}, S, P)$ con las siguientes producciones:

$$\begin{array}{lcl} S & \rightarrow & X0A \mid 1A \\ A & \rightarrow & B \mid AB \\ B & \rightarrow & 0 \mid 1 \\ X & \rightarrow & \epsilon \end{array}$$

a. Construir el autómata de análisis sintáctico LR(0) y armar la tabla de análisis sintáctico.

b. Indicar los conflictos presentes en la tabla y verificar que G no es LR(0).

c. Mostrar que G es SLR.

d. Analizar sintácticamente la cadena 010 con el algoritmo SLR, mostrando cómo evoluciona el estado de la pila y cómo se consume la entrada a medida que se ejecuta el algoritmo.

Ejercicio 4. Dado un árbol de sintaxis abstracta para un lenguaje de expresiones booleanas con declaraciones de variables locales:

```
type Id = String
data Expr = ExprConstBool Bool      -- constante booleana
          | ExprAnd Expr Expr       -- conjunción (e1 and e2)
          | ExprLet Id Expr Expr    -- declaración (let x = e1 in e2)
          | ExprVar Id              -- variable (x)
type Cont = Bool -> Result
```

Definir un intérprete usando la estrategia de evaluación call-by-value (la construcción `let` debe evaluar `e1` antes que `e2`). El intérprete recibe como parámetros una expresión, un entorno de tipo `Env` que asocia variables a booleanos y una continuación de tipo `Cont` que dado un booleano devuelve un resultado final.

```
eval :: Expr -> Env -> Cont -> Result
```

Si E :: `Env` es un entorno y x :: `Id` es una variable, la expresión $E(x)$ :: `Bool` denota el valor de la variable. Si b :: `Bool` es un booleano, $E[x := b]$ :: `Env` es el entorno extendido asociando la variable x al valor b.