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

Recuperatorio del 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. Considerar el lenguaje $\mathcal{L}$ de las palabras en el alfabeto $\{0, 1\}$ que tienen exactamente 2 ó 3 ocurrencias de 0, es decir, $\mathcal{L} = \{\alpha \in \{0,1\}^\star \mid \#_0(\alpha) \in \{2,3\}\}$. Dar un AFD que reconozca el lenguaje $\mathcal{L}$.

Indicar claramente el estado inicial, los estados finales y las transiciones del autómata. Recordar que **todos** los estados de un AFD deben tener transiciones para **todos** los símbolos del alfabeto.

Ejercicio 2.

a. Mostrar que la siguiente gramática $G = (\{a, b, c\}, \{S, A\}, S, P)$ es ambigua:

$$\begin{array}{lcl} S & \to & aSA \mid c \\ A & \to & \epsilon \mid bS \end{array}$$

Sugerencia: considerar la cadena *aacbc*.

b. Mostrar que la siguiente gramática $G' = (\{a, b, c\}, \{S, A\}, S, P')$ es LL(1) y por lo tanto no es ambigua:

$$\begin{array}{lcl} S & \to & aSA \mid \epsilon \\ A & \to & c \mid bS \end{array}$$

Ejercicio 3. Dada la siguiente gramática $G = (\{a, b, c\}, \{S, A, B\}, S, P)$:

$$\begin{array}{lcl} S & \to & Aa \mid AaS \mid Bb \mid BbS \\ A & \to & a \\ B & \to & a \mid Ac \end{array}$$

a. Construir el autómata y la tabla de análisis sintáctico LR(0), y mostrar que G no es LR(0). Indicar los conflictos presentes en la tabla.

b. Decidir si G es SLR.

Ejercicio 4. Considerar un lenguaje con expresiones numéricas y tuplas. Una expresión se representa con un árbol de sintaxis abstracta de tipo `Expr`:

```
data Expr = ExprInt Int          -- constante numérica
          | ExprAdd Expr Expr    -- suma
          | ExprTuple Expr Expr  -- constructor de tuplas
          | ExprFst Expr         -- proyección de la primera componente de la tupla
          | ExprSnd Expr         -- proyección de la segunda componente de la tupla
```

Toda expresión denota un valor de tipo `Val`:

```
data Val = VInt Int
         | VTuple Val Val
```

Una continuación de tipo `Cont` es una función que dado un valor devuelve un resultado final de tipo `Result`:

```
type Cont = Val -> Result
```

Definir un intérprete *continuation-passing* que reciba una expresión y una continuación y devuelva un resultado final. El resultado final es el que se obtiene al aplicar la continuación al valor que denota la expresión.

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

Justificar todas las respuestas.