Universidad Simón Bolívar
Dpto. de Computación y Tecnología de la Información
CI3721 - Traductores e Interpretadores
Abril-Julio 2006

Proyecto Unico - Parte 2

Esta segunda etapa del proyecto corresponde al módulo de análisis sintáctico del interpretador de Máquinas de Turing [1, Sección 7.2] descrito en el enunciado de la Parte 1. Primero debe desarrollar una serie de preguntas preliminares relacionadas con éste módulo y, luego, implemente el analizador sintáctico usando Happy.

Preliminares

Sea $G1$ la gramática $(\{Expr\}, \{+, \mathbf{num}\}, P1, Expr)$, con $P1$ compuesto por las siguientes producciones:

$$\begin{aligned} Expr &\rightarrow Expr + Expr \\ Expr &\rightarrow \mathbf{num} \end{aligned}$$

1. Muestre que la frase $\mathbf{num} + \mathbf{num} + \mathbf{num}$ de $G1$ es ambigua.
2. Dé una gramática no ambigua $Izq(G1)$ que genere el mismo lenguaje que $G1$ y que asocie las expresiones aritméticas generadas hacia la izquierda. Dé también una gramática $Der(G1)$ con las mismas características pero que asocie hacia la derecha.
3. Construya la gramática $G1$ en Happy. ¿Qué responde Happy al procesarla?
4. Happy debe haberle indicado la presencia de un "conflicto", cuyo significado exacto no nos atañe por el momento más que como indicador de ambigüedad. Happy es capaz de resolver la ambigüedad de $G1$ por Ud., simplemente marcando el token $'+'$ con `%left` o `%right`. La solución favorece la asociación hacia izquierda o derecha, respectivamente. Verifique que el conflicto desaparece al añadir esas directivas
5. Dé una derivación "más a la izquierda" (*leftmost*) y una derivación "más a la derecha" (*rightmost*) de $G1$ para $\mathbf{num} + \mathbf{num}$.
6. Considere la expresión de ejecución de MTD como se define más abajo. Suponga que se emplea la gramática $G2$ definida con producciones

$$\begin{aligned} E &\rightarrow \mathbf{ejecutar\ id\ usando}\ E \\ &\rightarrow \mathbf{ejecutar\ id\ usando}\ E\ \mathbf{mostrando}\ D \\ &\rightarrow \mathbf{id} \\ D &\rightarrow \mathbf{transiciones} \\ &\rightarrow \mathbf{todo} \end{aligned}$$

   (a) ¿Es ambigüa?
   (b) ¿Qué tipo de conflicto presenta al ser suministrada a Happy?
   (c) Proponga dos soluciones a la ambigüedad, una modificando el lenguaje y otra sólo reescribiendo la gramática.

Implantación

Ud. debe desarrollar el analizador sintáctico completo de MTD utilizando el Generador de Analizadores Sintácticos Happy en el lenguaje de programación Haskell, además de hacer una breve revisión de los conceptos y métodos teórico-practicos relevantes.

El analizador sintáctico a ser construido en esta etapa deberá ser encapsulado en un módulo Haskell de nombre `Parser`, que exporte **únicamente** la función `parser` generada por Happy. Utilizará el analizador lexicográfico desarrollado en la Parte 1, el cual deberá estar encapsulado en un módulo `Lexer` separado que exporte **únicamente** el tipo de datos de los *tokens* y la función `lexer`. La función `mtd` que recibirá como argumento una cadena alfanumérica que indica el nombre del archivo de texto a procesar, y que producirá una cadena con las producciónes utilizadas, debe encapsularse en un módulo Haskell de nombre `Main` que importa los módulos `Lexer` y `Parser` para hacer su trabajo.

Desarrolle la gramática para MTD considerando que:

- En un mismo archivo **deben** aparecer al menos una definición de máquina, una definición de cinta y una definición de corrida.
- En un mismo archivo **pueden** aparecer varias definiciones de máquina, varias definiciones de cintas y varias definiciones de corridas, en **cualquier** orden.
- Todas las listas asociadas a conjuntos (estados, finales, gamma, sigma) **deben** tener al menos un elemento.
- El conjunto de transiciones **puede** estar vacío. Note que **no debe haber** un punto y coma después de la última transición, y no lo habría si sólo hay una transición.
- Los elementos que definen una máquina (estados, inicial, final, etc.) deben aparecer **una** sola vez cada uno y en el mismo orden que en el ejemplo del enunciado de la Parte 1.
- Las cintas de entrada **pueden** estar vacías.
- La instrucción `ejecutar` ha sido extendida. Ahora es posible utilizar la instrucción `ejecutar` de dos maneras:
  - De la misma forma y con el mismo efecto con que se definió en el enunciado de la primera parte (con o sin `mostrando`), i.e.

    `ejecutar` *`maquina`* `usando` *`cinta`*

  - Usando como cinta de entrada la cinta producida por una ejecución previa. En otras palabras, es posible anidar ejecuciones de máquinas, e.g. si existen las máquina $m1$ y $m2$, y la cinta de entrada $c1$, entonces se puede ejecutar $m1$ usando como entrada $c1$ y la cinta producida será utilizada como entrada para $m2$ escribiendo en nuestro lenguaje MTD

    ```
    ejecutar m2 usando ejecutar m1 usando c1
    ```

  - Puede anidarse la ejecución de manera arbitraria.
  - Si se usa la cláusula `mostrando` para conseguir una traza de las ejecuciones, siempre estará asociada a la ejecución más interna, por lo tanto es posible tener

```
ejecutar m2 usando
  ejecutar m1 usando c1
  mostrando transiciones  # Las de m1
ejecutar m2 usando
  ejecutar m1 usando c1
  mostrando transiciones  # Las de m1
mostrando todo            # Todo lo de m2
```

- En todas las producciones de los símbolos no terminales de la gramática, indique a Happy que imprima cada producción. En aquellas producciones donde haya *tokens*, imprima su valor particular en la posición que corresponda[1].
- Si se detecta un error de sintaxis, el parser debe abortar inmediatamente indicando la posición en el archivo donde ocurrió.

Detalles de la Entrega

- **Implantación**. Ud. debe entregar en el formato electrónico acordado en clase personalmente o por correo electrónico los archivos `Lexer.x` con el código para Alex y la definición del TAD para Tokens, `Parser.y` con el código para Happy y `Main.hs` con el código para el programa principal. Los módulos deben incluir documentación Haddock para describir los detalles de implementación suficientes.
- **Revisión Teórico-Práctica**. Ud. debe entregar en papel o en PDF el desarrollo de las preguntas preliminares.
- **Fecha de Entrega.** Jueves 8 de Junio (Semana 7) hasta las 6:00pm
- **Valor de Evaluación.** Nueve (9) puntos.

References

[1] J. Hopcroft, J. Ullman
Introduction to Automata Theory, Languages and Computation
Addison-Wesley Publishing Company, 1979
ISBN 0-201-02988-X QA267.H56

[1] Esto es, si hubiera una producción de la forma

```
expr : id mas num
```

donde `expr` es no terminal, `id` es un *token* identificador con valor 'foo', `mas` es un *token* sin valor particular y `num` es un *token* número con valor 123, entonces debe imprimirse

```
expr -> id('foo') mas num('123')
```

Si hay producciones vacías, indíquelo imprimiendo `lambda`.