Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Análisis de Lenguajes de Programación

Trabajo Práctico Final

EDSL de Automatas de pila
Manual de Usuario

Cavagna, Lucas Gastón

Febrero 2023

Índice general

1 Introducción

Se le da la bienvenida al manual de usuario del EDSL de autómatas de pila programado en Haskell. En este caso, el usuario dispondrá de múltiples comandos y características junto con un intérprete que también puedo brindarle ayuda.

El enfoque principal de este EDSL es la definición, ejecución y combinación de autómatas de pila (en particular no deterministas). En cuanto a la definición, es posible definir a los autómatas utilizando gramáticas independientes del contexto. Una vez introducidas, se construye un autómata de pila que acepta el mismo lenguaje.

1.1. Instalación de software necesario

Inicialmente, lo primero que es necesario instalar es Haskell. En caso de no tenerlo ya instalado puede consultar Haskell para realizar la instalación correctamente.

En este caso, para la construcción de este lenguaje de dominio específico embebido se utilizó el gestor de proyectos **STACK** en Haskell. Stack tiene muchas utilidades, pero ahora nos vamos a concentrar sus funciones básicas.

Puede que sea posible que tenga que instalarlo, para esto puede ir a HaskellStack donde hay guías de instalación para distintas plataformas.

Stack se encarga de instalar la versión correcta de GHC, instalar los paquetes necesarios entre otras cosas. Para las primeras dos, basta con abrir una terminal en el directorio TPFINAL y ejecutar:

stack setup

Esto puede demorar un rato porque se encarga de descargar e instalar la verisón correcta de GHC. Este comando solo se debería tener que ejecutar una única vez.

1.2. Ejecución del programa

Ya instalado Haskell y Stack, tenemos que dejar que Stack haga su parte y compile el proyecto. Esto se llevara a cabo utilizando lo siguiente en una terminal en el directorio TPFINAL:

stack build

Además de compilar el proyecto es posible que se instalen algunos paquetes claves del proyecto. Esto último puede tomar un tiempo.
Y ahora ya estamos listos para utilizar el DSL. Solo debemos abrir una terminal en el directorio TPFINAL y ejecutar:

stack exec TPFINAL-exe

Incluso podemos agregar archivos que queremos que se carguen de la siguiente manera:

stack exec TPFINAL-exe <ruta del archivo> <ruta del archivo> <ruta del archivo> ...

2 Operaciones

Antes de mencionar y explicar todos los comandos que se encuentran disponibles en este EDSL vamos a resaltar todas las operaciones utilizables que tenemos entre los autómatas de pila:

- Operación de Unión: esta operación binaria nos permite unir 2 autómatas de pila, teniendo así uno nuevo con la capacidad de aceptar los lenguajes de los dos autómatas participantes. Esta operación se nota como 'Union' y se coloca entre los dos autómatas. De modo tal que nos queda: α Union β, donde α y β son autómatas.
- Operación de Concatenación: esta operación binaria nos permite concatenar 2 autómatas de pila, generando un autómata que acepta la cadena ab, donde a es una cadena que acepta el primer autómata y b es una aceptada por el segundo. Esta operación se nota como 'Concat' y se coloca antes de los dos autómatas. De modo tal que nos queda: Concat α β, donde α y β son autómatas.
- Operación Clausura de Kleene: es una operación unaria que nos permite generar un autómata que acepta la unión infinita de las potencias del lenguaje aceptado por el autómata argumento.

$$L^* = \bigcup_{n \geq 0} L^n$$

Es decir, L^* es la unión infinita

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots$$

donde L es el lenguaje acepatado por el autómata

Esta operación se nota como '*' y se coloca antes del autómata. De modo tal que nos queda: * α , donde α es un autómata

Ahora, dado un lenguaje L, definimos inductivamente su potencia L^n como:

$$L^0 = \{\lambda\}$$
$$L^{n+1} = L^n L$$

- Operación Clausura Positiva de Kleene: es una operación unaria que nos permite generar un autómata que acepta la unión infinita de las potencias del lenguaje acepado por el autómata de entrada.

$$L^+ = \bigcup_{n \geq 1} L^n$$

$$L^+ = L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots$$

donde L es el lenguaje aceptado por el autómata

Esta operación se nota como '+' y se coloca antes del autómata. De modo tal que nos queda: + α , donde α es un autómata

- Operación Potencia: esta operación binaria nos permite crear un autómata que acepta la potencia n-esima (con n>0) del lenguaje aceptado por el autómata argumento.

  Esta operación se nota como 'Potency' y se coloca antes del autómata y el número de potencia que queremos. De modo tal que nos queda: Potency α β, donde α es un autómata y $\beta \in \mathbb{N}$.

- Operación Reverso: esta operación unaria nos permite generar un autómata que acepta las cadenas reversas o inversas del autómata argumento.

  Dada una cadena $u \in \Sigma^*$, definimos inductivamente la cadena reversa u^R como:

  $$\lambda^R = \lambda$$
  $$(ua)^R = au^R$$

  donde $a \in \Sigma$ y $u \in \Sigma^*$.

  Esta operación se nota como 'Reverse' y se coloca antes del autómata. De modo tal que nos queda: Reverse α , donde α es un autómata

Ahora bien su orden de precedencia es de mayor a menor:

Clausura Positiva de Kleene(+), Clausura de Kleene(*) y Reverso(Reverse)
Concatenación(Concat)
Unión(Union)

y sus asociatividades son

- Isquierda : Union y Concatenación
- Derecha : Clausura Positiva de Kleene, Clausura de Kleene y Reverso

3 Comandos del intérprete

Existen algunos comandos, los cuales comienzan con ':', que solo pueden usarse en consola una vez se está ejecutando el intérprete:

- :browse : nos permite saber cuáles son los autómatas en scope
- :print <nombre del autómata> : printea en pantalla toda la información de un autómata: estados iniciales , transiciones , estado inicial , etc. Incluso su AST. Cabe aclarar que el autómata tiene que estar en scope

  NOTA : el nombre de los autómatas son cadenas compuestas por únicamente letras(mayúsculas o minúsculas) con una longitud mayor o igual a 1.
- :load <ruta del archivo> : carga un archivo con la extensión '.ap' con comandos(no los del intérprete)
- :reload : intenta volver a cargar el ultimo archivo
- :quit : nos permite salir del intérprete
- :help/:? : nos muestra la lista de comandos

4 Comandos

4.1. Autómatas de pila , gramáticas y expresiones

Antes de empezar a describir los diferentes comandos disponibles, procedemos a detallar algunas cosas:

- Autómata de pila (Ap): un autómata de pila se estructura de la siguiente manera:
  S | Σ | Γ | T | σ | Ac donde
  - S es el conjunto finito de estados del autómata .El nombre de los estados es cualquier cadena (únicamente compuesta por letras mayúsculas o minúsculas) de longitud mayor o igual a 1 y estas se encuentran separadas por ','.
  - Σ es el conjunto finito de símbolos de entrada .Cada símbolo es un único carácter(únicamente letras mayúsculas o minúsculas) y estos se encuentran separadas por ','.
  - Γ es el conjunto finito de símbolos de pila .Cada símbolo es cualquier cadena(únicamente compuesta por letras mayúsculas o minúsculas) de longitud mayor o igual a 1 y estas se encuentran separadas por ','.
    NOTA : se reserva el uso del símbolo # para uso especifico.
  - T es una colección finita de transiciones
    $$T \subseteq S \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \times S$$
    donde las transiciones se estructuran: (a,b,c,d,e). Estas se separan entre sí a través de '-'.
    NOTA : en este caso representamos a λ con '/'
  - σ($\sigma \in$ S)es el estado inicial . Al igual que en S es cualquier cadena(únicamente compuesta por letras mayúsculas o minúsculas) de longitud mayor o igual a 1.
  - Ac(Ac $\subseteq$ S) es el conjunto de estados de aceptación .El nombre de los estados es cualquier cadena(únicamente compuesta por letras mayúsculas o minusculas) de longitud mayor o igual a 1 y estas se encuentran separadas por ','.

  Ejemplo = I,A,B,C | a,b | a,x | (I,/,/,x,A)-(A,a,/,a,A)-(A,/,/,/,B)-(B,b,a,/,B)-(B,/,x,/,C) | I | I,C
  Es importante mencionar que S , Σ y Ac deben ser no vacíos, es decir, tener al menos un elemento.
  Observacion: .
- Gramática (Gi): una gramática independiente de contexto se estructura de la siguiente manera:
  N | T | P | σ
  - N es el conjunto finito de no terminales. Los no terminales son cualquier cadena(únicamente compuesta por letras mayúsculas o minúsculas) de longitud mayor a 1 o un simple carácter en mayúsculas, y estas se encuentran separadas por ',' .
  - T es el conjunto finito de terminales. Cada terminal es un único carácter (únicamente letras minúsculas) y estos se encuentran separadas por ','.
  - P es un conjunto finito de reglas de producción
    $$A \rightarrow \alpha$$
    donde A $\in N$ y $\alpha \in (N \cup T)^*$. Las reglas se estructuran como: A - > α donde
    - A $\in N$
    - α es una colección de terminales y no terminales separados con comas ó el símbolo _ que representa el λ .

Estas se separan entre sí a través de '-'.

- $\sigma(\sigma \in N)$ es el no terminal inicial . Al igual que en N es cualquier cadena(unicamente compuesta por letras mayusculas) de longitud mayor o igual a 1 .

Ejemplo = A,B | a,b | A -> a,B,b - B->_ - B -> a,B,b | A

Es importante mencionar que N, T y P deben ser no vacíos, es decir, tener al menos un elemento.

- Expresión (Exp): una expresión se construye inductivamente de la siguiente manera:

  - <nombre de un autómata> es una expresión (es una cadena compuesta por letras mayúsculas o minúsculas de longitud mayor o igual a 1).
  - Si e1 y e2 son expresiones entonces Concat e1 e2 y e1 Union e2 son expresiones.
  - si e es una expresion entonces *e ,+e ,Reverse e , Potency e n (con n $\in \mathbb{N}$) y (e) son expresiones

  Es importante aclarar que al colocar los nombres de los autómatas en la expresión estos deben estar en scope en ese momento. De no ser así, la expresión no se evaluara.

  Además, aunque se encuentra disponible la operación **Reverse** estas es recomendable utilizarla con mucho cuidado o directamente no hacerlo. Esto se debe a la complejidad lógica que tiene hacer el reverso de un autómata de pila haciendo una tarea titánica el hecho de verificar si una cadena es aceptada por un autómata reverso. Esto último se debe a la cantidad absurda de transiciones que se crean. En caso de que decida usarla es recomendable que utilice el freno de transiciones que está disponible cuando se verifica que un AP acepta una palabra.

El usuario podrá utilizar comentados tanto en consola como en archivos especiales con la extensión '.ap'. Ahora veremos exactamente cuáles son los comandos posibles que tenemos disponibles para utilizar tanto en consola como en los archivos con la extensión '.ap':

Comandos de definicion

- **defA <nombre del autómata> = Ap ;** : nos permite definir un autómata de pila a través de su construcción. El <nombre del autómata> es una cadena compuesta por letras de longitud mayúsculas o minúsculas mayor o igual a 1.
- **defE <nombre del autómata> = Exp ;** : nos permite definir un autómata de pila a través de una expresión. El <nombre del autómata> es una cadena compuesta por letras mayúsculas o minúsculas de longitud mayor o igual a 1.
- **defG <nombre del autómata> = Gi ;** : nos permite definir un autómata de pila a través de una gramática independiente del contexto. El <nombre del autómata > es una cadena compuesta por letras mayúsculas o minúsculas de longitud mayor o igual a 1.

Comandos de ejecucion

- **Exp <cadena> [<numero>] ;** : nos permite dada una expresión verificar que el autómata resultante de su evalucaión acepta la cadena teniendo en cuenta el número. La <cadena> puede estar compuesta por letras mayúsculas o minúsculas de longitud mayor o igual a 1 o es _ que representa la cadena vacía, y el <numero> es un entero positivo.

  NOTA: el número ingresado será utilizado para limitar las recursiones/transiciones al intentar verificar que el autómata producto de la Exp acepta la cadena. Este es opcional y en caso de no colocarlo se utiliza el valor por defecto, 4096.

- **Exp ;** : nos permite dada una expresión evaluarla , construyendo así un autómata de pila y luego este será impreso en pantalla pero no será cargado.

  Siempre que se evalue una expresión ,el autómata resultante tendra sus estados renombrados.

En el caso de que estemos escribiendo en un archivo podremos encadenar múltiples comandos pero cuando se los introduzca por consola solo podemos con uno a la vez.

Observación 1 : es importante aclarar que en el caso de que se quiera definir un autómata con un nombre ya cargado este se sobrescribirá con la nueva definición.

Observación 2 : en el caso de que se llame a un autómata que no está en el scope ,independientemente de que se trate de un printeo en pantalla o un comando que utiliza expresiones, esta se cancelara y se le notificara al usuario.

5 Capasidades adicionales

En particular tanto en el archivo como en consola es posible escribir comentarios ya sea líneas de comentarios utilizando '- -' como bloques de comentarios usando '{- -}', de manera idéntica a los comentarios en Haskell.

Ejemplos:

- - - comentario
- {- comentario -}