Universidad Nacional de Rosario

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Análisis de Lenguajes de Programación

Trabajo Práctico Final

EDSL de Automatas de pila
Informe descriptivo del poryecto

Cavagna, Lucas Gastón

Febrero 2023

Índice general

1 Informe 3

1 Informe

1.1. introducción

En este informe descriptivo del trabajo practico final de Analisis de Lenjuges de Programacion haremos incapie en varios temas:

- Objetivos del proyecto
- Capacidades del EDSL
- Toma de decisiones
- Organizacion de los archivos
- Código Utilizado
- Bibliografía

Cada uno de estos será desarrollado apropiadamente a continuación con el objetivo de dar un una descripción completa del trabajo.
Es inportante aclarar que junto a este infomrme tambien se encuntra un manual de usuario, el cual da una descripcion mas detallada del uso del EDSL creado.

1.2. Objetivos del proyecto

La objetivo principal de este trabajo práctico final es el diseñar un EDSL o lenguaje de dominio especifico embebido.

Un **lenguaje de dominio especıfico (DSL)** es un lenguaje de programación diseñado para resolver problemas de un dominio en particular. Unos ejemplos de DSL's serían el SQL o Latex.

Estos disponen de algunas ventajas , tales como:

- Programas más pequeños, claros e intuitivos.
- Nivel de abstracción mayor.
- Semántica restringida.

Sin embargo también tienen sus desventajas:

- Hay que escribir herramientas específicas (syntax highlighting, testing, documentación, etc.).
- Es tedioso redefinir booleanos, enteros, etc. (y sus operaciones)
- Hay que hacer un parser, type-checker, compilador. Debido a esto, a menudo están ligados a una sola plataforma.

Ahora, un **DSL embebido** aprovecha la infraestructura de un lenguaje anfitrión pre-existente y reúsa, su sintaxis, sistema de tipos, compilador, bibliotecas, herramientas. Pero el diseño queda limitado por las capacidades del lenguaje anfitrión.

En este caso, la inspiración para esto proyecto surge de la necesidad de tener una herramienta que nos permita interactuar con autómatas de pila no deterministas.

Con esto dicho, se planteó que sea posible lo siguiente:

- Poder definir automatas de pila no determinstas
- Poder construir expresiones a partir de autómatas de pila no deterministas y diferentes operaciones , y evaluarlas
- Poder chaquear si la evaluación de una expresión acepta una cadena dada o no

Cabe acalrar que todo el diseño de esste EDSL fue realizado sobre el lenguaje anfitrion Haskell.

1.3. Capacidades del EDSL

Teniendo en cuenta lo visto en la sección 1.2, se le brindaron las siguientes capacidades a este EDSL:

- Es posible definir autómatas de pila de tres maneras diferentes:
  - Utilizando un definición base.
  - A través de una expresión compuesta de autómatas ya definidos y diferentes operaciones.
  - Utilizando una gramática independiente del contexto.
- Se pude construir una expresión compuesta de autómatas ya definidos y diferentes operaciones, y ver el autómata resultante de su evaluación.
- Utilizar una expresión con el objetivo de ver si su evolución acepta una cadena dada o no.

Las operaciones disponibles son las siguientes: unión, concatenación, reverso, potencia, clausura de Kleene, y clausura positiva de Kleene.
Ahora, es importante aclarar que lo lenguajes de tipo 2(que son los que aceptan los autómatas de pila no deterministas y las gramáticas independientes del contexto) no son cerrados bajo las operaciones de intersección, resta y complemento. Es por esto que dichas operaciones no se encuentran disponibles en este EDSL.
Si precisa de más información, se encuentra disponible el manual de usuario que explica más a fondo estas capacidades.

1.4. Sintaxis y semántica

Presentamos la sintaxis abstracta del EDSL de este proyecto:

SingleComm ::= *CommExp*
| *CommDef*
CommDef ::= ’defA’ *ap* ’=’ *AP* ’;’
| ’defE’ *ap* ’=’ *Exp* ’;’
| ’defG’ *ap* ’=’ *GI* ’;’
CommExp ::= *Exp Cadena NaturalsZ* ’;’
| *Exp* ’;’
Exp ::= ’Union’ *Exp Exp*
| ’Concat’ *Exp Exp*
| ’Potency’ *Exp Naturals*
| ’*’ *Exp*
| ’+’ *Exp*
| ’Reverse’ *Exp*
| *ap*

Cabe aclarar que *Naturals* son los $\mathbb{N}$,*NaturalsZ* son los $\mathbb{N} + \{0\}$,*Cadena* es la cadena que se quiere ver que es aceptada por el autómata y que *ap* son nombres de autómatas de pila a definir o definidos.

Ahora una vez presentada la sintaxis abstracta del lenguaje, para definir la semántica de las expresiones entre autómatas de pila, utilizaremos una semántica operacional de paso grande. Los valores de estas expresiones se definen de la siguiente manera:

vap::= Autómatas de pila(ap)

Definimos la relación de evaluación para las expresiones inductivamente mediante las siguientes reglas:

$$\frac{}{vap \Downarrow_{exp} vap}\ \textsc{B-VAp}$$

$$\frac{e_1 \Downarrow_{exp} vap_1 \ e_2 \Downarrow_{exp} vap_2}{e_1 \ \mathbf{Union} \ e_2 \Downarrow_{exp} vap_1 \ \mathrm{U} \ vap_2}\ \textsc{B-Union}$$

$$\frac{e_1 \Downarrow_{exp} vap_1 \ e_2 \Downarrow_{exp} vap_2}{\mathbf{Concat} \ e_1 e_2 \Downarrow_{exp} \mathrm{C} \ vap_1 \ vap_2}\ \textsc{B-Concat}$$

$$\frac{e \Downarrow_{exp} vap}{\mathbf{Reverse} \ e \Downarrow_{exp} \mathrm{R} \ vap}\ \textsc{B-Reverse}$$

$$\frac{e \Downarrow_{exp} vap}{\mathbf{*} \ e \Downarrow_{exp} \mathrm{KS} \ vap}\ \textsc{B-KlenneS}$$

$$\frac{e \Downarrow_{exp} vap}{\mathbf{+} \ e \Downarrow_{exp} \mathrm{KP} \ vap}\ \textsc{B-KlenneP}$$

$$\frac{e \Downarrow_{exp} vap}{\mathbf{Potency} \ e \ \mathrm{n} \Downarrow_{exp} \mathrm{P} \ vap \ \mathrm{n}}\ \textsc{B-Potency}$$

Donde tenemos que KP *ap*, Ks *ap* ,P *ap* n ,R *ap* , C ap_1 ap_2 y ap_1 U ap_2 generan autómatas de pila. Para saber con precisión que hace cada una de las operaciones, puede ver el manual de usuario.

1.5. Toma de decisiones

Durante la creación y diseño de este EDSL se tomaron las siguientes decisiones:

1. La utilización del gestor de proyectos **Stack** para una mejor organización del trabajo practico
2. Usar **Happy** para la creación del parser de este proyecto
3. La estructura utilizada para representar a los autómatas de pila fue la siguiente:
   Ap [String] [Char] [String] [Trans] String [String] donde
   - **Trans** tiene la estructura **T String Char String String String**.De manera que T a b c d e es una transición de a hasta e , leyendo b , sacando de la pila c y poniendo d en la misma

   De modo tal que esta estructura represente la siguiente 6-upla:

   $$(S, \Sigma, \Gamma, T, \sigma, Ac)$$

   donde

   - S es un conjunto finito de estados
   - Σ es un conjunto finito de símbolos de entrada
   - Γ es un conjunto finito de símbolos de pila
   - T es una colección finita de transiciones

     $$T \subseteq S \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \times S$$

   - $\sigma \in S$ es el estado inicial
   - Ac $\subseteq$ S es un conjunto de estados de aceptación

Y se decidió tomar el símbolo '/' para representar la no lectura del carácter de la cadena, la no extracción de la pila y el no agregado a la pila.

4. Se decidió que también se podría trabajar con gramáticas independientes del contexto, permitiendo al usuario definir autómatas con estas gramáticas.

5. La estructura definida para las gramáticas independientes del contexto es:
   Gi [String] [Char] [Rules] String donde

   - **Rules** tiene la estructura **R String [String]**.De manera que (R A α) es una regla de producción tal que:

     $$\mathrm{A} \rightarrow \alpha$$

     donde A $\in N$ y $\alpha \in (N \cup T)^*$.

   De modo tal que esta estructura represente la siguiente 4-upla:

   $$(\mathrm{N},\mathrm{T},\mathrm{R},\sigma)$$

   donde

   - N es un conjunto finito de símbolos llamados no terminales
   - T es un conjunto finito de símbolos, llamados terminales o alfabeto, tal que $N \cup T = \emptyset$
   - Γ es un conjunto finito de símbolos de pila
   - P es un conjunto finito de reglas de producción, donde

     $$P \subseteq ((N \cup T)^* - T^*) \times (N \cup T)^*$$

6. Se limitó la función de ejecución(execute) de los autómatas a una cantidad, la cual puede ser cambiada por el usuario , de recursiones con el objetivo de evitar que el sistema se caiga

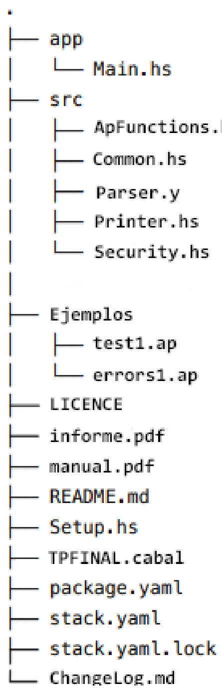
7. Se decidió que la extensión de los archivos que serán levantados tengan la extensión '.ap'

8. Se implementó la capacidad de que el usuario pueda realizar todos los comandos (sin incluir los del interprete) tanto en consola como en los archivos con la extensión .ap

9. Debido a las implicaciones que tiene construir el reverso de un autómata de pila, se le advirtió al usuario en el manual.

1.6. Organización de los archivos

Este proyecto tiene la siguiente disposición de archivos:



```
.
├── app
│   └── Main.hs
├── src
│   ├── ApFunctions.hs
│   ├── Common.hs
│   ├── Parser.y
│   ├── Printer.hs
│   └── Security.hs
│
├── Ejemplos
│   ├── test1.ap
│   └── errors1.ap
├── LICENCE
├── informe.pdf
├── manual.pdf
├── README.md
├── Setup.hs
├── TPFINAL.cabal
├── package.yaml
├── stack.yaml
├── stack.yaml.lock
└── ChangeLog.md
```

- En el directorio *app* se define el módulo Main , que implementa el ejecutable final.
- En el directorio *src* se encuentran los módulos que componen este proyecto:
  - **Common** contiene los tipos de datos que se comparten entre los diferentes modulos.
  - **Printer** modulo que implementa la impresión en pantalla.
  - **Security** modulo que nos brindan funciones de control.
  - **Parser.y** define el parser utilizado para el EDSL. Esto fue construido utilizando **Happy**.
  - **ApFunctions** define todas las operaciones necesarias para la utilización del EDSL.
- En el directorio Ejemplos se encuentran algunos archivos .ap con ejemplos funcionales y otros que prueban diferentes errores. Aquí también es posible agregar más archivos para su posterior uso.
- Los archivos de texto informe.pdf y manual.pdf contienen este archivo y el manual de usuario del EDSL respectivamente.
- El resto de los archivos son de configuración del proyecto.

1.7. Codigo utilzado

Es importante declarar que el uso del contenido del archivo Main.hs del trabajo practico 3 de la materia Análisis de Lenguajes de Programación para construir el intérprete disponible en este proyecto.

1.8. Bibliografia

- Teoría de la computación - J. Glenn Brookshear
- AP_print.pdf de LFyC -Profesor Verde , Pablo
- lfyg_print.pdf - Profesor Verde , Pablo
- Introduction to Theory of Computation -Anil Maheshwari-Michiel Smid