Índice general

Índice de figuras

Índice de cuadros

Capítulo 2

Formalismos

2.1. Objetivos

El Proyecto Fin de Carrera `gp1990c` gira en torno a dos conceptos:

I. **Desarrollar el estándar ISO Pascal 7185:1990**[1] **además de la construcción de un prototipo** para su parte léxica (basada en Flex[2]) y su parte sintáctica (basada en Bison[3]).

II. **Síntesis y Lenguaje Matemático propio de la Teoría de Lenguajes de Programación** así como su evolución e influencias históricas.

2.2. Descripción general del proyecto

El Software estará compuesto por dos elementos atómicos desde el punto de vista funcional pero que se interconectan para constituir, como hemos dicho el analizador.

Brevemente enumeraremos sus partes:

I. Analizador Léxico (`gp1990la`): Es el elemento encargado de verificar que el conjunto de palabras de código fuente pertenecen al lenguaje. Genera un fichero `lex.yy.c`.

II. Analizador Sintáctico[4]. (`gp1990sa`): Es el elemento encargado de comprobar que el orden de esas palabras corresponde a la propia sintaxis (Reglas) del lenguaje. Genera un fichero `y.tab.c`

Compilación: El proceso para crear un ejecutable a partir de código Flex/Bison[5] sería:

```
$ bison -yd gp1990sa.y
$ flex gp1990la.l
$ gcc y.tab.c lex.yy.c {lfl {o programa
```

[1] http://www.moorecad.com/standardpascal/
[2] http://flex.sourceforge.net/
[3] http://www.gnu.org/software/bison/
[4] En inglés se denomina "parser"
[5] Compatible con Lex/Yacc.

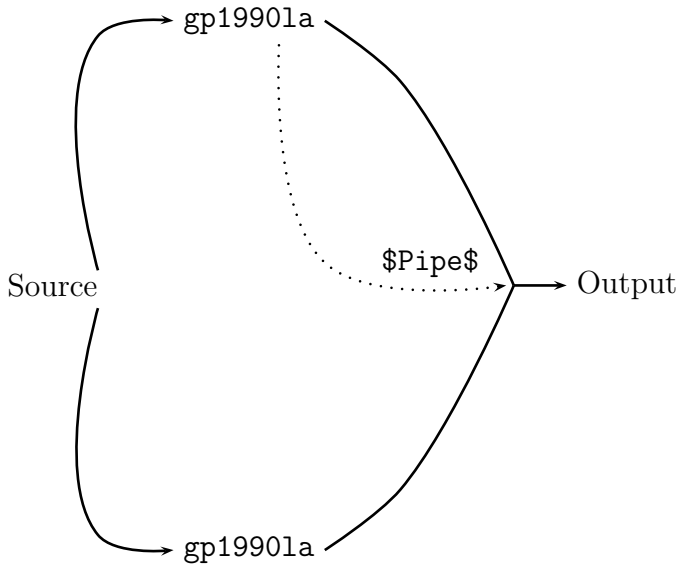

Figura 2.1: Síntesis y partes de `gp1990c`.

2.3. El proceso de transformación de una expresión regular en Software

Dicho proceso comprende una serie de etapas:

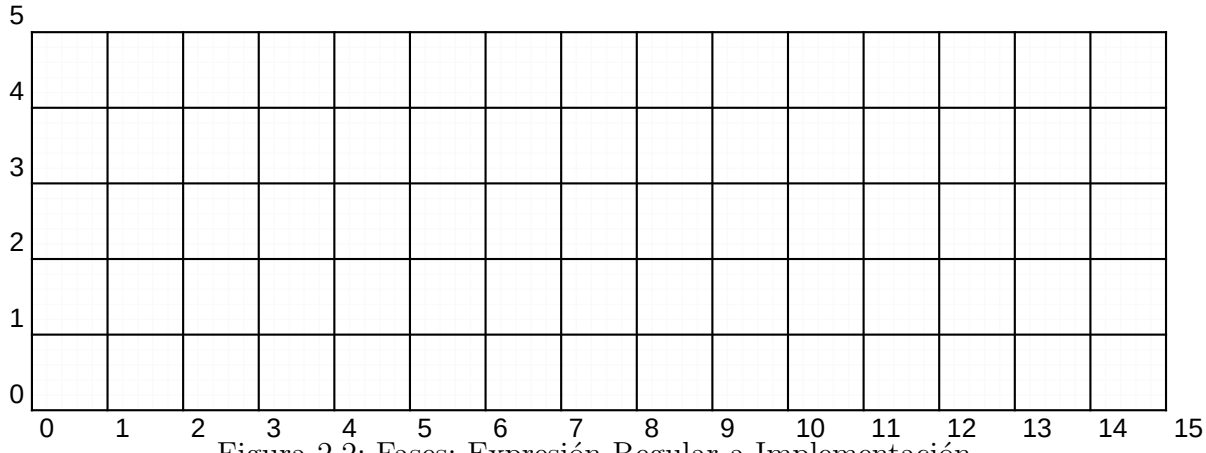

Figura 2.2: Fases: Expresión Regular a Implementación.

Definición 2.3.1. Expresión Regular: Se trata de una simplificación de una cadena de caracteres. Ver sección (**??**).

Definición 2.3.2. AFND (Autómata Finito no Determinista): Reconocedor de expresiones regulares con transiciones δ del tipo:

$$\delta(i, o) \rightarrow q; \backslash e \in Q \wedge s \in \Sigma \cup \{\lambda\} \wedge q \subset Q \tag{2.1}$$

Definición 2.3.3. AFD (Autómata Finitio Determinista): Reconocedor de expresiones regulares con transiciones δ del tipo:

$$\delta(i, o) \rightarrow q_i; \backslash e \in Q \wedge s \in \Sigma \cup \{\lambda\} \wedge q_i \subset Q \tag{2.2}$$

Definición 2.3.4. Minimización de los estados: Dicho proceso se basa en el siguiente teorema:

Teorema 2.3.5. *Para cualquier autómata finito, existe un autómata finito mínimo equivalente*[6].

Implementación: La implementación y desarrollo de un analizador depende en gran medida de tipo de lenguaje base. Existen la siguiente clasificación para el análisis de Gramáticas Libres de Contexto[7]

2.4. Breve descripción de gp1990la

Definición 2.4.1. `gp1990la` **en un Analizador Léxico para ISO Pascal 7185:1990**.

Implementación: Hay tres tipos de implementaciones para un Analizador Léxico:

I. Implementación Software con Lenguaje de Alto Nivel: Se programa el analizador con un lenguaje que permita rutinas de bajo nivel, normalmente Lenguajes C y C++.

II. Implementación en Lenguaje Ensamblador: Código "a priori" nativo para una determinada arquitectura.

III. Lex: Se trata de un programa que se adapta a las necesidades de un alfabeto y es capaz de reconocer y ordenar tokens.

Implementación	**Eficiencia**	**Velocidad**	**Portabilidad**
Aplicación Lex	Regular	Regular	Óptima
Código C	Buena/Muy Buena	Buena/Muy Buena	Óptima
Código C++	Buena	Buena/Muy Buena	Buena/Muy Buena
Código Ensamblador	Muy Buena	Óptima	Muy Mala

Cuadro 2.1: Comparativa para los distintos tipos de implementaciones para un AL.

Nota: Las formalidades que describen a un Analizador Léxico se tratan con más detalle en el: Capítulo (**??**).

[6]Falta cita.

[7]Gramática chomskiana de Tipo 2: P = $\{(S \rightarrow \lambda) \vee (A \rightarrow p_2) \mid p_2 \in \Sigma^+; A \in N\}$

2.5. Breve descripción de gp1990sa

Definición 2.5.1. `gp1990sa` **en un Analizador Sintáctico para ISO Pascal 7185:1990**.

Definición 2.5.2. La gramática de ISO Pascal 7185:1990 se presenta en notación Backus-Naur Form (BNF) que se describe como $G = \{\sigma_T, \sigma_N, S, P)$

Dónde:

I. σ_T es el conjunto de los símbolos terminales.

II. σ_N es el conjunto de los símbolos no terminales.

III. S es el símbolo inicial de la gramática.

IV. P es el conjunto de las reglas de producción.

Definición 2.5.3. El análisis sintáctico descendente (ASD) trata de encontrar en las producciones de una gramática determinada la derivación por la izquierda del símbolo incial para un conjunto de caracteres de entrada.

Definición 2.5.4. La finalidad de `p1990sa` o Analizador Sintáctico es la de certificar que las palabras del lenguaje se organizan de acuerdo a una estructura.

Corolario 2.5.5. *Nuestro análisis será predictivo*

Definición 2.5.6. El analizador predictivo se encarga de anticipar la regla a aplicar tomando como entrada el primer símbolo de una estrcutura.

Corolario 2.5.7. *Con ello conseguimos una complejidad lineal:* θn. Ver Capítulo (**??**).

Definición 2.5.8. Las gramáticas que son analizadas sintácticamente de forma descendente predictivo con un único símbolo de entrada pertenecen al grupo LL(1).

I. Analizador Sintáctico Descendente (Top-Down-Parser): Se basa en analizar una gramática a partir de cada símbolo no terminal (es una desarrollo de arriba hacía abajo). Son formalmente denominados Analizadores LL.

II. Analizador Sintáctico Ascendente (Bottom-Up-Parser): Analizan la gramática a partir de los símbolos terminales (por ello es un desarrollo de abajo hacía arriba). Son formalmente denominados Analizadores LR.

III. Yacc: A partir de una gramática no ambigua (aunque también acepta gramáticas ambiguas con resolución de problemas) genera un autómata tipo LALR (analizador sintáctico LR con lectura anticipada).

2.6. Métodos y Fases de desarrollo

El proyecto se construirá siguiendo el modelo clásico de **Ciclo de desarrollo en Cascada**[8]: Análisis, Diseño, Codificación y Pruebas.

I. Análisis: Consistirá en un estudio sobre los fundamentos matemáticos de los compiladores (con especial énfasis en el Lenguaje de Programación Pascal) además de una contextualización y evolución de los compiladores.

II. Diseño: Sobre la base teórica antes descrita, se hará un estudio teórico-práctico sobre las herramientas Lex/Yacc cara a la especificación de los prototipos: `gp1990la` y `gp1990sa`.

III. Codificación: Para las herramientas Flex/Bison:

   i. `gp1990la.l`: Fichero de especificación léxica. Será un modelo funcional que incluirá todo lo necesario para realizar programas sencillos.

   ii. `gp1990sa.y`: Fichero de especificación de las reglas sintácticas.

   iii. GNU Build System (Autoconf[9]): Ficheros `makefile.am` y `configure.ac` con el objetivo mejorar la compatibilidad de la herramientas con otras familias UNIX (principalmente GNU/Linux y BSD) además de ser una potente ayuda para futuras correcciones y mejoras

IV. Pruebas: Usando GNU Pascal[10] (`gpc`) y Free Pascal[11] (`fpc`) se realizará una batería de pruebas sobre las partes léxica y sintáctica basadas en algoritmos clásicos:

   i. Algoritmos de Ordenación: Selección Directa, Inserción Directa, Intercambio Directo, Ordenación Rápida (*Quick Sort*) y Ordenación por Mezcla (*Merge Sort*),

   ii. Algoritmos de Búsqueda: Búsqueda Secuencial, Búsqueda Secuencial Ordenada y Búsqueda Binaria.

Fase	**Tiempo (Meses)**
Análisis	3 Meses
Diseño	4 Meses
Codificación	2 Meses
Pruebas	2 Meses

Cuadro 2.2: Relación de desarrollo del proyecto en meses.

[8] Debido al contexto del proyecto se omite la fase de Mantenimiento.
[9] http://www.gnu.org/software/autoconf/
[10] http://www.gnu-pascal.de/gpc/h-index.html
[11] http://www.freepascal.org/

2.7. Entorno de desarrollo

I. GNU/Linux: Sistema Operativo base (Gentoo GNU/Linux[12]) para el desarrollo del Software y la documentación. La elección de GNU/Linux se debe principalmente a la plena compatibilidad con las herramientas de desarrollo tanto del Software como de la documentación (escrita con LaTeX 2_ε). También es resaltable el hecho de que es compatible con otras familias UNIX como BSD.

II. BSD: Principalmente usaremos la versión FreeBSD[13] (derivado de BSD-Lite 4.4) para mejorar la compatibilidad del Software. Se usará especialmente para configurar y ajustar las herramientas GNU Build System así como la pruebas de estabilidad y optimización del código fuente.

III. GCC[14]: Metacompilador que nos servirá para generar programas ejecutables.

IV. GNU Build System: Conjunto de herramientas:

   i GNU Autoconf: Se trata de una herramienta de propósito general para generar ficheros ejecutables para distintas versiones de UNIX. Usa: `configure.ac` y `makefile.in` para generar `makefile` sobre el entorno.

   ii GNU Automake: Autogenera el fichero `makefile.in` a partir de las especificaciones de `makefile.am` necesario para Autoconf.

   iii GNU Libtool: Se trata de una herramienta que genera bibliotecas estáticas y dinámicas para las distintas versiones de UNIX.

V. Flex[1]: Flex (Fast Lexical Analyzer Generator) se trata de un programa para el análisis léxico de Lenguajes Regulares (versión GNU de Lex). Internamente es un Autómata Finito Determinista (AFD).

VI. Bison[2]: Se trata de una analizador sintáctico (versión GNU de Yacc) para Gramáticas Libres de Contexto (también es capaz de generar código para algunos tipos de Gramáticas Ambiguas). Se trata de una analizador que genera un autómata LALR para los Lenguajes C, C++ y Java (principalmente).

VII. TeX Live 2011: Es la Metadistribución de TeX común para sistemas GNU. Contiene todos los paquetes oficiales propuestos por TeX Users Group[15]. Se usarán además los entornos PStricks y MetaPost para la generación de gráficos vectoriales.

[12] http://www.gentoo.org/
[13] http://www.freebsd.org/
[14] http://gcc.gnu.org/
[15] http://tug.org/

Notas del capítulo

[1]**Flex (Fast Lexical Analyzer Generator)**:

I. Desarrollador: Vern Paxson.

II. Última versión estable: 2.5.37 (3 de Agosto de 2012)

III. Tipo de sistema base: UNIX y clones.

IV. Licencia: Licencia BSD.

V. Página Web: http://flex.sourceforge.net/

[2]**Bison (GNU Bison)**:

I. Desarrollador: Proyecto GNU.

II. Última versión estable: 3.0 (26 de Julio de 2013)

III. Tipo de sistema base: UNIX y clones.

IV. Licencia: Licencia GPL.

V. Página Web: http://www.gnu.org/software/bison/

Bibliografía

[ea07] Alfred V. Aho et al. *Compilers: Principles, Techniques and Tools 2ed*. Pearson Education, 2007.

[ISO91] ISO/IEC. Pascal ISO 90 (ISO/IEC 7185:1990). *ISO/IEC*, 1991.

[Roj10] Miguel Ángel Quintans Rojo. Apuntes de la asignatura: Lenguajes de Programación. *Ingeniería Técnica en Informática de Gestión. Universidad de Alcalá*, 2010.