Desarrollo de Software de Apoyo a la Enseñanza de la Teoría de Lenguajes y Autómatas y al Diseño de Compliladores

Jorge Aguirre
E-mail: jaguirre@reinfo.edu.ar

Marcelo Arroyo
E-mail: marroyo@exa.unrc.ar

Valentina Grinspan
E-mail: vgrinspan@exa.unrc.edu.ar

Ruta 8 Km. 603. Área de Computación. Fac. de Cs. Exactas Fco.-Qcas. y Nat.
Universidad Nacional de Río Cuarto

Resumen. El proyecto aquí descripto fue realizado por la cátedra de Autómatas y Lenguajes de la UNRC y propone la creación gradual de una biblioteca de herramientas de software para resolver los problemas decidibles estudiados en la asignatura, tales como la transformación de un NFA en un DFA, la reducción de autómatas finitos, la decisión sobre la vacuidad, finitud o equivalencia de lenguajes regulares, entre otros.
En la actualidad se dispone de herramientas estándar que no resuelven pasos intermedios ni abordan casos que si bien no se presentan frecuentemente en las aplicaciones, sí tienen importancia desde el punto de vista didáctico. El proyecto tiene, por lo tanto, una doble finalidad: por un lado proponer a los alumnos el desarrollo de las herramientas mencionadas a través del desarrollo de módulos de software que implementen los modelos y las soluciones estudiados en la asignatura, y por otro enriquecer el trabajo práctico de los alumnos a medida que se va disponiendo de las herramientas desarrolladas, ya que con su asistencia se pueden abordar problemas más complejos e insumir menos tiempo en cálculos rutinarios.

En cuanto a los objetivos del proyecto referentes al proceso de enseñanza-aprendizaje, se pretende que el alumno comprenda profundamente la retroalimentación entre el desarrollo teórico y el avance en el campo de las aplicaciones, que sea capaz de transformar algoritmos complejos desde una definición de muy alto nivel hasta su implementación en computadora, y que sea capaz de resolver problemas complejos asistiéndose en el uso de herramientas existentes.

1. Introducción

El proyecto aquí descripto fue desarrollado en la cátedra de Autómatas y Lenguajes de la Licenciatura en Ciencias de la Computación de la UNRC. La asignatura mencionada pertenece al primer cuatrimestre del cuarto año y en el plan hay otra asignatura Compiladores que actualmente es un taller de diseño de Compiladores.

El proceso de desarrollo de software ha ido resolviendo problemas cada vez más abstractos, planteados gradualmente en un nivel más elevado respecto del ambiente de la aplicación, mientras que paralelamente ha ido creciendo la proporción y el grado de abstracción de los problemas que son directamente resueltos por herramientas de software. Lejos de apartarse de estas características generales, la problemática ligada a los lenguajes de programación, es un ejemplo arquetípico.

Así en un determinado momento, para analizar y resolver problemas de búsqueda de cadenas que cumplan determinados patrones se procedía directamente buscando en forma heurística el diseño de la solución. Luego se vio que un modelo abstracto, los autómatas finitos servían para razonar con mayor claridad y para construir las implementaciones más eficientes posibles de solución. A partir de ese momento se comenzó a usar este modelo como instrumento conceptual de trabajo. Poco después, para especificar estos autómatas se dio un modelo conceptual de uso mucho más claro que los autómatas mismos y se dieron herramientas de transformación que permitían obtener el mejor de los autómatas factibles para resolver un problema dado. Poco después, construídas las herramientas de software necesarias se pudo resolver los problemas originales especificándolos en el modelo más simple y obteniendo la implementación más eficiente automáticamente. El proceso de implementacíon de estas últimas herramientas resultaba, por cierto, largo y costoso pero pronto surgieron a su vez otras herramientas de nivel superior capaces de generar automáticamente implementaciones de herramientas de una familia de problemas mucho más general y que comprendía a las anteriores, usando un modelo conceptual más general, el de los lenguajes determinísticos y la traducción guiada por sintaxis.

La asignatura *Autómatas y Lenguajes* centra sus contenidos en algunas de las teorías que han sustentado este proceso de generalización abarcativa del proceso de desarrollo de software antes mencionado. Durante su transcurso el alumno estudia los modelos conceptuales que sirven para la resolución de ciertos problemas y las implementaciones prácticas de las soluciones obtenidas por ellos usando herramientas cada vez más generales. En la actualidad la cátedra dispone de herramientas estándar para la construcción de software para resolver los problemas más usuales en el desarrollo de aplicaciones, pero no de herramientas que resuelvan los pasos intermedios o que permitan abordar casos que si bien no se presentan frecuentemente en las aplicaciones, sí tienen importancia desde el punto de vista didáctico.

El proyecto aquí descripto consiste en la construcción, en el marco del desarrollo del cursado de la asignatura, de un conjunto básico de las herramientas arriba mencionadas. El proyecto tiene una doble finalidad: por un lado obtener los correspondientes productos de software, por otro, enriquecer el proceso de enseñanza aprendizaje con una actividad de desarrollo de módulos de software que implementan los modelos y soluciones estudiados en la asignatura.

Formación previa de los alumnos

Los alumnos que habrían de participar en esta experiencia contaban con los siguientes conocimientos y capacidades previamente adquiridos:

Conocimientos de diseño de algoritmos, tipos abstractos de datos y estructuras de información. Solvencia en el uso de PASCAL, que fue el primer lenguaje utilizado y sobre el que trabajaron los dos primeros años de la carrera. Conocimientos sobre los modelos de

ejecución que requieren los distintos tipos de lenguajes de programación, sobre alcance de variables, ligadura (binding), representación y verificación de tipos. Conocimientos básicos sobre programación concurrente y los constructores de alto y bajo nivel usados. Somera práctica de uso de lenguaje C. Abundante práctica de uso de MS-DOS. Conocimientos básicos de los paradigmas de programación existentes.

2. Descripción del proyecto.

El proyecto propone la creación gradual de una biblioteca de herramientas de software o caja de herramientas (*tool box)* para resolver los problemas decidibles estudiados en la asignatura. Eventualmente en un futuro también se podría asistir a la heurística de búsqueda de solución para algunos no decidibles.

El proyecto tiene una doble finalidad. Por un lado a medida que se va disponiendo de herramientas, el trabajo práctico de los alumnos se ve enriquecido, ya que pueden abordarse problemas más complejos con su asistencia e insumir menos tiempo en cálculos rutinarios. Por otro lado, luego de un cuidadoso trabajo de diseño y modularización realizado por la cátedra, se propuso la participación de los alumnos en la implementación de las nuevas herramientas. Con esto se logró que los ejercicios de implementación realizados durante los trabajos prácticos ganen el interés de la utilidad de la tarea que pasará a integrar la biblioteca y se experimente la utilidad práctica de muchos resultados teóricos.

Es de destacar que el resultado del proyecto podría ser de utilidad general, por lo cual se tratará de difundir sus resultados y distribuir el producto obtenido.

2.1 Objetivos

Objetivos generales de la asignatura en la que se inserta el proyecto

- Conocimientos básicos sobre la teoría de lenguajes y autómatas.
- Capacidad de trabajo con formalismos, demostración de propiedades y obtención de conclusiones.
- Conocimientos de los algoritmos y técnicas usadas en el análisis sintáctico.
- Habilidad en la transformación de gramáticas y en la construcción de definiciones guiadas por sintaxis.
- Conocimiento de las herramientas y técnicas usadas en la construcción de compiladores.

Objetivos específicos del proyecto

La doble finalidad del proyecto conduce a la búsqueda de dos tipos de objetivos diferentes:

a) los objetivos del desarrollo.
b) los objetivos del proceso de enseñanza aprendizaje.

a) Objetivos del desarrollo
a.1) Objetivo general:

Contar con una biblioteca general de procedimientos que permitan resolver problemas de la Teoría de Lenguajes Formales y Autómatas, y Construcción de Compiladores.

a.2) Objetivos específicos de esta primera etapa:

- Implementación del Tipo Abstracto Relación de un conjunto finito en sí mismo.
- Implementación de un editor de Autómatas Finitos.
- Implementación de la transformación de un Autómata Finito no determinístico en uno determinístico.
- Implementación del cómputo de la relación de indistinguibilidad de estados.
- Implementación de la reducción de Autómatas Finitos.
- Implementación de un módulo de animación de Autómatas Finitos que permita visualizar su comportamiento frente a una entrada dada.
- Implementación de generadores de analizadores sintácticos para los métodos LL(1), precedencia simple y precedencia de operadores.
- Implementación de módulos que permitan decidir, sobre lenguajes regulares, los problemas de:
    - vacuidad,
    - finitud y
    - equivalencia

b) Objetivos específicos del proyecto referentes al proceso de enseñanza aprendizaje

- Comprensión profunda de la retroalimentación entre el desarrollo teórico y el avance en el campo de las aplicaciones.
- Capacidad de transformar algoritmos complejos desde una definición de muy alto nivel hasta su implementación en computadora.
- Experiencia de desarrollo de software según la disciplina que impone el trabajo dentro de un equipo.
- Capacidad de resolución de problemas complejos asistiéndose en el uso de herramientas existentes.
- Capacidad de producir documentación de software.

3. Plan de Trabajo

1. Definición general de los módulos.
2. Especificación funcional de cada módulo.
3. Especificación de las interfaces.
4. Diseño e implentación de cada módulo.
5. Verificación de funcionamiento de los módulos. Integración de los módulos en herramientas.
6. Aplicación de los módulos desarrollados en la realización de los trabajos prácticos.

4. Diseño

Las diferentes herramientas desarrolladas y por desarrollar pueden clasificarse en dos grandes grupos: herramientas para lenguajes regulares y herramientas para lenguajes libres de contexto.

Se realizó un diseño orientado a objetos para aprovechar al máximo la reusabilidad de

componentes de software y aprovechar las relaciones inherentes del sistema en sí. Así por ejemplo un autómata finito determinístico (AFD), un autómata finito no determinístico (AFND-λ) y un autómata pila (PDA), pertenecen todos a la clase abstracta autómata (ver Fig. 2).

4.1 Herramientas para Lenguajes Regulares (HReg)

De las herramientas para lenguajes regulares descriptas en los objetivos específicos detallados en la sección 2, restan implementar las utilidades referidas a expresiones regulares y conjuntos regulares, como así también los problemas de decisión acerca de la vacuidad, finitud y equivalencia.

4.1.1 Módulos del Sistema

Se desarrolló una aplicación, llamada HReg, que permite definir lenguajes regulares mediante los distintos formalismos conocidos (autómatas finitos, gramáticas regulares, conjuntos regulares, expresiones regulares) y realizar operaciones y transformaciones sobre ellos.

Esta aplicación contiene los módulos que muestra la Fig.1, en donde se indican además las relaciones de uso entre dichos módulos.

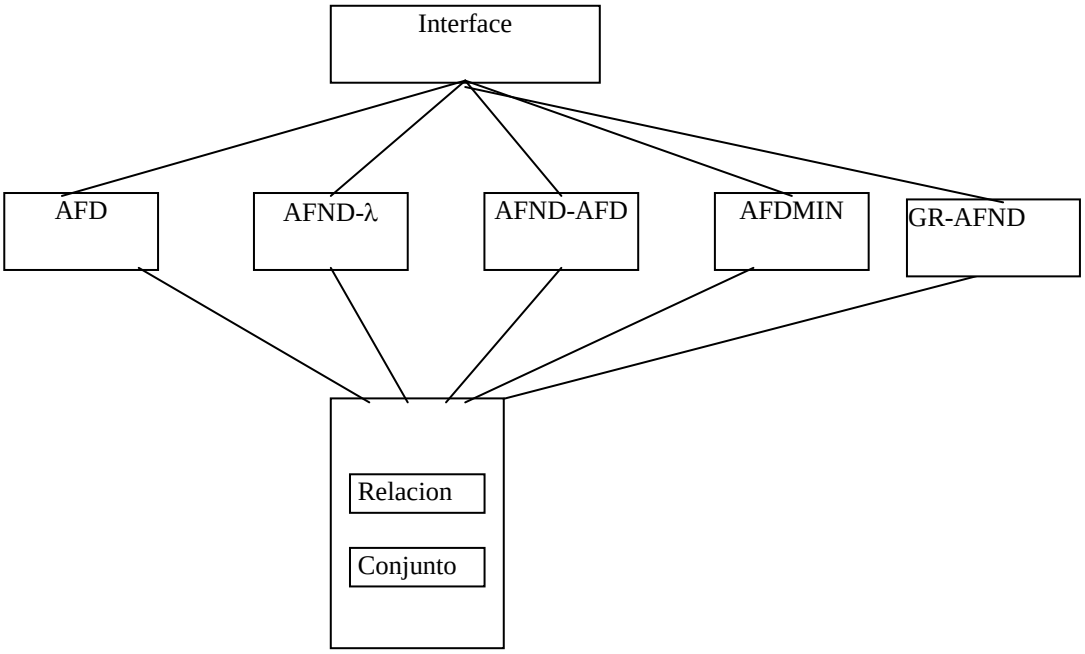


Figura 1. Relaciones de uso de módulos.

A continuación se describe, brevemente, el comportamiento de los módulos que componen el sistema.

Tipos de Datos Abstractos de Utilidad: Relación y Conjunto

Los algoritmos desarrollados para este conjunto de herramientas necesitan operar con relaciones y conjuntos. Por ello se diseñaron tipos abstractos de datos (implementados en C++) de utilidad para todos los módulos. Se implementó una clase Relación, que encapsula todas las operaciones necesarias sobre relaciones, como también una clase Conjunto que permite almacenar objetos heterogéneos y realizar las operaciones de pertenencia, unión, intersección y diferencia de conjuntos. Además permite seleccionar un elemento no

marcado (no seleccionado con anterioridad) y marcarlos (indicar que yá fue seleccionado).

Todos los módulos que realizan operaciones con relaciones o conjuntos utilizan estas clases.

Clase Autómata

Esta clase es una clase genérica (no hay instancias de esta clase). Sirve solamante como clase base para derivar los diferentes tipos de autómatas (AFD, AFND-λ, etc). Esta clase contiene los métodos (virtuales) básicos comunes a todos los tipos de autómatas, definiendo su interface básica.

Módulo AFD

La clase AFD representa un autómata finito determinístico. Contiene la matriz de transición (Estado $\times$ Alfabeto $\rightarrow$ Estado), los métodos que hacen a su definición (número de estados, conjunto de estados finales, función de transición, etc), los métodos que luego permiten ejecutar el autómata (realizar movimientos) y verificar si acepta una cadena de entrada o no, y por último, el método que permite minimizar el autómata.

Módulo AFND

Implementa un autómata finito no determinístico con transiciones vacías. Es similar a AFD con la diferencia de que su matriz de transición (Estado $\times$ Alfabeto $\rightarrow$ ConjEstados) y los métodos de ejecución del autómata implementan el no determinismo usando la técnica de backtracking.

Módulo AFND-AFD

Este módulo implementa el algoritmo de transformación de un AFND-λ a un AFD equivalente. Para ello utiliza el módulo Conjunto.

Módulo AFDMin

Este módulo implementa el algoritmo de reducción de un AFD. Toma como entrada un objeto de la clase AFD y produce como salida otro objeto de la misma clase que representa un AFD mínimo (con la mínima cantidad de estados) equivalente.

Módulo GR-AFDN

Permite definir una gramática regular (GR) y generar un AFND-λ equivalente.

Interface

Todos los módulos son accedidos desde un subsistema principal que implementa la interface con el usuario, creando objetos y enviando mensajes para realizar todos las operaciones descriptas a continuación.

- Definir un AFD, AFND-λ o una gramática regular.
- Obtener un AFD a partir del AFND-λ.
- Obtener un AFND-λ a partir de la gramática regular
- Ejecutar el AFD (de a un paso o que consuma toda la cadena).
- Minimizar (reducir) el AFD.
- Realizar pruebas de aceptación de cadenas.

En cada transformación el autómata resultante se visualiza en una nueva ventana. De este modo, el usuario puede partir desde una definición e ir realizando transformaciones hasta obtener el resultado deseado. Por ejemplo, puede partir de una especificación de una gramática regular y obtener el AFND-λ equivalente a partir del cual puede obtener el AFD equivalente y por último generar el AFD mínimo.

La implementación de esta interface gráfica de usuario está basada en V[1], una jerarquía de clases C++ portable, desarrollada por Dr. Bruce E. Wampler de la Universidad de Nuevo Mexico, Estados Unidos. Esto permite que el código de todo el toolbox sea portable a otros ambientes operativos.

4.1.2 Definición de clases

El sistema consiste de un conjunto de clases que define a los distintos mecanismos de definición de lenguajes regulares. Así se definen las clases AFD, AFND-λ, GR (gramáticas regulares), ER (expresiones regulares) y CR (conjuntos regulares). Además, las clases Conjunto y Relación son las clases de utilidad ya mencionadas, y V es la jerarquía de clases que implementa el GUI (Graphical User Interface). Finalmente, la clase CAST contiene los métodos de conversión de objetos AFND-λ a AFD, GR a AFND-λ, ER a AFND-λ, y de CR a ER.

La Fig. 2 muestra la jerarquía de clases que constituye el sistema y las relaciones de uso entre clases.

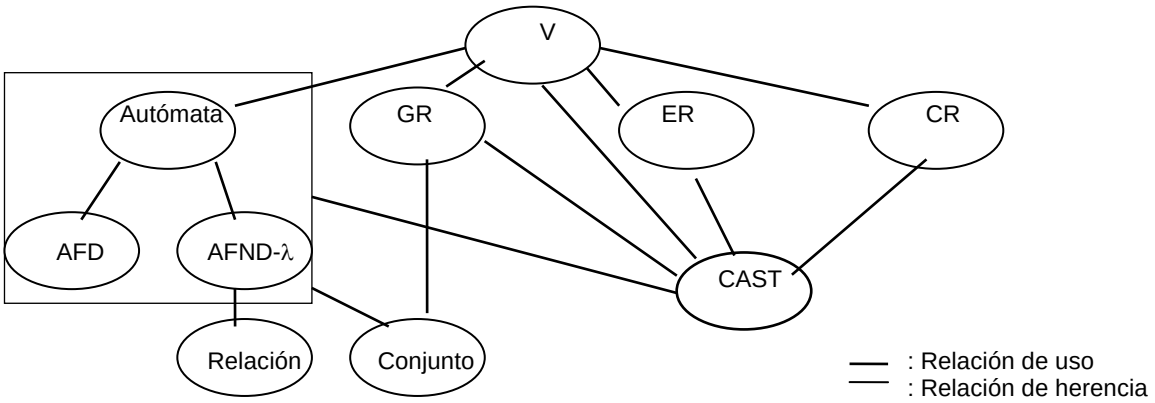


Figura 2. Jerarquía de clases de HReg

4.2 Herramientas para Lenguajes Libres de Contexto

De la caja de herramientas para lenguajes libres de contexto sólo se implementó el generador de parsers de precedencia simple, y resta la implementación de analizadores sintácticos de precedencia de operadores y de gramáticas LL(1) para el próximo dictado de la materia. El diseño del generador de parsers de precedencia simple se describe a continuación.

Generador de Analizadores de Precedencia Simple (PS)

[1]Se puede transferir V por ftp desde la dirección ftp.cs.unm.edu/pub/wampler.

Dada una gramática GP independiente de contexto y reducida esta herramienta permite:

Determinar si GP es de Precedencia Simple

si no lo es visualizar las relaciones de precedencia y los conflictos por los cuales no pertenece a la clase citada.

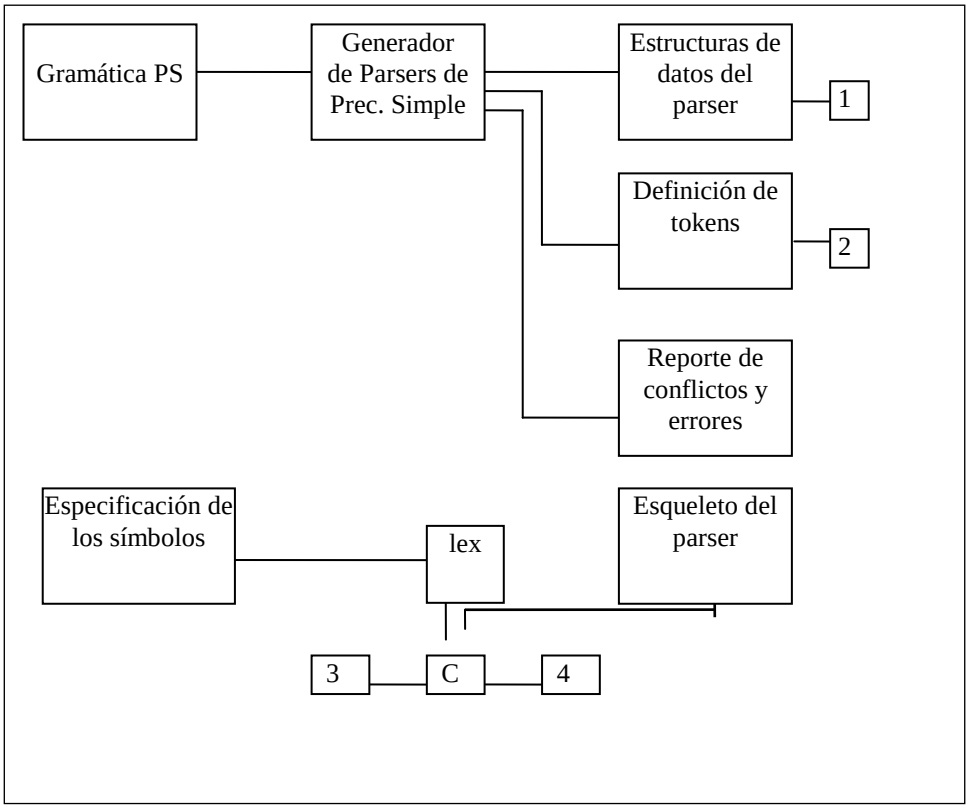
si es de PS visualizar las relaciones y generar un analizador o parser de precedencia simple para el lenguaje que genera (*L*(GP)). El analizador producido utilizará a su vez, un Analizador Léxico que debe escribir el usuario. El Analizador Léxico puede ser especificado usando **lex.**

Realizar el análisis sintáctico de las cadenas del lenguajes mediante el Analizador Sintáctico citado.

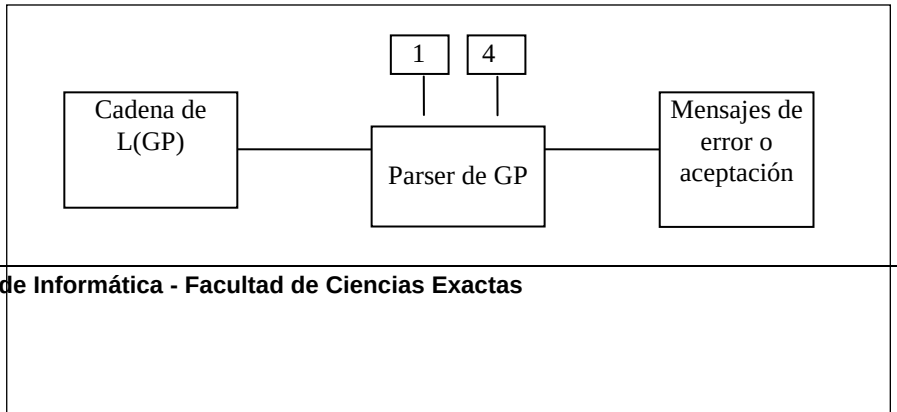
La gramática se especifica con una sintaxis similar a la requerida por yacc.

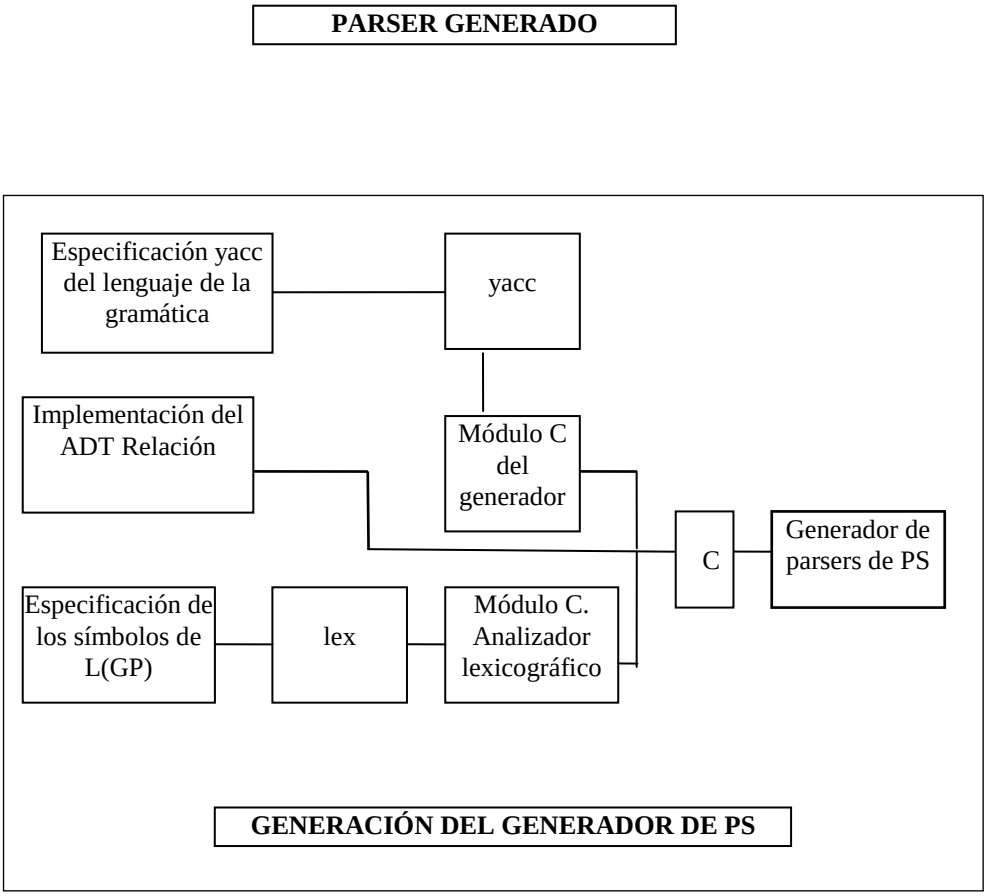
Se provee un mecanismo transparente de referencia de tokens similar al implementado por **yacc.**

El diseño del analizador de precedencia simple responde al siguiente diagrama:



GENERADOR DE PARSERS ESPECIFICADO POR EL USUARIO





5 Desarrollo del proyecto

Se realizó una división del trabajo en distintos grupos. La cantidad de alumnos permitió conformar 5 grupos y cada uno de ellos se dedicó a la implementación de un módulo del sistema. Los grupos estuvieron formados por dos y tres integrantes.
La cátedra realizó el diseño de las interfaces de las clases a implementar en esta herramienta para que cada grupo de trabajo ya tuviese desde el primer momento de desarrollo las interfaces necesarias.
En esta primera etapa del proyecto se fijó como fecha límite el final del cuatrimestre, en el cual todos los grupos tuvieron que entregar su parte.
Los grupos tuvieron que interactuar entre sí para realizar las pruebas de software. Por ejemplo, el grupo que implementó la transición de AFND-λ a AFD tuvo que interactuar con el grupo que implementó la clase AFD para verificar que el AFD obtenido era correcto. Esta interacción obligó a cada grupo a terminar su parte a tiempo para no trabar el desarrollo de otros grupos dependientes.

6 Resultados alcanzados

Los resultados alcanzados en esta primera etapa fueron satisfactorios. Los alumnos

necesitaron conocer muy bien, desde el primer momento, los formalismos y los algoritmos de transformación, ya que sin estos conocimientos estaban imposibilitados a realizar su implementación.

Los alumnos se encontraron con algunas dificultades que hicieron que el tiempo de desarrollo fuese algo mayor que el esperado. Algunas de esas dificultades fueron las siguientes:

- Conocimiento básico de programación orientada a objetos, ya que anteriormente sólo habían desarrollado pequeños programas.
- Escasa experiencia de trabajo en grupo para desarrollo de un producto final.
- Tiempo de aprendizaje de uso de V, la jeraquía de clases para la implementación de la interface gráfica de usuario.
- La necesidad de apegarse a interfaces previamente definidas. Aunque la cátedra estuvo siempre abierta a propuestas de cambios, en cuyo caso se informó inmediatamente a todos los grupos.

7 Conclusiones y trabajos futuros

A juicio de los autores el proyecto cumplió sus objetivos: los alumnos participaron activamente en su componente de desarrollo. Permitió realizar una experiencia de desarrollo de software complejo, utilizar y comprender la utilidad de herramientas existentes y comprender la importancia práctica de destacados resultados teóricos estudiados y la necesidad de su conocimiento para poder implementar soluciones de ciertos problemas concretos.

Se alcanzó a utilizar algunas de las herramientas realizadas, para la resolución de trabajos prácticos, como en el caso del analizador de precedencia simple. Esto permitió abordar la resolución de problemas más interesantes que los que se usaban anteriormente.

Se piensa dar continuidad al proyecto utilizando las herramientas ya implementadas y proponiéndose el dasarrollo de las que aún no han sido obtenidas. Entre ellas se encuentran la transformación de expresiones regulares en autómatas y la implementación de analizadores sintácticos LL(1) y de Precedencia de Operadores. Asimismo, en una segunda etapa sería interesante abordar la construcción de herramienas de asistencia a la búsqueda de soluciones para problemas no determinísticos como la obtención de gramáticas, de los distintos tipos estudiados, para lenguajes determinísticos.

8 Bibliografía

"Compiler construction". W. White - G. Goos. Springuer Verlag.

"The Theory and Practice of Compilers Writing". J. Tremblay - P. Sorenson. Mc. Graw Hill. 1985.

"Compiler Design in C". A. Holub. Prentice Hall. 1990.

"Writing Compilers and Interpreters". R. Mak - John Wiley & Sons. 1991.

"Compilers. Principles, Techniques and Tools". Aho - Sethi - Ullman. Adison Wesley. 1988.

"The Design and Construction of Compilers". R. Hunter. John Wiley & Sons. 1981.

"The Implementation of Functional Programming Languages". Peynton Jones. Prentice Hall. 1987.

"Lex & Yacc" J. Levine, T. Mason & D Brown. O'Reilly & Associates 1992

"The design of the UNIX operating systems". M Bach. Prentice Hall 1986.

"The UNIX operating systems" K. Christian. John Wiley & Sons 1983

" The theory of Parsing Translation and Compiling". Aho, Ullman, Prentice Hall 1973.

"The C Programming Language". Kernighan Ritchie. Prentice Hall 1988.

"The UNIX Programming Environment". Kernighan & Pike. Prentice Hall 1984.

"Introduction to Authomata Theory, Languages and Computation" Hopcroft Ullman . Addisson Wesley 1979.