Minerva: Una Herramienta para un Curso de Lenguajes Formales y Autómatas

Fernanda Cuadrado Estrebou **Mariela Lanza** **Virginia Mauco**[1]

Rosana Barbuzza[2] **Liliana Favre**[3]

Departamento de Computación y Sistemas
Facultad de Ciencias Exactas
Universidad Nacional del Centro
de la Provincia de Buenos Aires
Argentina

Resumen

Se describe en este trabajo una herramienta didáctica, *Minerva*, que fue diseñada para asistir al alumno en un curso introductorio de Ciencias de la Computación implementado en el primer año de la carrera de Ingeniería de Sistemas de la Universidad Nacional del Centro de la Provincia de Buenos Aires (U.N.C.P.B.A.) en Argentina. El curso brinda una introducción al estudio de lenguajes formales, autómatas, y conceptos básicos de complejidad y computabilidad.
Minerva es una herramienta interactiva y visual, implementada en Java, que asiste al alumno en el diseño, depuración y ejecución de autómatas finitos, de pila y máquinas de Turing. También permite experimentar con gramáticas y con el Lema *Pumping*. El uso de la computadora en un curso de lenguajes formales le crea al alumno un ambiente de trabajo similar al de otros cursos de carreras de Ciencias de la Computación. Además, la integración de la teoría con experimentación práctica sobre la computadora ayuda a una mejor comprensión de los conceptos teóricos.

Palabras claves: Lenguajes formales, Autómatas, Software educativo.

Abstract

In this paper we describe the software tool *Minerva* to provide interaction and feedback in an introductory Computer Science course implemented in the Undergraduate Degree Program in Systems Engineering at Universidad Nacional del Centro de la Provincia de Buenos Aires (U.N.C.P.B.A.), Argentina. This course provides an introduction to the study of formal languages, automata, and basic concepts about complexity and computability.
Minerva is an interactive and visual tool, implemented in Java, that allows the students to design, debug, and run finite and pushdown automata and Turing machines. It also supports experimentation with grammars and the Pumping Lemma. Using a software tool for hands-on experience in a formal languages course creates an atmosphere that resembles other Computer Science courses. Moreover, combining theory with experimentation on a computer brings more meaning to the theory.

Keywords: Formal Languages, Automata, Educational Software.

[1] INTIA vmauco@exa.unicen.edu.ar
[2] PLADEMA – ISISTAN rbarbu@exa.unicen.edu.ar
[3] CIC (Comisión de Investigaciones Científicas de la Pcia. de Buenos Aires) – INTIA lfavre@exa.unicen.edu.ar

1. Introducción

El surgimiento de nuevos paradigmas de computación impone modificaciones en los planes de estudio de carreras de Computación que contemplen la interacción de la teoría de la computación con otras disciplinas [1, 11, 14, 17]. Parece conveniente que los estudiantes analicen la naturaleza y los límites de los procesos computacionales desde etapas tempranas de su formación a fin de brindarles bases sólidas para abordar problemas relacionados a nuevas tecnologías.

Los cursos tradicionales de Teoría de la Computación abarcan la teoría de lenguajes formales y autómatas y nociones de complejidad y computabilidad. En general, estos cursos analizan una jerarquía de autómatas y su potencia computacional como reconocedores de lenguajes. El alumno accede a estos contenidos cuando ya ha cursado más del 50% de los cursos de la carrera. Esto dificulta que los contenidos puedan ser presentados como pilares en una currícula de Ciencias de la Computación impidiendo a los estudiantes relacionar los aspectos teóricos con los prácticos abordados en cursos previos. Si bien, el alumno ya ha cubierto gran parte de su formación básica, aún no percibe las limitaciones de los procesos computacionales. Además, dado el enfoque eminentemente teórico de estos cursos el amplio rango de aplicaciones de la Ciencia de la Computación teórica queda oculto entre definiciones, teoremas y demostraciones.

Teniendo en cuenta lo anterior, se incluyó en el plan de estudios de la carrera de Ingeniería de Sistemas de la U.N.C.P.B.A. un curso introductorio denominado “Ciencias de la Computación I”. El objetivo del curso es brindar una introducción al estudio de los procesos computacionales y explorar su alcance en el contexto de una jerarquía de autómatas, con un enfoque accesible a alumnos de primer año de la carrera. Este curso no pretende sustituir a otros de enfoque más teórico sino acercar a los estudiantes conceptos recurrentes, fundamentales en Ciencias de la Computación que serán aplicados a lo largo de toda la carrera. La intención es que los alumnos no estudien estos tópicos en un único curso aislado donde la formalización sea un fin en sí misma, sino que sean capaces de traducir problemas en abstracciones usando modelos formales y razonar sobre sus propiedades de una manera rigurosa. La diferencia de este curso con los tradicionales no está en sus contenidos sino en el nivel de los alumnos para el que fue pensado, y la metodología de enseñanza aplicada.

En general, los cursos tradicionales de lenguajes formales no utilizan software para asistir a la enseñanza. La ejercitación práctica vinculada al diseño de autómatas y gramáticas se realiza con papel y lápiz, siendo difícil, y en muchos casos tedioso, realizar un análisis de propiedades y detectar errores en estos diseños. En consecuencia, los alumnos consideran a estos cursos menos atractivos, y en algunos casos más difíciles, que otros en los que interactúan con la computadora. Luego, consideramos beneficioso implementar una herramienta didáctica que se ajustara a los contenidos y la metodología seguida en el curso.

En [6] se describió en detalle el curso, la metodología de enseñanza y la evaluación de la experiencia de introducir estos contenidos en un primer año de la carrera. En este artículo el énfasis está puesto en la descripción de *Minerva*, una herramienta interactiva y visual, implementada en Java, para diseñar, depurar y ejecutar autómatas finitos, de pila y máquinas de Turing y gramáticas. Además, permite experimentar con propiedades de lenguajes como el Lema *Pumping* y realizar comprobaciones de resultados al aplicar algoritmos de conversión entre diferentes tipos de autómatas finitos, gramáticas y expresiones regulares. La interacción con *Minerva* permite al alumno no sólo detectar errores mientras lleva a cabo la comprobación de resultados sino también durante la edición de autómatas o gramáticas. La herramienta fue utilizada en el dictado del curso en el año 2001 siendo el resultado de la experiencia satisfactorio.

Minerva fue desarrollada a partir de una tesis de grado de la carrera de Ingeniería de Sistemas en la U.N.C.P.B.A. [5].

En la sección 2 se describen algunos trabajos relacionados. La sección 3 resume las características del curso y la sección 4 describe a *Minerva*. Finalmente, se presentan conclusiones en la sección 5.

2 Trabajos Relacionados

Actualmente, existen algunas herramientas educativas que podrían utilizarse en cursos introductorios de lenguajes formales. Entre las más recientes, la herramienta JFLAP [8, 10] provee a los alumnos una representación visual de autómatas y gramáticas, permitiendo diseñar y simular el comportamiento de cualquiera de estas representaciones. Si bien es una herramienta muy completa, el enfoque que utiliza difiere al seguido en el curso en algunos aspectos. Por ejemplo, JFLAP permite el uso de reglas contractivas para generar lenguajes libres del contexto y provee poca ayuda para la autocorrección. Así por ejemplo, durante la construcción de autómatas no se diferencia entre autómatas determinísticos y no determinísticos, con lo cual si el alumno desconoce el concepto de determinismo, la herramienta no permite distinguir la diferencia entre los dos modelos. Además, sólo es posible experimentar con máquinas de Turing de hasta dos cintas. Por otra parte, existen otras herramientas como PumpLemma [8] y Turing’s World [4] que cubren sólo algunos temas del contenido del curso. La primera de ellas, es una herramienta que permite al usuario experimentar con el uso del Lema *Pumping* para un conjunto restringido de lenguajes no regulares. Esta herramienta ha sido desarrollada exclusivamente para probar que un lenguaje no es regular usando el lema, pero no provee ningún medio para razonar sobre lenguajes regulares. La segunda herramienta, Turing’s World, permite diseñar, depurar y ejecutar máquinas de Turing, en un ambiente gráfico sobre Macintosh.

Minerva brinda las mismas facilidades que las herramientas citadas. Sin embargo, el énfasis fue puesto en aspectos

didácticos pensando en el nivel de alumnos que la utilizarán y en el enfoque del curso. Fue diseñada específicamente para ser usada en los primeros cursos de carreras de Ciencias de la Computación. Además fue implementada en Java a fin de que pueda ser empleada sobre diferentes plataformas.

3 El Curso "Ciencias de la Computación I"

A continuación se describe el contenido del curso, su ubicación en el plan de estudios de la carrera y la metodología de enseñanza aplicada. Una descripción detallada puede consultarse en [6].

3.1 Contenido

El enfoque del curso es principalmente práctico y comprende el estudio de una jerarquía de máquinas abstractas (autómatas finitos, autómatas de pila y máquinas de Turing). Para los distintos modelos computacionales, se considera la capacidad o poder computacional en el contexto del reconocimiento de lenguajes. También se introducen las gramáticas como generadores de lenguajes.

La primera unidad presenta los conceptos básicos, operaciones elementales y la notación usada para alfabetos, cadenas y lenguajes.

En la segunda unidad se estudian los lenguajes regulares. Se definen los formalismos asociados a los lenguajes regulares: autómatas finitos, gramáticas regulares y expresiones regulares. Se introduce la noción de no determinismo junto con los algoritmos para convertir lenguajes de un formalismo a otro. Estas conversiones son autómata finito no determinístico (AFND) en autómata finito determinístico (AFD), AFD a AFD con cantidad mínima de estados, AFND a gramática regular y gramática regular a AFND, expresión regular a AFND y AFD a expresión regular. Se definen, además, los autómatas finitos traductores como modelos de procesos que producen una salida en función de alguna entrada. Se presentan algunos problemas fuera de aplicaciones de procesamiento de lenguajes de programación (como por ejemplo, la máquina expendedora de boletos, un semáforo, etc.), para que el alumno amplíe el campo de aplicación de este modelo de autómata y comience a comprender la utilidad de los modelos formales como una herramienta para el diseño de soluciones a problemas prácticos. Se estudian también algunas propiedades de los lenguajes regulares.

La tercera unidad abarca el estudio de los lenguajes libres del contexto y su relación con los autómatas de pila reconocedores y traductores, y las gramáticas libres del contexto. Se definen los autómatas de pila, se propone el diseño de autómatas de pila determinísticos y no determinísticos y se muestra que ambos modelos no son equivalentes. Se introducen las gramáticas libres del contexto como un mecanismo para generar los mismos lenguajes que pueden reconocer los autómatas de pila. Se presenta también la notación BNF (Backus-Naur form) para que el alumno se familiarice con la misma, ya que será utilizada posiblemente en otras materias para presentar la sintaxis de distintos lenguajes de programación, como por ejemplo Pascal en el primer año de la carrera. Se demuestran propiedades de clausura de los lenguajes libres del contexto.

En la cuarta unidad el tema central son las máquinas de Turing y los lenguajes que éstas reconocen. Esta unidad contiene definiciones básicas y las distintas versiones de máquinas de Turing determinísticas y no determinísticas para reconocer y traducir lenguajes, o para calcular funciones (máquina de una cinta, máquina multicinta, autómata linealmente acotado). Se presentan los lenguajes sensibles al contexto como los reconocidos por los autómatas linealmente acotados y generados por las gramáticas sensibles al contexto.

Con el objetivo de integrar las clases de lenguajes estudiadas, la unidad cinco presenta la jerarquía de lenguajes y su correspondencia con la jerarquía de autómatas analizada. El objetivo es que el alumno pueda distinguir, dado un lenguaje arbitrario, cuál es el modelo de autómata más restrictivo que se puede diseñar para reconocerlo, y cuál es la gramática correspondiente.

La última unidad trata el tema de la computabilidad en forma introductoria, con el objetivo de mostrar que el límite de las capacidades computacionales de las máquinas estudiadas corresponde a los procesos computacionales mismos y no a la tecnología utilizada. Se introducen conceptos básicos de complejidad espacial y temporal, y se muestran algunos ejemplos.

3.2 Contexto

La carrera de Ingeniería de Sistemas consta de 5 años; los tres primeros corresponden al núcleo de formación básica y los restantes al ciclo de especialización. Este último está organizado en áreas temáticas garantizando una formación relativamente completa de alguna rama específica de conocimiento (Ingeniería de Software, Procesamiento de Señales, Arquitectura de Computadoras, etc).

El curso "Ciencias de la Computación I" se dicta en el segundo cuatrimestre del primer año de la carrera. Los conocimientos matemáticos necesarios se proveen en el curso de ingreso obligatorio establecido en el plan de estudios.

Un curso con las características de "Ciencias de la Computación I" debe integrarse obviamente con cursos avanzados que motiven un análisis profundo en un determinado dominio. Los contenidos de "Ciencias de la Computación I" están actualemente integrados con otros cursos, como por ejemplo:

- Durante el segundo año en las materias "Análisis y Diseño de Algoritmos I" y "Análisis y Diseño de Algoritmos II" se implementan algoritmos eficientes vinculados al procesamiento de lenguajes. Por ejemplo,

algoritmos que permitan obtener autómatas no determinísticos desde expresiones regulares, que transformen autómatas no determinísticos en determinísticos, analizadores léxicos y sintácticos simples, etc.

- En la materia "Ciencias de la Computación II" se abordan nuevamente aspectos de computabilidad en el contexto de la lógica.
- La materia "Análisis y Diseño de Algoritmos II" introduce las bases de la teoría de complejidad computacional a través del análisis de algunos problemas relacionados a las clases P y NP.
- En distintos cursos del núcleo de formación básica se utilizan los modelos formales analizados en Ciencias de la Computación I .
- La carrera incluye un curso obligatorio y otro optativo de construcción de compiladores
- El alumno puede tomar cursos específicos de computabilidad y complejidad a través de materias optativas del área de informática teórica.
- El ciclo de especialización incluye, por ejemplo, cursos de Inteligencia Artificial, Redes de Comunicación, Recuperación y Visualización de Información, etc.

3.3 Metodología

La duración del curso es de un cuatrimestre (70 horas aproximadamente), dictándose una clase teórica de 2 horas y una clase práctica de 3 horas por semana.

En general, la metodología seguida para enseñar los contenidos detallados en la Sección 3.1 consiste en introducir las ideas fundamentales informalmente por medio de ejemplos, para motivar su uso en casos prácticos. Luego, se presentan las definiciones formales correspondientes.

En cada clase práctica se realiza una breve introducción teórico-práctica para la resolución del trabajo práctico correspondiente, mostrando ejemplos de algunos ejercicios tipo. Los alumnos disponen también de apuntes de cada uno de los temas incluidos en los trabajos prácticos y ejercicios resueltos con una explicación paso a paso de los mismos, junto con información detallada de la bibliografía a ser utilizada para cada tema [16]. Fueron elaborados especialmente para presentar los temas de una manera accesible para estudiantes de primer año, ya que, en general, los textos tradicionales están dirigidos a estudiantes de cursos superiores y por lo tanto no responden al enfoque propuesto [2, 3, 9, 12, 13, 15].

Cada estudiante debe aprobar dos exámenes, un parcial y un final. El examen parcial incluye ejercicios prácticos similares a los propuestos en los trabajos prácticos. En el examen final se evalúan los contenidos completos del curso.

Desde el año 2001 los alumnos pueden utilizar *Minerva* para diseñar, depurar y ejecutar autómatas finitos, de pila y máquinas de Turing determinísticos y no determinísticos, diseñar gramáticas, aplicar los algoritmos de conversión para lenguajes regulares y utilizar el Lema *Pumping*. *Minerva* se ha utilizado también en las clases teóricas y prácticas para introducir algunos temas y para desarrollar y mostrar ejemplos.

4 Minerva

Minerva es una herramienta interactiva y visual que soporta la creación de distintos tipos de autómatas (finitos, de pila y máquinas de Turing), gramáticas y la comprobación de resultados. En particular permite:

- decidir la pertenencia de una cadena a un lenguaje reconocido por un autómata
- decidir si una cadena puede ser generada por una gramática
- generar cadenas que pertenecen al lenguaje representado por una gramática
- realizar comprobaciones de resultados al aplicar los siguientes algoritmos de conversión: AFND con transiciones vacías a AFND, AFND a AFD, AFD a AFD con cantidad mínima de estados, AFD a expresión regular, expresión regular a AFND, AFND a gramática regular, gramática regular a AFND.
- aplicar el Lema *Pumping* a lenguajes regulares
- comprobar que un lenguaje no es regular a partir del Lema *Pumping*

Minerva fue implementada en Java. Su diseño consistió en modelar las vistas y el modelo por separado, utilizando el patrón de diseño *Observer* [7] para vincular la funcionalidad del sistema con la interfaz a usuario.

En las siguientes secciones se describen brevemente las etapas de análisis y diseño de *Minerva* y se ejemplifican algunas de sus funcionalidades.

4.1 Análisis y Diseño

En la etapa de análisis se estudiaron los requerimientos que debían ser cubiertos por la aplicación, surgiendo así el diseño de la jerarquía de clases que especifica la funcionalidad del sistema. La interfaz fue diseñada teniendo en cuenta los requerimientos didácticos a los cuales está enfocada la aplicación y el tipo de usuario al cual se orientó la herramienta.

La Figura 1 muestra las clases principales que modelan los diferentes autómatas. El comportamiento de un autómata se representó mediante la clase *Automaton*, que fue extendida para modelar los autómatas finitos, autómatas de pila y máquinas de Turing. Los principales atributos que contiene esta clase son *states*, *transitions* y *alphabet*.

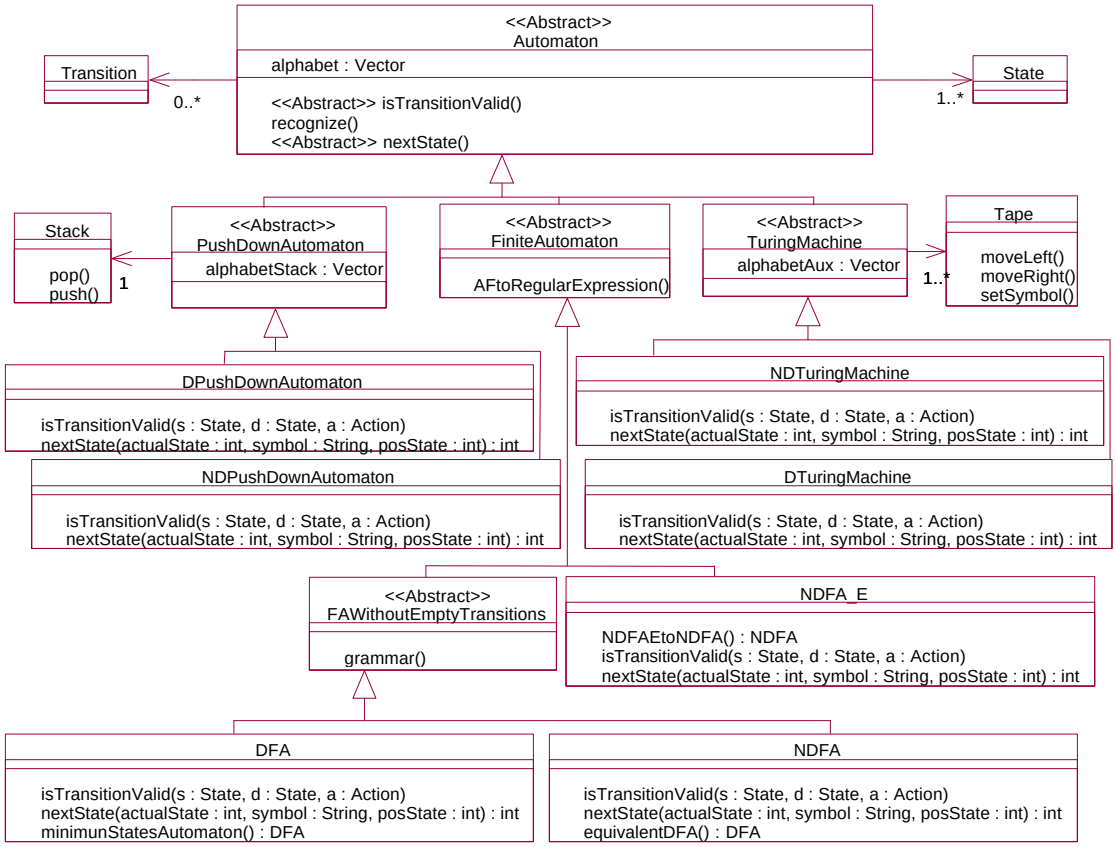

Figura 1 – La jerarquía *Automaton*

Dentro de los métodos implementados para manejar los atributos mencionados, se deben destacar aquellos que permiten el ingreso de una transición debido a que la comprobación de su validez depende del tipo de autómata. Por este motivo, los métodos mencionados son métodos *template* debido a que invocan al método abstracto *isTransitionValid(...)*.

El comportamiento general de los autómatas consiste en reconocer una cadena de entrada, funcionalidad que es llevada a cabo por la clase *Automaton* mediante el método *template recognize(...)* que invoca al método *nextState(...)*. Este último es abstracto ya que la diferencia en el reconocimiento de una cadena entre los diferentes tipos de autómatas está dada por la forma en que se obtiene el estado que puede ser alcanzado a partir de otro.

La vista de la aplicación es la responsable de mostrar al usuario la información representada en el modelo. Para asegurar la consistencia entre los datos a ser visualizados y los almacenados en el modelo, cada vista "observa" a un objeto del modelo según corresponda. Este comportamiento fue llevado a cabo mediante la utilización del patrón de diseño O*bserver*, en donde las vistas son los observadores y los objetos del modelo los observados.

4.2 Funcionalidad

Minerva permite el diseño de los diferentes tipos de autómatas y gramáticas y la comprobación de resultados. En todos los casos, el usuario no sólo puede acceder a ayudas específicas que indican cómo llevar a cabo una acción, sino que se provee un vínculo a una página que contiene los apuntes del curso. En las siguientes secciones se ejemplifica la edición de un autómata de pila, una máquina de Turing y una gramática regular, así como la aplicación del Lema *Pumping*.

4.2.1 Autómatas de Pila

Para la creación de este tipo de autómatas se provee una ventana en la que se va mostrando la definición formal a medida que se efectúan cambios sobre el autómata (Figura 2). Ésta contiene una ventana de edición que muestra la función de transición mediante un diagrama de transición de estados, y una descripción del alfabeto, el alfabeto de la pila, el símbolo distinguido de la pila, el conjunto de estados, el estado inicial y el conjunto de estados finales del autómata.

Durante la edición se verifica que cada acción llevada a cabo respete la definición del tipo de autómata que se está editando. Por ejemplo, si el autómata que se está editando es determinístico *Minerva* verifica que cada transición lleve a un único estado; en caso contrario, se informa al usuario.

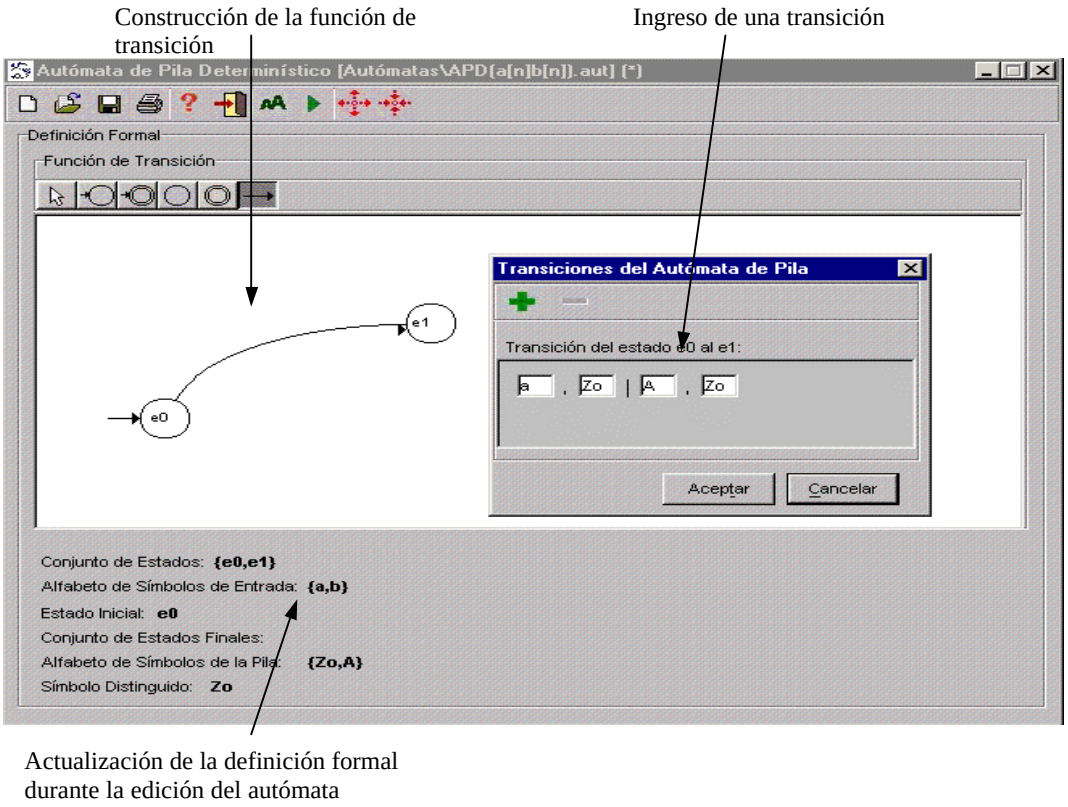

Figura 2: Edición de un autómata de pila

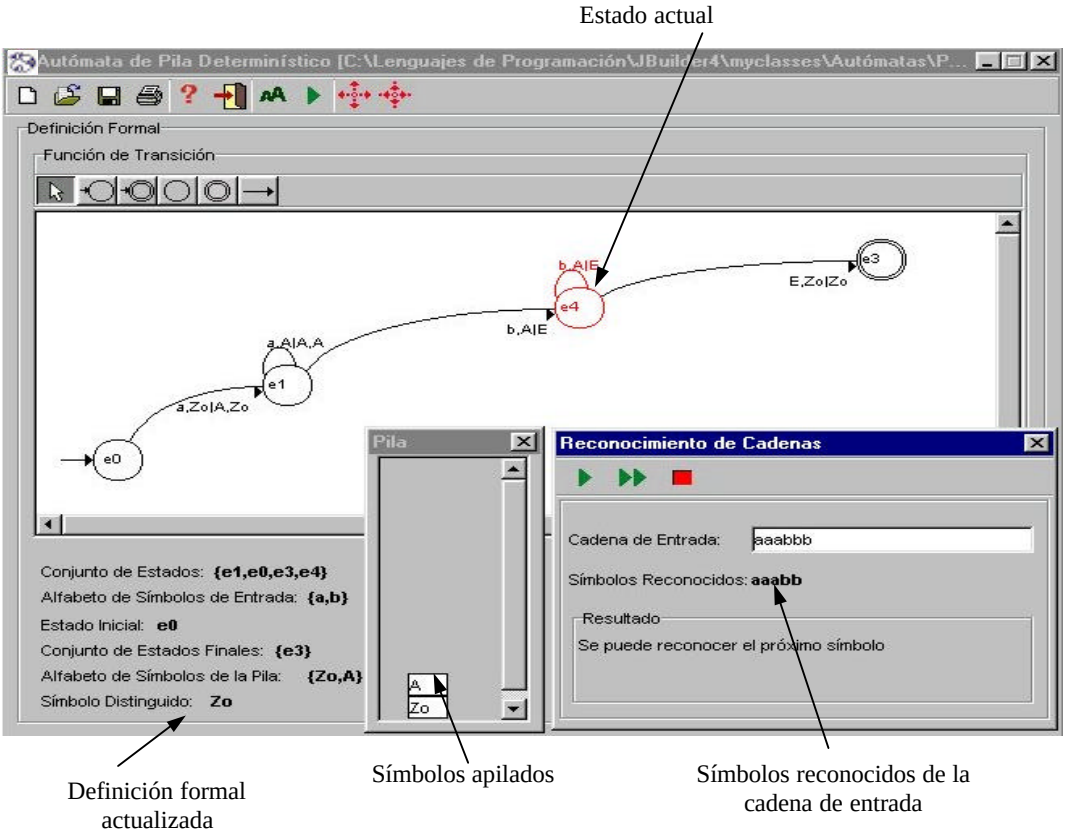

Figura 3 – Ejecución de un autómata de pila determinístico

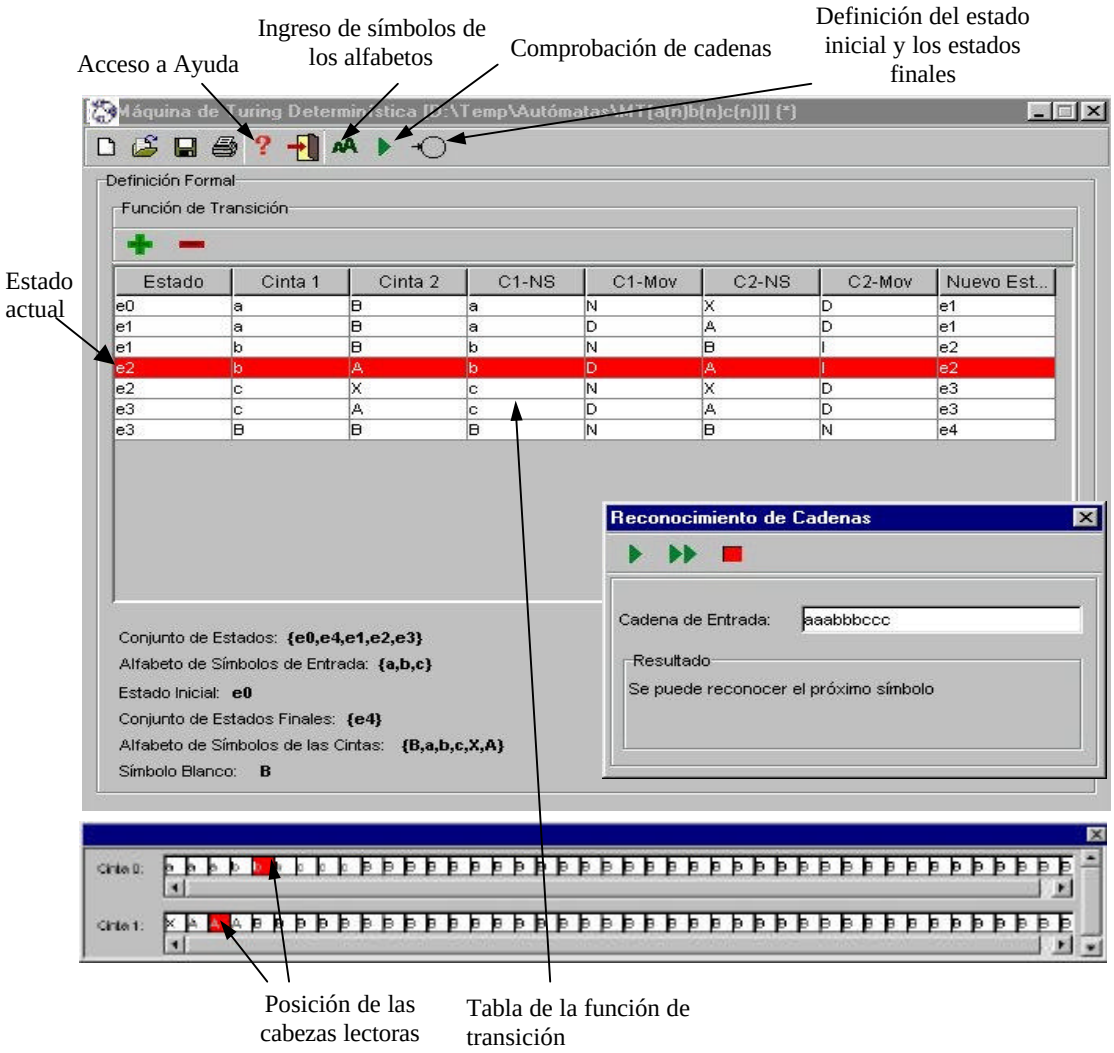

Figura 4 – Ejecución de una máquina de Turing determinística

En los autómatas de pila el usuario debe agregar los símbolos del alfabeto de entrada y los del alfabeto de la pila. Si al agregar una transición, la misma contiene símbolos que no están en los alfabetos, se le informa el error al usuario.
Durante la comprobación de si una cadena es reconocida por el autómata diseñado se muestra en qué estado se encuentra el autómata, a través de qué transición arribó al mismo, cuáles son los símbolos de la cadena de entrada que se han reconocido y el estado de la pila. En la Figura 3 se ejemplifica el reconocimiento de la cadena *aaabbb* por un autómata de pila que reconoce al lenguaje $L_1 = \{a^n b^n / n > 0\}$.
Mostrar cuál es el comportamiento de la pila ayuda al usuario a detectar si el autómata de pila se comporta como él desea ya que permite observar qué símbolo es apilado (o desapilado) cuando se lee cada uno de los símbolos de la cadena de entrada. Por otra parte, se puede detectar cuál es el efecto de una transición dada, debido a que se va mostrando el estado actual, cuál fue el símbolo leído, y además, se muestra la configuración de la pila en el nuevo estado.

4.2.2 Máquinas de Turing
A diferencia de los autómatas de pila, la edición de la función de transición de las máquinas de Turing se realiza en una tabla (Figura 4). Durante la edición de la máquina de Turing se deben ingresar los símbolos del alfabeto de entrada y los símbolos auxiliares. También se deben definir el estado inicial y los estados finales, accediendo a una ventana de diálogo. En esta ventana, sólo se permite ingresar un único estado inicial, con lo cual se garantiza que la máquina esté bien definida. Por último, se debe completar la tabla que representa la función de transición; cada fila de la misma indica una transición en la máquina de Turing.

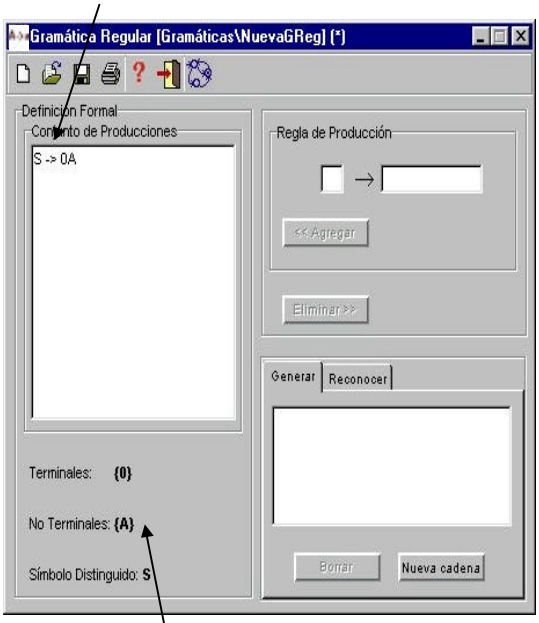

Figura 5 – Edición de una gramática regular

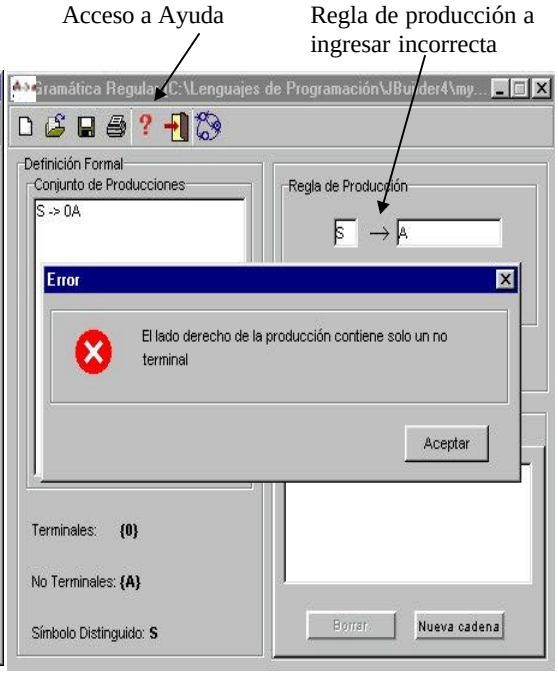

Figura 6 – Ingreso de una regla de producción incorrecta

Una vez editada la máquina de Turing, el usuario puede comprobar si la misma reconoce cadenas del lenguaje para el cual fue diseñada. Para esto se debe acceder a una ventana de diálogo (Figura 4) en la que se ingresa una cadena de entrada. Cuando se abre esta ventana también se muestra una vista de las cintas de la máquina.
En la Figura 4 se muestra la configuración de una máquina de Turing en un determinado estado durante el reconocimiento de una cadena. Esta máquina reconoce el lenguaje $L_2 = \{a^n b^n c^n / n > 0\}$. Las vistas de las cintas van mostrando cuáles son los símbolos que contienen y la posición en la que se está leyendo. Esto ayuda al usuario a observar cómo se comporta la máquina de Turing, y a detectar posibles errores en el diseño de la misma.

4.2.3 Gramáticas

La edición de gramáticas consiste en el ingreso de las reglas de producción. El resto de los elementos de la definición formal de una gramática (símbolos terminales y no terminales, símbolo distinguido) son actualizados automáticamente por la herramienta.
En la Figura 5 se ejemplifica la edición de una gramática regular que genera al lenguaje L_3 = { x/x∈ {0, 1}*y x contiene la subcadena 00 ó x contiene la subcadena 11}. El usuario ingresa cada regla de producción en los campos de texto *"Regla de Producción"*. Supongamos que una de las ingresadas es la regla S→0A. Los conjuntos de símbolos terminales y no terminales se actualizan en forma automática mostrando los símbolos *0* y *A*, respectivamente. En el caso que la regla de producción ingresada no respete el formato permitido para las reglas de producción de una gramática regular, se avisa del error al usuario, indicándole por qué la regla a ingresar es incorrecta. Si la regla ingresada es, por ejemplo, S→A y no responde al formato de una regla de producción del tipo de la gramática, se informa al usuario que es incorrecta y el motivo del error (Figura 6).
Minerva, además de asistir al usuario durante la edición de una gramática, permite analizar si la misma genera cadenas que pertenecen a un lenguaje. Existen dos formas de llevar a cabo esta comprobación; una de ellas es ingresando una cadena para que la herramienta verifique si puede ser derivada a partir de las reglas de producción de la gramática creada, y la otra es a través de la generación de cadenas por parte de la aplicación. En el primer caso, si la cadena puede ser derivada a partir de las reglas de producción dadas se muestra al usuario cuáles fueron las reglas de producción utilizadas; en caso contrario, se informa al usuario a través de un mensaje. En el segundo caso, el usuario puede acceder a un diálogo en donde se muestran cuáles son las reglas de producción utilizadas para derivar dicha cadena. Las Figuras 7 y 8 ejemplifican el comportamiento de *Minerva* para una gramática regular que genera al lenguaje L_3.
En el caso particular de las gramáticas regulares, se debe mencionar que el usuario puede acceder a una ventana que muestra un autómata finito que reconoce el lenguaje generado por la gramática (Figura 9).

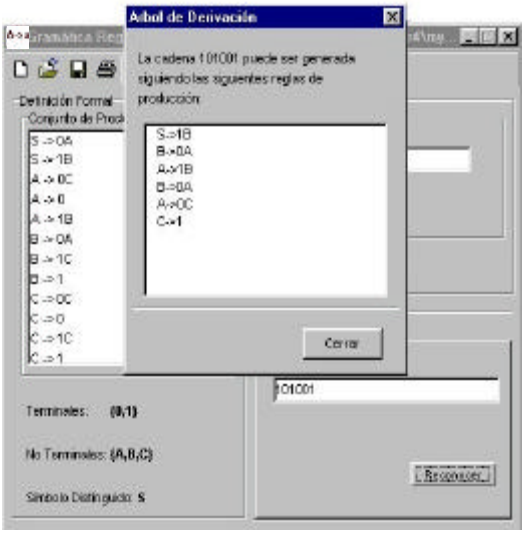

Figura 7 - Pertenencia de una cadena al lenguaje generado por una gramática regular

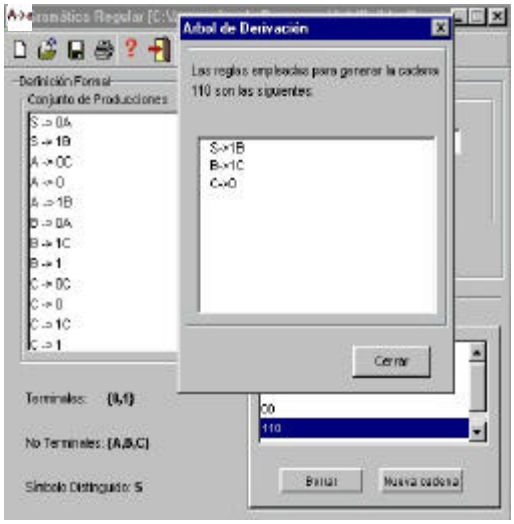

Figura 8 - Generación de cadenas a partir de una gramática regular

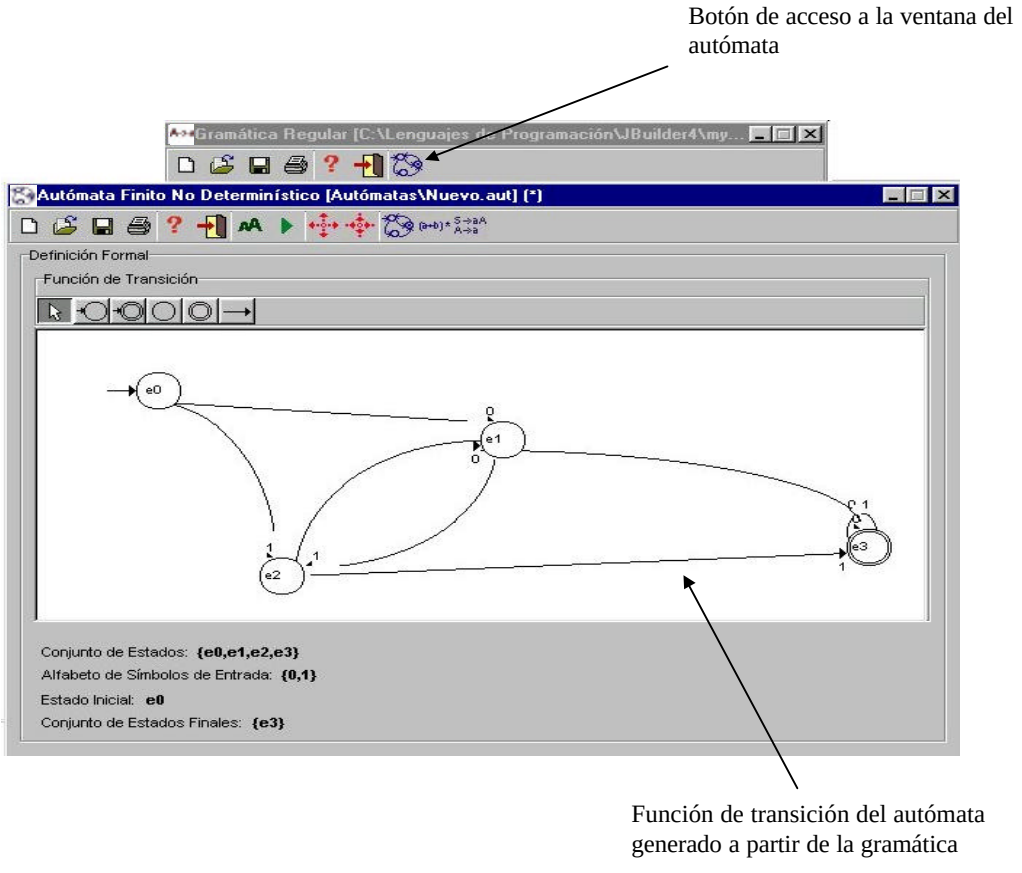

Figura 9 – Conversión de una gramática regular a un AFND

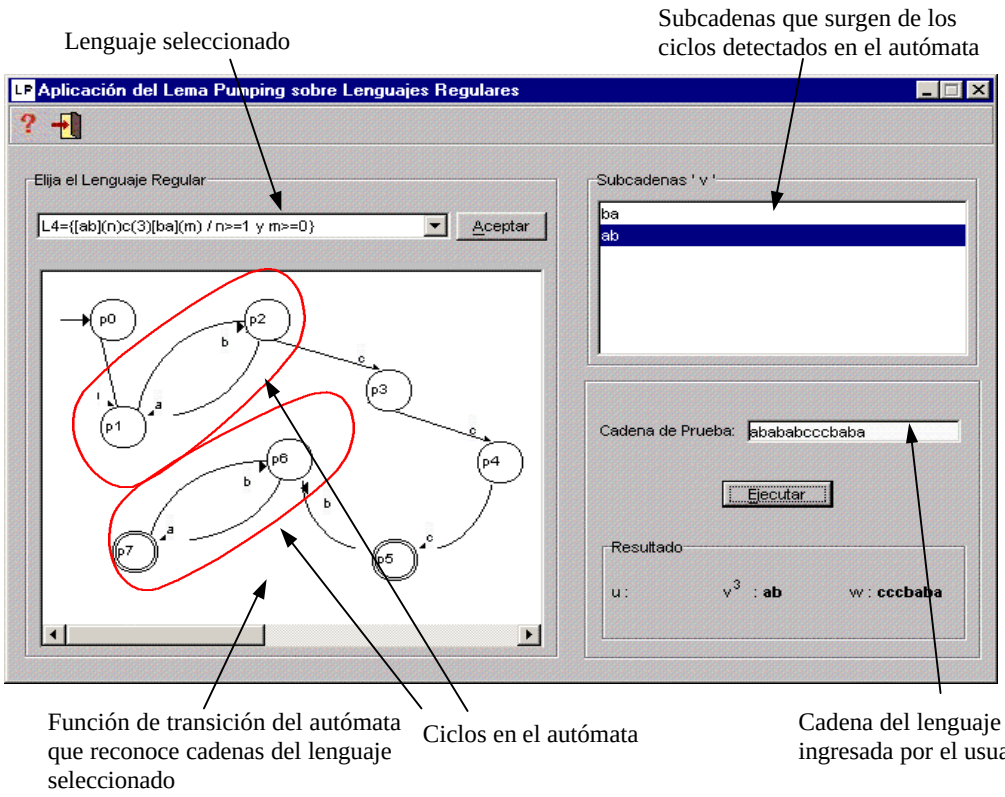

Figura 10 - Aplicación del lema Pumping sobre un lenguaje regular

4.2.4 Lema Pumping

En esta sección se ejemplifica la aplicación del lema a un lenguaje regular y del contrarrecíproco del lema a un lenguaje no regular.

Aplicación del Lema Pumping

Dado un lenguaje regular, *Minerva* permite al usuario experimentar con algunas cadenas de prueba z pertenecientes al lenguaje y tales que $|z| \geq n$ siendo n un valor mayor o igual que la cantidad de estados de un autómata finito que reconoce al lenguaje. Cada cadena es de la forma $z = uv^i w$, tal que $|uv| \leq n$, $|v| \geq 1$ y $i \geq 0$. Por ejemplo, para el lenguaje regular $L_4 = \{(ab)^n c^3 (ba)^m / n \geq 1 \text{ y } m \geq 0\}$ se muestra al usuario la función de transición de un autómata finito que es capaz de reconocer cadenas del lenguaje L_4 (Figura 10), y una lista con subcadenas que surgen de la concatenación de los símbolos asociados a las transiciones que forman ciclos dentro del diagrama de transición de estados. El usuario puede entonces seleccionar una de las subcadenas e ingresar una cadena que pertenezca al lenguaje; cuando se presiona el botón *Ejecutar*, se muestra cuáles son los valores que toman las subcadenas *u*, *v* y *w*, y el valor del índice *i*. De esta forma, el usuario puede observar cuál es el significado de cada uno de los elementos que referencia el lema.

Aplicación del Contrarrecíproco del Lema Pumping

En este caso, el usuario ingresa un lenguaje no regular y trata de probar que no es regular. La experimentación puede realizarse solamente sobre lenguajes ordenados. Por ejemplo, podría experimentarse con el lenguaje $L_5 = \{ a^p b^p / p > 0\}$, pero no con el lenguaje $L_6 = \{w / w \in \{a, b\}^* \text{ y } |w|_a = |w|_b\}$. Cuando el usuario ingresa un lenguaje éste es analizado para verificar que sea ordenado.

Supongamos que el usuario ingresa el lenguaje L_5 y el valor de la variable *n*, que debe ser un factor del exponente utilizado en L_5; en este caso posibles valores válidos de *n* serían *p*, *2p*, y *3/2p*. Si el usuario ingresa un valor no válido, la herramienta se lo comunica mencionándole la causa del error.

Luego de haber ingresado los datos anteriores y de presionar el botón *Continuar* (Figura 11), se muestra una lista de los posibles valores que puede tomar la subcadena *v*; al seleccionar uno de estos, por ejemplo b^k, se muestra el formato de las subcadenas *u*, *v* y *w*, es decir:

$u = a^p b^j \quad p > 0, j \geq 0$

$v = b^k$ donde $k \geq 1$, debido a que $|v| \geq 1$

$w = b^{p-j-k}$

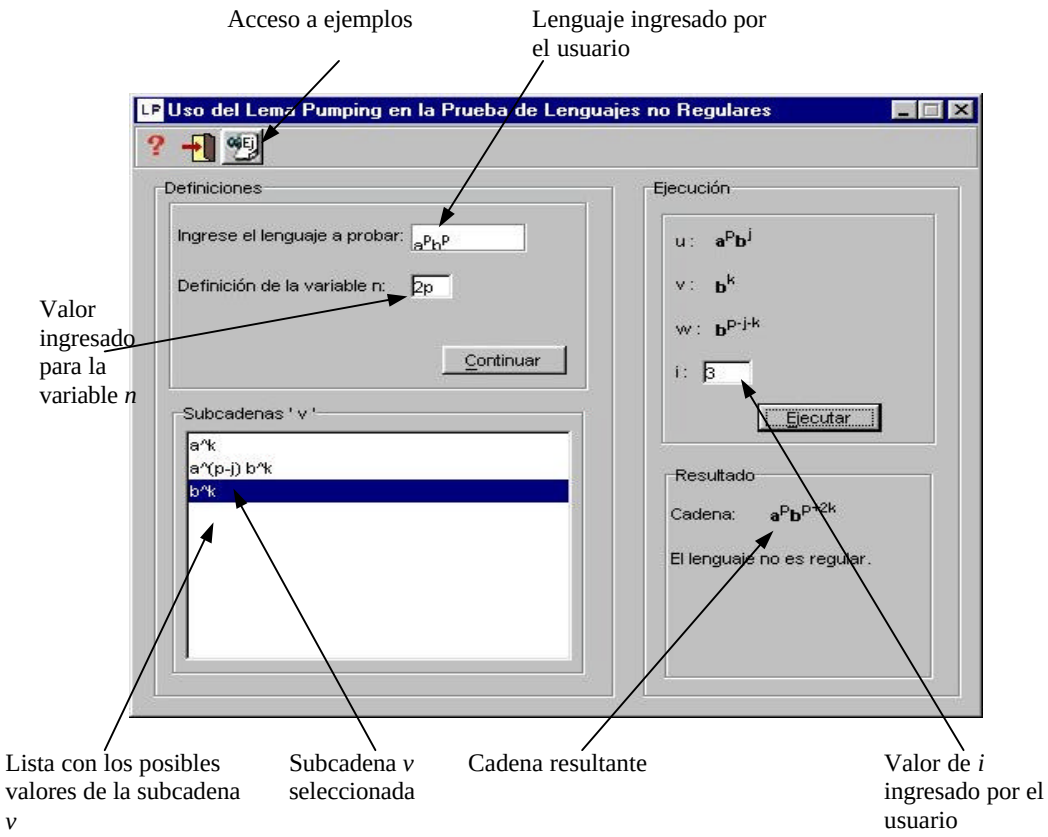

Figura 11 - Aplicación del contrarrecíproco del lema Pumping sobre un lenguaje no regular

El siguiente paso es el ingreso del valor de la variable *i*, que en el ejemplo es 3. Luego de esto, si se presiona el botón *Ejecutar* (Figura 11), se muestra la cadena uv^iw que en este caso es a^pb^{p+2k}. Se puede observar que la misma no pertenece a L_5, debido a que L_5 describe cadenas que contienen una cantidad de *a*'s seguida de la misma cantidad de *b*'s. Por lo tanto, se puede concluir que el lenguaje ingresado no es regular.
La aplicación del contrarrecíproco del Lema *Pumping* también se puede llevar a cabo sobre un conjunto de lenguajes provistos por *Minerva*. Los pasos que se deben realizar son los mismos que los descritos anteriormente.

5. Conclusiones

Se describió en este trabajo una herramienta didáctica, *Minerva*, utilizada en el curso "Ciencias de la Computación I" de la carrera de Ingeniería de Sistemas en la U.N.C.P.B.A.
Minerva soporta el diseño, la depuración y ejecución de autómatas finitos, autómatas de pila, y máquinas de Turing. Además permite experimentar con gramáticas, con equivalencias entre distintas formalizaciones para lenguajes regulares y con propiedades de lenguajes regulares y libres del contexto. Un problema que mereció especial análisis fue la incorporación en la herramienta de facilidades para experimentar con el Lema *Pumping*, dado que la experiencia recogida en distintos dictados del curso mostraba que la comprensión del mismo resultaba compleja a los alumnos.
Minerva está implementada en Java y su diseño facilita el mantenimiento ante los cambios que imponga la evolución del curso o las variantes que requiriesen otros cursos de similares características.
Minerva se utilizó parcialmente en el dictado del curso durante el año 2001 como soporte en algunas clases teóricas y prácticas. La evaluación de esa experiencia fue satisfactoria. La motivación en el curso fue mayor, siendo esto un factor importante en todo proceso de aprendizaje. Se distribuyó una encuesta a alumnos de diferentes años que ya habían realizado el curso sin la utilización de *Minerva*. Según la opinión de los estudiantes encuestados *Minerva* es simple de usar y ayuda al alumno en la autocorrección de los ejercicios propuestos en la práctica. Los alumnos también aportaron valiosos comentarios acerca de posibles facilidades que podrían ser incorporadas a la interfaz a usuario de *Minerva*, algunas ya fueron incorporadas y otras se tendrán en cuenta en futuras extensiones. También, dado que el curso se dicta en distintas sedes de la Universidad, se prevé extender la herramienta para que pueda ser utilizada en educación a distancia sobre la *web*.

Este año *Minerva* estuvo disponible para los alumnos desde el comienzo del curso; sin embargo, recién al finalizar el dictado del mismo se podrá evaluar más rigurosamente el efecto del uso de la herramienta en el rendimiento de los alumnos.
A nuestro juicio, esta experiencia podría reproducirse en otros planes de estudio en carreras de Ciencias de la Computación y Sistemas de Información.

Referencias

[1] ACM Curricula Recommendation. *Computing Curricula 1991*. ACM/IEEE-CS. 1991. Disponible en: www.acm.org/education/curricula.html

[2] Aho, A., Ullman, J. *Foundations of Computer Science*. Computer Science Press. 1995

[3] Atallah, M. *Algorithms and Theory of Computation HandBook*, CRC Pr. 1999.

[4] Barwise, J., Etchemendy, J. An Introduction to Computability Theory Logic Software from CSLI Publications. Cambridge University Press.. En: http://www-csli.stanford.edu/hp/Turing1.html

[5] Cuadrado Estrebou, M. F., Lanza, M. "*Minerva*: Una herramienta didáctica para el curso Ciencias de la Computación I". Tesis de grado de la carrera de Ingeniería de Sistemas, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina, 2001.

[6] Favre, L., Mauco, V., Barbuzza, R. Introducing First-year Students to Theoretical Computer Science". Proc. ISECON 2000, *Information Systems Education Conference*. Philadelphia, USA, noviembre de 2000.

[7] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[8] Gramond, E. and Rodger, S. Using JFLAP to Interact With Theorems in Automata Theory. *Thirtieth SIGCSE Technical Symposium on Computer Science Education.* 1999, pp. 336-340.

[9] Hopcroft, J., Ullman, J. *Introduction to Automata Theory, Language, and Computation*. Addison Wesley, 1979.

[10] Hung, T. and Rodger, S. Increasing Visualization and Interaction in the Automata Theory Course. 2000. Disponible en: www.cs.duke.edu/~rodger/papers

[11] IS' 97. *Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*. Association for Computing Machinery (ACM), Association for Information Systems (AIS) and Association of Information Technology Professionals (AITP). 1997. Disponible en: www.acm.org/education/curricula.html

[12] Kelley, D. *Teoría de Autómatas y Lenguajes Formales.* Prentice Hall. 1995.

[13] Lewis, H. and Papadimitriou, C. *Elements of the Theory of Computation*, Second Edition, Prentice Hall. 1998.

[14] Loui, M. et al. Strategic Directions in Research in Theory Computing. *ACM Computing Surveys.* Vol 28, 4, 1996, pp. 575-590.

[15] Mandrioli, D. and Ghezzi, C. *Theoretical Foundations of Computer Science.* John Wiley and Sons. 1987.

[16] Mauco, V. and Barbuzza, R. Apuntes para Ciencias de la Computación I. Facultad de Ciencias. Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina. 1998.

[17] MSIS' 2000. *Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems*. Association for Computing Machinery (ACM), Association for Information Systems (AIS). 2000. Disponible en: www.acm.org/education