Grupo de Procesadores de Lenguajes
Línea: Reformulación de la presentación de la Teoría de Lenguajes Formales y Autómatas.

Jorge Aguirre, Marcelo Arroyo, Francisco Bavera
{jaguirre,marroyo,fbavera}dc.exa.unrc.edu.ar, tel +54-358-4676*235/*(*530* fax)

Resumen

El temprano desarrollo de la teoría de Lenguajes Formales y Autómatas dentro de las Ciencias de Computación hizo que su cuerpo de conocimientos y algoritmos se conformara antes de que se contara con métodos de diseño que garanticen corrección, que permitan derivar algoritmos a partir de especificaciones o verificarlos usando técnicas especiales para ello. Como consecuencia el enfoque usual de la bibliografía consiste en presentar los algoritmos o autómatas y luego demostrar que satisfacen el problema a resolver. En esta línea de trabajo se pretende reformular la presentación de los resultados usando métodos de diseño que garanticen la corrección. Un resultado de ésta línea ya obtenido es un método para la construcción de autómatas finitos a partir del predicado que define su conjunto de aceptación. Dicho método garantiza la corrección de los autómatas construidos, que ha sido usado tanto para rehacer todas las construcciones de autómatas finitos requeridos en el cuerpo de la teoría de Lenguajes Formales, incluyendo el teorema fundamental del análisis LR como para probar su aplicación a la construcción de diversos casos de estudio con objetivos prácticos.

1. Introducción

Las Ciencias de la Computación brindan métodos formales o semi-formales para la construcción de algoritmos correctos, basándose en la metodología de Hoare, E. Dijkstra y D. Gries [5][6][7] y muchas carreras universitarias introducen a sus alumnos en la programación utilizándolos. Sin embargo en la Teoría Lenguajes Formales se presentan algoritmos cuya concepción es difícil de imaginar y luego se da la demostración de la corrección de los mismos generalmente usando inducción. Este es el estilo que es usado en la bibliografía clásica que se ocupa de demostrar la corrección de las construcciones – tal el caso de los textos de Hopcroft [1] , Aho y Ullman [2], Brookshear [9], Tremblay [10], Waite [11]-, mientras que otros textos sólo presentan los algoritmos, sin ocuparse de su corrección – como los de ººFraser [12], Morgan [13], Isasi [14] y Appel [16]-.

El enfoque de presentar algoritmos y luego dar la demostración de su corrección no aporta ideas que puedan usarse en problemas de similar estructura y tiene las siguientes desventajas didácticas:

- No se afianzan en el alumno las prácticas y metodologías adquiridas en las asignaturas de construcción de algoritmos.
- No se desarrolla su capacidad para resolver problemas y diseñar soluciones.
- Se oscurece la comprensión de los algoritmos y generalmente se induce a la memorización de las demostraciones.

Por estas razones los autores creen que se debe introducir el uso de métodos de construcción que garantizan la corrección en todos los contextos en que sea posible, particularmente en la de la Teoría de Lenguajes y Autómatas, sumamente rica en construcciones algorítmicas de distintos tipos. Consideramos que este enfoque elimina

las desventajas señaladas y permite recrear la construcción de los algoritmos, los que pierden así que su apariencia de elementos revelados.

Cuando los objetos construidos son autómatas en lugar de algoritmos también lo usual es presentar el autómata buscado primero y luego dar una demostración recursiva de su corrección. En este último caso no se pueden aplicar las metodologías antes mencionadas y si se quiere evitar la presentación apriorística de la solución debería realizarse el doble trabajo de inducir su construcción intuitivamente primero y luego demostrar su corrección formalmente. Esto puede evitarse usando un método, desarrollado previamente por los autores [15], que da una heurística para la derivación de Autómatas Finitos a partir de su especificación, dada mediante predicados de primer orden.

El método mencionado brinda una estrategia que facilita la obtención de soluciones, acercando la posibilidad de que sean recreadas por el alumno y garantizando la solución obtenida, no haciendo falta; no siendo necesaria ninguna demostración adicional.

El método se basa en la noción de invariante de estado y particularmente en la de conjunto de invariantes fuerte (cief), que se esquematiza posteriormente. Parte de la especificación del lenguaje de aceptación del autómata finito buscado mediante un predicado de primer orden. Luego aplicando transformaciones según una cierta heurística, construye conjuntamente: 1) los estados finales del autómata y sus invariantes, 2) los estados intermedios conjuntamente con sus invariantes y la función de transición, 3) el estado inicial y sus transiciones. La mencionada construcción se realiza de atrás hacia delante sobre prefijos del lenguaje. Este enfoque *backward* en el diseño de autómatas coincide con el principio expresado por Gries [8] para el diseño de programas: la programación es una actividad dirigida por metas.

La necesidad de obtener implementaciones de máxima eficiencia temporal y espacial de problemas modelables con lenguajes regulares aparece en numerosas aplicaciones prácticas. Para ello basta lograr representarlos con cualquiera de los formalismos de la teoría de lenguajes y muchas veces los Autómatas Finitos son los más faciles de obtener. Cuando se ha logrado modelar un problema mediante un Autómata Finito, los resultados de la teoría de Autómatas permiten obtener mecánicamente la solución óptima, por lo cual no es necesario tener en cuenta ninguna consideración de eficiencia en el proceso de diseño del autómata inicial. En otros casos, tales como el reconocimiento de patrones léxicos, la tarea de construcción de autómatas se realiza de la forma más clara y natural mediante la especificación previa del problema mediante expresiones regulares. Una vez que se ha obtenido esta especificación se puede obtener el Autómata deseado mecánicamente para lo cual se cuenta con diversas herramientas de software como lex, flex [3][4] y Jlex [16]. No obstante la construcción de un autómata que solucione un determinado problema no siempre resulta tarea fácil; tampoco suele serlo la demostración inductiva de su corrección. La experiencia en la docencia universitaria convalida la afirmación anterior, dado que alumnos, próximos a terminar su ciclo de grado en carreras de Ciencias de Computación e Ingeniería, no son capaces de encontrar soluciones correctas para muchos problemas prácticos.

2. Resultados obtenidos

2.1 En la construcción de Autómatas

Se ha estudiado el uso de invariantes de estado, definéndose los conjuntos de invariantes de estado fuertes (cief) como sigue

dado $M=<K,\Sigma, \delta,q_0, F>$:AF (Autómata Finito), con $K=\{q_0, \ldots ,q_n\}$,
un conjunto de predicados $\{P_i: \Sigma^* \rightarrow Bool / 0 \leq i \leq n \}$ es un conjunto de invariantes de estado fuerte para M – donde P_i es el invariante de qi -

$$\forall\, \alpha \in \Sigma^* ((q0,\alpha) \succ^* (qi,\lambda) \Leftrightarrow Pi(\alpha))$$

Los cief tienen la propiedad de que el lenguaje generado por el autómata está formado por las cadenas que satisfacen al invariante de algún estado final; o sea

$$L(M) = \{\alpha / V_{qi \in F}\ P_i(\alpha) \} .$$

El método de construcción obtenido se basa en el siguiente lema.

Lema del *cief*: Dado un autómata finito $M=<K,\Sigma, \delta,q_0, F>$, dado un conjunto de predicados $I=\{P_i\}$ sobre las cadenas de su alfabeto, tal que cada q_i tiene asociado P_i , I es un ***cief*** para M, si cumple las tres condiciones siguientes ([8]):

La cadena vacía satisface al invariante asociado al estado inicial y de éste el autómata puede moverse espontáneamente a todos aquellos estados cuyos invariantes son también satisfechos por la cadena vacía.

Si el autómata se mueve del estado i al estado j por a entonces el hecho de que el invariante P_i se satisfaga para la cadena α implica que el invariante P_j se satisface para la cadena αa.

Si un invariante P_i se satisface para una cadena αa entonces hay en I algún invariante P_h que se satisface para α y el autómata se mueve de q_h a q_i por a.

Mas formalmente

Un conjunto de predicados $\{Pi: \Sigma^* \rightarrow Bool / 0 \leq i \leq n \}$ es un ***cief*** para M si cumple:

$$\forall i,j,\forall \alpha \in \Sigma^*$$
$$(\quad (1:\ P_0(\lambda) \ \wedge\ (P_j(\lambda) \Rightarrow (q_0, \lambda) \succ^* (q_j, \lambda)) \ \wedge$$
$$(2: \forall a \in \Sigma \cup \{\lambda\} ((q_i,a) \succ^* (q_j,\lambda) \Rightarrow ((P_i(\alpha) \Rightarrow P_j(\alpha a)))) \ \wedge$$
$$(3: \forall a \in \Sigma (P_i(\alpha a) \Rightarrow \exists\, h : P_h(\alpha) \wedge (q_h, a) \succ^* (q_i, \lambda)))$$
$$)$$

El metodo guía el siguiente proceso de construcción:

Dado $C=\{\alpha \in \Sigma^* / P(\alpha)\}$: conjunto regular (donde $P: \Sigma^* \rightarrow Bool$) construir $M=<K, \Sigma, \delta, q_0, F>: AF \ / \ L(M)=C$.

El método busca construir un conjunto de predicados I y el autómata M a partir del predicado P de manera que I sea un *cief* para M.
Inicialmente se introducen predicados que deben ser satisfechos por las cadenas de C y para cada una de ellos se incorpora un estado final, a K y F. A partir de este conjunto inicial de predicados se busca extender I y correlativamente K y δ, hasta lograr que se cumplan las
condiciones 2 y 3 de un *cief*. Finalmente se define el estado inicial de forma que se cumpla la condición 1.

A continuación se describen detalladamente las tres etapas en que consiste:

Paso 1: Construcción de los invariantes de los estados finales. Tratar de expresar P como una disyunción natural de predicados.

$P = \bigvee_{i=1,k} P_i \qquad (\bigvee_{i=1,k} P_i = P_1 \vee P_2 \vee \ldots P_k)$
Hacer: $I=\{P_1,..P_k\}$

En general conviene elegir la descomposición más fina. Si no resultara posible realizar una descomposición de esta forma se toma $k=1$, o sea $I=\{P_1\} = \{P\}$

Para cada uno de los P_i debe introducirse un estado final q_i del cual será invariante. Por lo cual $\forall i: P_i \in I$ deberá cumplirse $q_i \in F \subseteq K$.

Paso 2: Extender I con una familia de predicados $\{P_h\}$ tales que: para cada α, a y P_i de I, si P_i se satisface para αa, debe haber un P_h en la familia, que se satisfaga para α. Cada vez que se agregue a I un nuevo predicado P_h, debe crearse un nuevo estado q_h del cual sea invariante.

Finalmente, ya sea que el P_h hallado haya sido incorporado en este paso o que ya perteneciera a I previamente, debe garantizarse que $(q_h, a) \succ^* (q_i,\lambda)$.
Esto puede lograrse:

- Definiendo $\delta(q_h, a) = \{q_i\}$ o agregando q_i a $\delta(q_h, a)$ si esta ya había sido definida.
- Sin necesidad de realizar ningún cambio si ya se verificaba que $(q_h, a) \succ^* (q_i,\lambda)$.
- Agregando transiciones espontáneas que permitan llegar de q_r a q_i si en I hay algún P_r satisfecho por αa tal que $(q_h, a) \succ^* (q_r, \lambda)$

Este procedimiento de extensión de I y correlativamente de K, debe continuarse hasta que no sea necesario incorporar ningún predicado nuevo, o sea hasta que :

$$\forall \alpha \; \forall a \; \forall Pi \in I : (P_i(\alpha a) \Rightarrow \exists P_h \in I : P_h(\alpha) \wedge (q_h, a) \succ^* (q_i, \lambda)))$$

Si se fracasa al tratar de extender I o se advierte que resulta infinito, analizar si C no es regular, caso en el que no existe solución al problema planteado; si se sospecha que efectivamente es regular, revisar la estrategia usada.

Paso 3: Determinar el estado inicial q_0 de M de la siguiente forma:

- si hay sólo un predicado $P_k \in I / P_k(\lambda)$, o sea que es satisfecho por la cadena vacía, elegirlo como estado inicial.
- si hay más de uno, o ninguno – caso en que el lenguaje es vacío -, agregar un estado nuevo q_0 con invariante $P_o \equiv (\alpha=\lambda)$ y definir transiciones λ de él a ellos hasta garantizar que de q_o se pueda acceder a cualquier q_i para el cual se verifica $P_i(\lambda)$.

2.2 En el diseño de Algoritmos

Se ha logrado reconstruir usando métodos que garantizan corrección una importante cantidad de algoritmos tanto destinados a la resolución de problemas como requeridos por demostraciones constructivas como los métodos de *parsing*, el algoritmo de Warshall, la minimización de autómatas y la conversión de distintos formalismos de representación.

3. Trabajos futuros

3.1 En la construcción de autómatas

Se tratará de extender el método a otros tipos de autómatas – no finitos-, en particular a los Autómatas Pila.

3.2 En el diseño de algorítmos

Se tratará de completar la reformulacion de las construcciones usadas en la Teoría de lenguajes Formales y Autómatas bajo el enfoque propuesto.

Se escribirán notas de los temas que se completen y quizás a partir de ellos se compagine un nuevo texto.

4. Referencias

1. "Introduction to automata theory, languages and computation" Hopcroft Ullman . Addisson Wesley 1979.
2. " The theory of Parsing Translation and Compiling". Aho, Ullman, Prentice Hall 1973
3. "lex & yacc" J. Levine, T. Mason & D Brown. O'Reilly & Associates 1992
4. "LEX a lexical analyzer generator, Lesk , CSTR 93, Bell Laboratories, 1975
5. Thompson K. , Regular expression search algorithm, Communications ACM 11:6 1968
6. "A Discipline of Programming". Dijkstra. Prentice Hall. 1978.
7. "Formal Develop of Programs". Dijkstra. Addison Wesley
8. "The Science of Programming". D. Gries. Springer Verlang 1981.
9. "Teoría de la Computación, Lenguajes formales, autómatas y complejidad" J.G. Brookshear. Addison Wesley 1993.
10. "The Theory and Practice of Compiler Writing". J. Tremblay, P. Sorenson. Mc Graw Hill 1985.
11. "Compiler Construction". W. Waite. G. Goos. Springer Verlang, 1984
12. "A Retargetable Compiler, Design and Implementation". C. Fraser, D. Hanson. The Benjamin/Cummings Publishing Company 1995.
13. "Building and Optimizing Compiler". R. Morgan. Butterworth Heinemann 1998.
14. "Lenguajes Gramáticas y Autómatas, un enfoque práctico". P. Isasi, P. Martínez, D. Borrajo. Addison Wesley 1997.
15. "Un método para el diseño de Autómatas basado en Invariantes". J. Aguirre, M. Arroyo. Anales del CACIC-2000, 2000.
16. "Modern Compiler Implementation in Java". A. Appel. Cambridge University Press 1998.