El uso de métodos que garanticen corrección por construcción en la enseñanza de la Teoría de Lenguajes y Autómatas

Jorge Aguirre
Universidad Nacional de Río Cuarto, Departamento de Computación, FCEFQyN,
Río Cuarto, Argentina, 5800
jaguirre@dc.exa.unrc.edu.ar

Marcelo Arroyo
Universidad Nacional de Río Cuarto, Departamento de Computación, FCEFQyN,
Río Cuarto, Argentina, 5800
marroyo@dc.exa.unrc.edu.ar

Resumen

En éste trabajo se analizan las ventajas de utilizar en la enseñanza de la Teoría de Lenguajes Formales los métodos de construcción de software correcto habitualmente usados en las asignaturas de construcción de algoritmos y programación, en lugar de la práctica habitualmente usada y seguida en la bibliografía, de presentar los algoritmos y luego dar para ellos una demostración de su corrección. Como una aplicación novedosa de ésta metodología se presenta el uso de un método para la construcción de autómatas finitos a partir del predicado que define su conjunto de aceptación. Dicho método, desarrollado por los autores, garantiza la corrección de los autómatas construidos. El método consiste en obtener a partir del predicado original, una familia de predicados y un conjunto de estados de los cuales los primeros son invariantes. A partir de ésta familia se define la función de transición y el estado inicial. Se incluye la aplicación del método a varias construcciones clásicas de la teoría de lenguajes y autómatas, entre ellas, el autómata que demuestra la propiedad fundamental de las gramáticas LR y a algunos ejemplos prácticos de construcción de Autómatas, tema que generalmente también es abordado en la misma asignatura.

Palabras clave: Autómatas Finitos, Cálculo de primer orden, Invariante, Corrección, Teoría de lenguajes formales.

Abstract

This work is concerned with the advantages of using methods for correct software construction in the teaching of Formal Languages Theory. That methods are usually taught in courses of programming and algorithms construction, but in Formal language theory courses the practice traditionally used and also the style followed by the bibliography, consist in: first to present the algorithms and then to construct their correctness proof. As a novel application of the first methodology, we present a method, developed by the authors, to construct finite automata from the characteristic predicate of its accepting set. If this method is followed, the correctness of the obtained finite automata is guaranteed. The method consists of obtaining a family of first order predicates and a set of states from the original predicate, in which each predicate is an invariant of a state. Next, from this predicate's family, the transition function and the initial state are defined. This paper also shows its application to give a constructive proof of several classic Formal Language and Automata theory results, including the LR grammars fundamental theorem. Some samples of the method application to practical cases of automata design are also included.

Keywords: Finite Automata, First Order Calculus, Invariant, Correctness, Formal Languages Theory.

Este trabajo ha sido desarrollado en el marco de proyectos subsidiados por la SECYT de la Universidad Nacional de Río Cuarto y por la Agencia Córdoba Ciencia.

1 Introducción

Generalmente las carreras de Ciencias de la Computación comienzan la formación de sus alumnos presentando 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]. Sin embargo, en las asignaturas de 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 correspondientes a las siguientes referencias [1], [2], [9], [10], [11]-, mientras que otros textos sólo presentan a los algoritmos, sin ocuparse de su corrección – como los que corresponden a las siguientes referencias [12], [13], [14], [16]-
-.
Presentar algoritmos y luego dar la demostración de su corrección tienen las siguientes desventajas:

- no se afianza 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.

Creemos que es conveniente realizar una revisión de los enfoques de la presentación de algoritmos y de su corrección introduciendo el uso de métodos de construcción que garantizan la corrección basados en heurísticas de diseño. Consideramos que este enfoque elimina las desventajas señaladas y permite que el alumno puede recrear la construcción de los algoritmos, que pierden su condición de elementos revelados. Idéntico problema se presenta en muchas demostraciones constructivas, donde el objeto es presentado y luego se da la demostración de que cumple las condiciones requeridas. Estos objetos muchas veces son algoritmos, en cuyo caso vale lo que hemos mencionado y también muchas otras son autómatas. También en este caso lo usual es presentar el autómata y luego dar una demostración recursiva de su corrección. En éste último caso no se pueden aplicar las metodologías antes mencionadas y si se quiere evitar la presentación del apriorística del algoritmo debería realizarse el doble trabajo de inducir su construcción intuitivamente primero y luego demostrar su corrección. En éste trabajo se presenta el uso de un método, desarrollado previamente por los autores [15], que da una heurística para la derivación de Autómatas Finitos a partir de su especificación mediante predicados de primer orden.

El método expuesto brinda una estrategia que facilita la obtención de soluciones, acercando la posibilidad de que sean recreadas por el alumno y garantiza la solución obtenida, no haciendo falta, cuando se lo aplica ninguna demostración adicional. El método se basa en la noción de invariante de estado y particularmente en la de *conjunto de invariantes fuertes (cief)*, que se describe posteriormente. El método parte de la especificación del lenguaje de aceptación del autómata finito 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 y, 3) el estado inicial. 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 la construcción de Autómatas, obviamente no solo se presenta por motivos teóricos, sino que suele aparecer, también en numerosas aplicaciones prácticas. Estos formalismos brindan la base para la implementación de algoritmos capaces de operar en tiempo lineal. Cuando se ha logrado modelar un problema mediante un Autómata Finito los resultados de la teoría de Autómatas permiten obtener mecánicamente la solución óptima, de manera tal que no es necesario tener en cuenta ninguna consideración de eficiencia en el proceso de diseño. En algunos 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, obteniéndose luego, a partir de ella, el autómata deseado; para lo cual se cuenta con algoritmos (5) y con diversas herramientas de software como *lex, flex* (3)(4), *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 e ingeniería, no son capaces de encontrar soluciones correctas para los ejemplos prácticos presentados en este trabajo. La habilidad de resolver problemas prácticos mediante la construcción de Autómatas Finitos también se adquiere usualmente en las asignaturas de Lenguajes Formales y Autómatas, en este trabajo también se presentan dos ejemplos de aplicación del método en este ámbito.

Este trabajo tiene la siguiente organización: en la sección 2 se aclara la notación utilizada; en la 3 se presenta el método de construcción usado; en la 4 se estudia su aplicación a resultados clásicos de la teoría de Automatas y Lenguajes Formales; en la 5 se muestra el uso del método en la construcción de AF para otros propósitos y en la 6 se exponen las conclusiones.

2 Notaciones utilizadas

2.1 Definición: Un *Autómata Finito, AF* es una tupla $< K, \Sigma, \delta, q_0, F >$, donde

K es el conjunto de estados
Σ el alfabeto
δ: $K \times (\Sigma \cup \{\lambda\}) \rightarrow P(K)$ es la función de transición
$F \subseteq K$ es el conjunto de estados finales
$q_o \in K$ es el estado inicial

2.2 Definición: Una *configuración* es un par de la forma $K \times \Sigma^*$.

2.3 Definición: Una *transición* es una relación sobre configuraciones:

$(q, a\ \alpha) \succ (r, \alpha)$ si y sólo si $r \in \delta(q, a)$ con $a \in \Sigma \cup \{\lambda\}$
donde M: AF, $\alpha \in L(M) \Leftrightarrow \exists f \in F$ tal que $(q_{0,} \alpha) \succ^* (f, \lambda)$

3 El método de diseño utilizado

3.1 Conjunto de invariantes de estado fuerte (*cief*)

Definición: 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 *cief* para M si

$$\forall \alpha \in \Sigma^* ((q_0,\alpha) \succ^* (q_i,\lambda) \Leftrightarrow P_i(\alpha))$$

Los *ciefs* tienen la importante propiedad siguiente –demostrada en un artículo anterior [8]- :

Si $I = \{Pi\}$ es un *cief* para M entonces $L(M) = \{\alpha / V_{qi \in F}\ P_i(\alpha)\}$

Esta propiedad será utilizada por el método usando al predicado que especifica el conjunto de cadenas aceptadas por el autómata finito a construir como el invariante de un único estado final – disyunción unaria – o descomponiendo dicho predicado en la disyunción de otros, cada uno de los cuales es invariante de uno de los estados finales del autómata.

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]):

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

2. 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.

3. 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

Dado $M=<K,\Sigma, \delta,q_0, F>:AF$, con $K=\{q_0, \ldots, q_n\}$
Un conjunto de predicados $\{P_i: \Sigma^* \rightarrow Bool / 0 \leq i \leq n\}$ es un ***cief*** para M si cumple:

$\forall i,j, \forall \alpha \in \Sigma^*$

$($ $(1:\ P_0(\lambda) \wedge (P_j(\lambda) \Rightarrow (q_0, \lambda) \succ^* (q_j, \lambda)) \wedge$

$(2:\ \forall a \in \Sigma \cup \{\lambda\}\ ((q_i,a) \succ^* (q_j,\lambda) \Rightarrow ((P_i(\alpha) \Rightarrow P_j(\alpha a)))) \wedge$

$(3:\ \forall a \in \Sigma\ (P_i(\alpha a) \Rightarrow \exists h : P_h(\alpha) \wedge (q_h, a) \succ^* (q_i, \lambda)))$

$)$

3.2 Construcción del autómata finito

Este método es útil para construir un autómata finito que acepte al conjunto de cadenas que satisfacen un predicado[1], o sea para dar una solución al siguiente problema:

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 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 $(q_h, a) \succ^* (q_r, \lambda)$ [2]

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 :

---

[1] Obviamente, para algunos P, el conjunto de cadenas $\alpha \in \Sigma^* / P(\alpha)$ no será regular, en cuyo caso no existirá ningún AF que lo acepte. El problema de determinar si un conjunto definido por un predicado es regular, es indecidible – no se puede resolver mediante un algoritmo – y por lo tanto este método configura una heurística, que puede fracasar en la búsqueda de la solución. En tal caso podrá intentarse probar que el conjunto no es regular - usando el lema de pumping [1] - y que por lo tanto no hay solución posible.

[2] El uso de esta última forma de asegurar que $(q_h, a) \succ^* (q_i,\lambda)$ permite reducir la cantidad de transiciones λ introducidas. Será usada en la sección 4.5.

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

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 conjunto 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 $P_i(\lambda)$.

3.3 Usando un AF para la construcción de otro.

Si se dispone de un autómata $M=<K, \Sigma, \delta, q_0, F>:AF$ y se lo quiere usar para la construcción de otro, o se encara una construcción modularmente; se puede hacer uso del conjunto de invariantes triviales:

A cada estado q_i esta asociado $P_i \equiv (q_0, \alpha) \succ^* (q_i, \lambda)$

Este conjunto es un *cief* para *M*.

4 Aplicaciones a la teoría de lenguajes y autómatas

4.1 Construir Autómatas Finitos para las expresiones regulares básicas: ∅, λ, a (a∈Σ)

1) Construir M tal que acepte al conjunto vacío

$M:<K, \Sigma, \delta, q_0, F> / L(M) = \varnothing$

Paso 1: $\varnothing=\{\alpha \in \Sigma^* / (P_1: false\}$ (el predicado característico de ∅ es *false* y no tiene sentido descomponerlo en disyuntos), $I=\{false\}$, $K=F=\{q_1\}$.

Paso 2: $\forall \alpha \ \forall a \ (P_1(\alpha a)=false)$ por lo que no es necesario introducir ningún P_h.

Paso 3: En *I* no hay predicado que se satisfaga para la cadena vacía, luego introducimos $(P_0: \alpha=\lambda)$ a *I* y q_0 a *K*. No hay que definir transiciones ya que P_0 no implica ningún P_i. Quedando $I=\{P_0,P_1\}$, $K=\{ q_0,, q_1\}$.

Figura 1: Autómata finito M obtenido para la ER: ∅

2) Construir M tal que acepte el conjunto formado por el carácter a

$M:<K, \Sigma, \delta, q_0, F> / L(M) = \{ a \}$

Paso 1: $\{a\}=\{\alpha \in \Sigma^* / P: \alpha=a\}$. $I = \{ P_1: (\alpha=a) \}$, $K=F=\{q_1\}$.
Paso 2: La única cadena y el único carácter que satisfacen $P_1(\alpha a)$ son *a* y λ luego hace falta agregar $P_2(\alpha) \equiv (\alpha=\lambda)$ a *I* , ya que $(\alpha=\lambda) \Rightarrow (\alpha a=a)$ y en concordancia q_2 *a K*. y definir $\delta(q_2,a)= q_1$.
Paso 3: Existe $P_2: \alpha=\lambda$ en *I* que se satisface para λ y es el único, entonces el estado inicial es q_0.

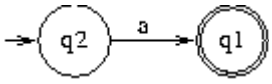

Figura 2: Autómata finito M obtenido para la ER a

3) Construir M tal que acepte el conjunto formado por la cadena vacía

M:<K, ∑, δ, q_0, F> / L(M) = { λ }

Paso 1: *{a}={α∈∑*/ P:α=a}. I = { P_1: (α=a) }, K=F={q_1}.*
Paso 2: Como no se cumple *P_1(α a)* para ningún *a* , no hace falta agregar más predicados ni estados.
Paso 3: *P_1* en *I* se satisface para *λ* y es el único, luego el estado inicial es *q_1*.

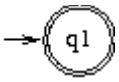

Figura 3: Autómata finito M que acepta la cadena vacía

4.2 Construcción de un AF equivalente a una gramática regular –GR-
Dada *G=<V_N , V_T, P, S>: GR,* construir *M= <V_T , ∑, δ, q_o, F>:AFD* tal que *L*(M) =*L*(G)
Se busca que *L(M)={α / S⇒*α ∧ α∈VT*}*
Como *G* es regular sus producciones serán de una de las siguientes formas

A→tB o
A→t con *A,B∈VN y t∈VT*

Por lo cual todas las cadenas derivables a partir de la raíz tendrán una de las siguientes formas

αA o *αt* donde, *A∈VN* , *α∈VT** y *t∈VT*.

Paso 1: El predicado característico de *L(M)* es

*P(α)= S⇒*α ∧ α∈VT**

Al cual se le asocia un estado final , resultando *I={P}, F= {qf}⊆K*

Paso 2: *P(αa)* implica *S⇒* α a* y esto a su vez

∃ A∈VN : S⇒ αA ⇒ α a (A → a)*

sea entonces *P_A ≡ S⇒* αA* - el *P_h* requerido por el método - y *q_A* el estado del cual es invariante, resultando conveniente definir *δ* de manera que *q_f ∈ δ(q_A,a),*
para cada *A∈V_N /A→ a* habrá que considerar los correspondientes *P_A* y *q_A*.

A su vez si

P_A(αa), que es equivalente a *S⇒* α aA,* entonces deberá verificarse que exista B tal que
S⇒ α B ⇒ α aA,*cumpliéndose *B→aA*

por lo tanto *P_B* es el predicado buscado que satisface *P_B(α) ⇒ P_A(α a).* A *P_B* debe asociársele un estado, que para simplificar llamaremos *q_B*.

Como esto debe considerarse para todos los posibles *P_X* y todas las cadenas *αX*, conviene definir

*I={P_A ≡ (∃ α : S⇒*α A) / A∈VN}, K={q_f } ⋃{q_A /A∈VN}*
y
δ(q_B, t) = { q_A / B→tA } ⋃ {q_f} si B→ t
{ q_A / B→tA } en otro caso

Con cuyas definiciones se cumplen las condiciones requeridas.

Paso 3: *P_S (λ) ≡ S⇒* λ S* se cumple y es el único que es verifica para *λ*, luego el estado inicial es *q_S*.

Nota: si *G* es reducida para cada *P_A (αa)* se encontrará un *P_H / P_H(α) ⇒* P_A (αa),* mientras que si G tiene símbolos inaccesibles esto puede no suceder, en cuyo caso el estado *q_A* también será inaccesible, o podrá ser eliminado junto con su invariante. Ej. *S→bS | b | a A*

4.3 Construcción de un autómata que acepte la unión de los lenguajes regulares A y B

Paso 1: *P* es $(\exists\, q_{Af} \in FA : (q_{Ao},\alpha) \succ A^* (q_{Af}, \lambda) \vee \exists\, q_{Bf} \in FB : (q_{Bo},\alpha) \succ B^* (q_{Bf}, \lambda)\}$

Como *FA* y *FB* son finitos

$P \equiv \bigvee_{q \in FA} ((qAo, \alpha) \succ A^* (q, \lambda)) \vee \bigvee_{q \in FB} ((qBo, \alpha) \succ B^* (q, \lambda))$

Resultando *I*, inicialmente formado por los invariantes triviales de cada estado final de *A* y de *B* y $F = FA \cup FB \subseteq K$

Paso 2: Basta agregar a *K* todos los estados de ambos *AF* y a *I* sus invariantes triviales
Resultando $K = KA \cup KB$.
y para que se cumpla (2) del *lema del cief*

$$\delta(q,a) = \begin{cases} \{\delta_A (q,a)\} & si\ \ q \in K_A \\ \{\delta_B (q,a)\} & si\ \ q \in K_B \end{cases}$$

Paso 3: Como los invariantes de ambos estados iniciales son satisfechos por la cadena vacía, se agrega un nuevo estado q_0 con invariante $P_0(\alpha) \equiv (\alpha = \lambda)$ y se definen las transiciones
$\delta(q_o, \lambda) = \{q_{Ao}, q_{Bo}\}$.
Con lo cual nuevamente se ha obtenido la solución clásica.

4.4 Intersección de dos lenguajes regulares

Dados dos lenguajes regulares L_A y L_B definidos por sus correspondientes AFDs, *A* y *B*.
$A = <KA, \Sigma, \delta_A, q_{Ao}, F_A>$, $B = <K_B, \Sigma, \delta_B, q_{Bo}, F_B>$ que cumplan $K_A \cap K_B = \varnothing$

Construir $M = <K, \Sigma, \delta, q_o, F>$:*AFD tal que* $L(M) = L_A \cap L_B$.
$LA \cap LB = \{\alpha \,/\, \exists\, q_{Af} \in K_A, \exists\, q_{Bf} \in K_B : (q_{Ao},\alpha) \succ^* (q_{Af}, \lambda) \wedge (q_{Bo},\alpha) \succ^* (q_{Bf}, \lambda) \}$

Paso 1. $P(\alpha)$ es $(\exists\, q_{Af} \in K_A, \exists\, q_{Bf} \in K_B : (q_{Ao},\alpha) \succ A^* (q_{Af}, \lambda) \wedge (q_{Bo},\alpha) \succ B^* (q_{Bf}, \lambda))$

Como K_A y K_B son finitos

$P(\alpha) \equiv \bigvee_{p \in FA,\ q \in FB} ((q_{Ao}, \alpha) \succ_A^* (p, \lambda) \wedge (q_{Bo}, \alpha) \succ_B^* (q, \lambda))$

Resultando I, inicialmente formado formado por todas las conjunciones que puedan formarse entre cada predicado trivial asociado aun estado de *FA* y cada predicado trivial asociado a un estado de *FB*.
Cada un de estos predicados se denotará

$Q_{p,r} \equiv (q_{Ao}, \alpha) \succ_A^* (p, \lambda) \wedge (q_{Bo}, \alpha) \succ_B^* (r, \lambda)$

Dada la correspondencia entre pares de estados finales y predicados de I conviene tomar
$F = F_A \times F_B \subseteq K \subseteq K_A \times K_B$

Paso 2. Para extender los predicados de manera que I sea cerrado respecto de la condición 3 del lema conviene considerar que

si se verifica $Q_{p,r}(\alpha a)$ también se verifica $(q_{Ao}, \alpha a) \succ_A^* (p, \lambda)$ y entonces $\exists\, h/(q_{Ao}, \alpha a) \succ_A^* (h, \lambda)$
y también $(q_{Bo}, \alpha) \succ_B^* (r, \lambda)$ por lo cual $\exists\, t/(q_{Bo}, \alpha a) \succ_B^* (t, \lambda)$

luego también se cumple $Q_{h,t}(\alpha)$ – es el P_h de la condición 3 del *lema del cief* – .
Hay que agregar $Q_{h,t}$ a *I* y se debe definir un estado del cual sea invariante, resulta claro definirlo como *(h,t)*.

La generalización de este razonamiento conduce a definir $K = K_A \times K_B$ y a la función de transición como sigue
$\delta((p,q),a) = \{|\delta_A(p,a)|, |\delta_B(q,a)|\}$ donde $|\{p\}| = p$ – los conjuntos de la imagen de δ son singulares por tratarse de *AFDs*.
La construcción realizada garantiza que

$(Q_{h,t}(\alpha) \Rightarrow Q_{p,q}(\alpha a)) \Rightarrow ((h,t), a) \succ_M^* ((p,q), \lambda)$.

por lo cual quedan cumplidas las condiciones pedidas por la condición (3) del lema del *cief.*

Paso 3. Como los invariantes de los estados iniciales se verifican para la cadena vacía, y son los únicos que lo hacen - por tratarse de Autómatas determinísticos - el estado inicial de *M* debe ser (q_{Ao}, q_{Bo}), con lo cual se completa la construcción.

4.5 Prefijos viables LR(0) de una gramática independiente del contexto (CFG)

En este caso el autómata que será diseñado constituye la demostración constructiva del teorema central de la teoría del parsing LR, esto es, que el conjunto de prefijos viables de una *G* es regular. Como es bien conocido, los prefijos viables son las cadenas que pueden aparecer en la pila de un reconocedor ascendente – shift - reduce – , por lo cual el hecho de que constituyan un lenguaje regular permite sustentar el algoritmo de reconocimiento en el correspondiente autómata finito.

Definiciones y propiedades del dominio

Dada una gramática $G=<V_N,V_T, P, S>$ independiente del contexto – *G*:CFG- y siendo
$V= V_N \cup V_T$
D 1) Derivación más a derecha ($\Rightarrow_R$): $\delta A\omega \Rightarrow_R \delta\alpha\omega$ *si* $A\rightarrow\alpha \wedge \omega\in V_T^*$
D2) Prefijo viable *(pv):* α es un *pv* si $\exists\delta,\rho.\omega,\beta\in V^* / \alpha=\delta\rho \wedge$
$S \Rightarrow_R^*\delta A\omega \Rightarrow_R \delta\rho\beta\omega$
D3) *Item:* $[A\rightarrow\rho.\beta] / A\rightarrow\rho\beta$, *Items:* $\{It :/ It:item\}$
D4) Item válido para (*iv*) una cadena: dada $\alpha\in V^*$
$[A\rightarrow\rho.\beta]$ *iv* α *si* $\exists\ \delta,\rho,\omega / \alpha=\delta\rho \wedge$
$S\Rightarrow_R^*\delta A\omega\Rightarrow_R\delta\rho\beta\omega$
P1) $\forall X\in V ([A\rightarrow\rho.X\beta]$ *iv* $\delta\rho \Leftrightarrow [A\rightarrow\rho X.\beta]$ *iv* $\delta\rho X)$
P2) $\forall X\in V_N\ \forall\eta: X\rightarrow\eta$ *(si* $[A\rightarrow\rho.X\beta]$ *iv* α *entonces* $[X\rightarrow.\eta]$ *iv* α *)*

Construcción.

Dada *G:CFG* reducida, se desea construir $M=<K, V, \delta, q_o, F>:AF$ tal que acepte
$PV=\{\alpha\in V/ \alpha$ *es pv de G}*

Paso 1. De la definición de prefijo viable –*pv*- e item válido –*iv*- resulta que α es un *pv* si y solo si existe algún item que sea válido para α, por lo cual
$PV=\{\alpha / \exists [A\rightarrow\rho.\beta]\in Items / [A\rightarrow\rho.\beta]$ *iv* $\alpha \}$

el predicado *P* es $P(\alpha) = (\exists It\in Items / It$ *iv* $\alpha)$

y como Items es finito *P* se puede expresar mediante la siguiente disyunción
$P(\alpha) = \bigvee_{It \in Items} It$ *iv* α o
$P(\alpha) = \bigvee_{It \in Items} P_{It}(\alpha)$, llamando $P_{It}(\alpha)$ al predicado *It iv* α

Como *P* es una disyunción convedrá tomar inicialmente $I= \{ P_{It} / It \in Items\}$ y
$F= Items \subseteq K$

Paso 2. Hay que lograr que para cada P_{It} para cada α y para cada *a*, haya un $P_{It'}$ tal que
$P_{It}(\alpha a) \Rightarrow P_{It'}(\alpha)$ y que $(It', a) \succ^* (It, \lambda)$.
Se dividirá el análisis en dos casos según que *It* sea de la forma $[A\rightarrow\rho a.\beta]$ o de la forma $[X\rightarrow .\eta]$.

1) Si $It= [A\rightarrow\rho a.\beta]$, $It'= [A\rightarrow\rho.a\beta]$ cumple la implicación, por *P1* y
debe definirse $\delta([A\rightarrow\rho a.\beta], a) = \{ [A\rightarrow\rho a.\beta]\}$

2) Si $It= [X\rightarrow .\eta]$ entonces It *iv* αa si y sólo si $S\Rightarrow_R^* \alpha aX\omega \Rightarrow_R \alpha a\eta\omega$.
Retrocediendo en esta última derivación – ver figura 4-, alguna vez ***a*** será reducida, resultando
$S \Rightarrow_R^* \alpha'B\omega'' \Rightarrow_R^* \alpha aX\omega \Rightarrow_R \alpha a\eta\omega$, con $\alpha=\alpha'\alpha''$ y $\omega=\omega'\omega''$
y si *k* es la longitud del camino de *B* a *X* se verificará

$B \Rightarrow_R \alpha''aY_1\omega'_1 \Rightarrow_R^* \alpha''aY_1\omega_1 \Rightarrow_R$ (donde los ω'_i pueden contener no terminales, pero $\forall i: \omega_i \in V_T^*$)
$\alpha''aY_2\omega'_2\ \omega_1 \Rightarrow_R^* \alpha''aY_2\omega_2\ \omega_1 \Rightarrow_R$.

. . .

$\alpha''aY_{k-1}\omega_{k-1}..\omega_1 \Rightarrow_R \alpha''aX\omega_k..\omega_1$ done $Y_i \in V_N$ y $\omega_k..\omega_1 = \omega'$ [3],

de aquí resulta que puede tomarse como *It´* a $[B \to \alpha''.\ aY_1\omega'_1]$ ya que es *iv* $\alpha = \alpha'\alpha''$.
A su vez se cumple que $[B \to \alpha''.\ aY_1\omega'_1]$ *iv* $\alpha = \alpha'\alpha''$ y por la construcción del caso 1:

$([B \to \alpha''a.Y_1\omega'_1], a) \succ ([B \to \alpha''.\ aY_1\omega'_1], \lambda)$.

Además, por P2

$[Y_1 \to .Y_2\omega_2]$ *iv* $\alpha a, .. , [Y_1 \to .Y_{k-1}\omega_{k-1}]$ *iv* αa

Esto permite definir

$\delta([A \to \rho.X\beta], \lambda) = \{[X \to .\eta] / X \to \eta\}$ para cada $[A \to \rho.X\beta] / X \in V_N$ (i)

y con esto basta para que $([B \to \alpha''.\ a\ Y_1\omega'_1], a) \succ^* ([X \to .\eta], \lambda)$ o sea $(It', a) \succ^* (It, \lambda)$, ya que

$([B \to \alpha''.\ a\ Y_1\omega'_1], a) \succ ([B \to \alpha''\ a\ .Y_1\omega'_1], \lambda) \succ^* ([X \to .\eta], \lambda)$ [4].

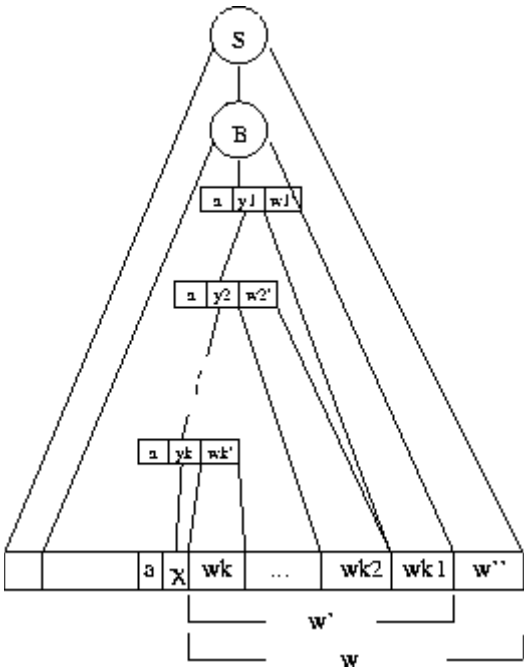

Figura 4: Paso 2, Autómata de prefijos viables

Paso 3. Como *It iv* λ si es de la forma $[S \to .\sigma]$ o de la $[X \to .\eta]$ tal que $\sigma \Rightarrow_R^* X\omega$
bastará incorporar q_o con invariante $(\alpha = \lambda)$ y definir

$\delta(q_o, \lambda) = \{ [S \to .\sigma] / S \to \sigma\}$ para garantizar que

$\forall P_{It}: P_{It}(\lambda) : (q_o, \lambda) \succ^* (It, \lambda)$

ya que (i) garantiza que si $\sigma \Rightarrow_R^* X\omega$ el autómata se mueve espontáneamente de q_o a $[X \to .\eta]$

5- Aplicación al diseño de AFs en general

5.1 Cadenas binarias con cantidad impar de ceros y par de unos

Se pretende construir un autómata finito M que acepte el conjunto de cadenas binarias cuya cantidad de ceros es impar y cuya cantidad de unos es par.
Más formalmente, se desea construir $M: AF / L(M) = L$ sobre $\Sigma = \{0, 1\}$

[3] Observese que como se trata de derivación más a derecha siempre debe derivarse Y_i y ω_i sólo puede estar compuesta por terminales.
[4] Vale la misma consideración de la llamada anterior para $([B \to \alpha''\ a\ .Y_1\omega'_1], \lambda) \succ^* ([X \to .\eta], \lambda)$.

donde $L=\{ \alpha \in \Sigma^* / \ | \Pi_{\{0\}}(\alpha) | es\ impar \wedge | \Pi_{\{1\}}(\alpha) | es\ par \}$ [5]

Paso 1. Se quiere aceptar al conjunto de cadenas que satisface la conjunción.

$P= I0 \wedge P1$ donde: $I0$ es "α tiene cantidad impar de ceros" y $P1$ es "α tiene cantidad par de unos" o
$I0 = (| \Pi_{\{0\}}(\alpha) | es\ impar)$ y $P1 = (| \Pi_{\{1\}}(\alpha) | es\ par)$

Como no surge ninguna descomposición natural de esta conjunción en una disyunción se toma un sólo $P_1 \equiv P$ y se crea el correspondiente etsado q_1 que lo tiene como invariante, resultando: $I=\{P_1\}$, $F=\{I0, P1\} \subseteq K$.

Paso 2 .

Si $I0 \wedge P1$ se satisface para αa
para α se satisface $\neg I0 \wedge P1$ *si a=0 y*
$I0 \wedge \neg P1$ *si a=1*

Resultando que hay agregar a I los predicados $(\neg I0 \wedge P1)$ y $(I0 \wedge \neg P1)$ para las distintas posibilidades de a, los correspondientes estados, de los cuales sean invariantes y las transiciones correspondientes, o sea:
q_2 con invariante $(\neg I0 \wedge P1)$,
q_3 con $(I0 \wedge \neg P1)$ y
$\delta(q_2, 0)=\{q_1\}$, $\delta(q_3 , 1)= \{q_1\}$

Extendiendo el razonamiento se obtiene que finalmente I estará integrado por todas las conjunciones aplicadas a las combinaciones de $\{P0,\neg P0\}$ con $\{P1,\neg P1\}$, a cada una las cuales hay que asociar un estado, resultando pues conveniente definir

$K= \{I0,\neg I0\} \times \{P1,\neg P1\}$ (si $q_i = (R,S)$ su invarainte P_i resulta $R \wedge S$) y
$\delta((R, S), 0) = \{(\neg R,S)\}$, $\delta((R,S), 1) =\{(R,\neg S)\}$

Paso 3. Construcción del estado inicial. Como tanto la cantidad de ceros como la de unos de la cadena vacía es cero y por lo tanto par, resulta que λ satisface $(\neg I0 \wedge P1) \in I$.

Además $(\neg I0 \wedge P1)$ es el único predicado que es satisfecho por la cadena vacía y es invariante de Q_3, por lo cual se toma Q_3 como estado inicial.

Con esto finaliza la construcción de M – figura 5- garantizándose que I por construcción constituya un *cief* para $M=<\Sigma,K,\delta,Q_3,F>$ por lo queda también demostrado que

$L(M) = \{\alpha / P_o(\alpha)\} = \{\alpha / I0 \wedge P1\} = L$.

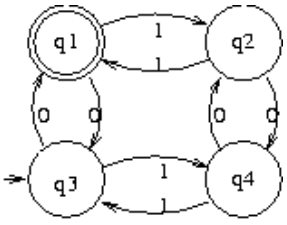

Figura 5. Cadenas binárias con cantidad par de unos e impar de ceros.

5.2 Cadenas decimales que representan números múltiplos de tres

Definiciones y propiedades del dominio que son usadas
$\Sigma=\{0,..9\}$, $v: \Sigma \rightarrow N$ / $v(0)=0,..v(9)=9$; $val:\Sigma^* \rightarrow N$ / $val(a_n .. a_0) = \sum_{i=0..n} v(a_i).10^i$ y
$val(\lambda)=0$

[5] $\Pi_C(\alpha)$ denota la proyección de α sobre C o sea
$\Pi: P(\Sigma) \times \Sigma^* \rightarrow \Sigma^*$ / $\Pi_C(\lambda)=\lambda$
$\Pi_C(a\alpha)=a\Pi_C(\alpha)$ *si* $a \in C$
$\Pi_C(a\alpha) = \Pi_C(\alpha)$ en otro caso

$s \equiv_3 t$ *sii* $mod(s,3)=mod(t,3)$; $\quad s +_3 t = mod(s+t, 3)$;
$val(a_n .. a_0) \equiv_3 v(a_n) +_3 . . +_3 v(a_0)$

El lenguaje buscado es $L = \{\alpha / mod(val(\alpha), 3)=0\} = \{\alpha / val(\alpha) \equiv_3 0\} =$
$\{\alpha=a_n..a_0 / v(a_o) +_3 .. +_3 v(a_n) = 0\} \cup \{\lambda\}$

Construcción:

Paso 1. Se comienza el análisis a partir de $P = val(\alpha) \equiv_3 0$.
No aparece ninguna manera razonable de descomponer P en una disyunción por lo cual se toma $I=\{P\}$ y se crea un estado q_o, del cual es invariante, q_o que debe ser final, resultando $I=\{P\}, F=\{q_o\} \subseteq K$

Paso 2. $val(\alpha a) \equiv_3 0$ entonces $\quad si\ a \equiv_3 0$ resulta $val(\alpha) \equiv_3 0$
$si\ a \equiv_3 1$ resulta $val(\alpha) \equiv_3 2$
$si\ a \equiv_3 2$ resulta $val(\alpha) \equiv_3 1$
resultando para α tres predicados , el primero de los cuales coincide con P. También resultan las correspondientes transiciones

$P_{0=}P = \quad val(\alpha) \equiv_3 0 \quad \delta(q_0, \underline{0})=\{q_0\}$
$P_1 \quad = \quad val(\alpha) \equiv_3 1 \quad \delta(q_1, \underline{2})=\{q_0\}$
$P_2 \quad = \quad val(\alpha) \equiv_3 2 \quad \delta(q_2, \underline{1})=\{q_0\}$

donde $\underline{0} = \{0, 3, 6, 9\}$
$\underline{1} = \{1, 4, 7\}$
$\underline{2} = \{2, 5, 8\}$

I se extiende a $\{P_0, P_1, P_2\}$, *K a* $\{q_0, q_1 q_2\}$ *siendo* P_i *el invariante de* q_i. *No hace falta extender más, pues se verifica*

$\forall\alpha\ \forall a \quad \forall P_j \in I : (P_j(\alpha a) \Rightarrow \exists P_i : P_i(\alpha))$

pero debe completarse δ para que, para cada P_j y cada P_i se verifique que $(q_i, a) \succ^* (q_j, \lambda)$, por lo cual la función de transición δ queda definida como lo muestra la figura 6.

Paso 3. Como está establecido que $val(\lambda)=0$ el estado inicial deberá ser q_0,complentándose la construcción de M.

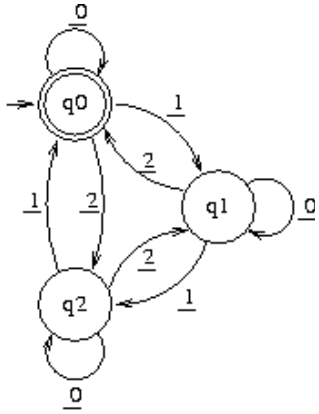

Figura 6. Cadenas decimales que representan múltiplos de tres

6 Conclusiones y trabajos futuros.

El enfoque de utilizar metodologías que permitan derivar soluciones correctas, ha sido usado por los autores en varios cursos de Teoría de Autómatas y Lenguajes, ubicados en el último de carreras de licenciatura en Ciencias de Computación. Se ha notado que pese a la práctica de uso de dichas metodologías realizada en los primeros años de esas carreras, los alumnos no tienen tendencia a utilizarlas en un nuevo contexto. Esta situación parecería convalidar la necesidad de insistir y abordar todos los problemas de construcción o diseño de soluciones de una manera uniforme y sistemática, de manera tal que finalmente el alumno adquiera la capacidad, y el hábito de apoyar su intuición en el proceso de construcción de soluciones, en razonamientos rigurosos – con mayor o menor grado de formalismo -, no sólo para asegurar la corrección de la solución obtenida, sino también para ayudarse en le proceso

mismo de su concepción. Aquí se presenta en particular un método que basa la construcción de Autómatas Finitos en la noción de invariante de estado; pero la noción de invariante atraviesa transversalmente, dando soporte al pensamiento, a toda la problemática de construcción sistemas formales comprendida en estas asignaturas; tanto en la elaboración de gramáticas o de otros tipos de Autómatas, como en el diseño de algoritmos.

La utilización del método presentado en dos cursos de Teoría de Lenguajes y Autómatas ha mostrado que es útil para los propósitos pedagógicos enunciados. Los alumnos que se apoyan en él llegan a resolver problemas que pocos alcanzan sin su apoyo. Una excesiva formalización, sin embargo suele crear resistencia a su uso, en tal sentido, sobre todo al comienzo de su aplicación puede resultar conveniente expresar los predicados en lengua natural, y en todos los casos se debe buscar para ellos una denotación comprensible y coherente con la que se use para los estados. También resulta conveniente sustentar las construcciones en diagramas, buscando que en ellos se encuentre toda la información necesaria. El diagrama debe hacer explícita la relación entre estados e invariantes sin necesidad de que se tenga que recurrir a información externa para determinarla.

Creemos que sería interesante extender el método a otros tipos de Autómatas. También contar con un texto que presente todas las construcciones de la Teoría de Lenguajes y Autómatas bajo la perspectiva de las heurísticas de derivación de soluciones correctas.

7 Referencias Bibliográficas

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.