Un método de diseño de autómatas, guiado por invariantes.

Jorge Aguirre[1] y Marcelo Arroyo[2]
Area Computación FCEFQyN Universidad Nacional de Río Cuarto[3]

Palabras clave: Autómatas Finitos, Cálculo de primer orden, Invariante, Corrección

Resumen

En este trabajo se describe un método de diseño de Autómatas Finitos a partir del predicado de primer orden que define su conjunto de aceptación. El método se basa en construir lo que aquí se ha denominado un Conjunto de Invariantes de Estado Fuerte, *cief*. Los invariantes son predicados sobre las cadenas de entrada, asociados a los estados. La construcción se realiza estudiando cómo se transforman los predicados al agregar un carácter a su argumento. Simultáneamente se introducen los estados de los cuales dichos predicados son invariantes y se define la función de transición. El método determina los estados finales y construye el inicial. Según las decisiones de diseño adoptadas se puede arribar a un Autómata Determinístico o No Determínistico. Aquí se definen los conjuntos de invariantes de Estado, se demuestran sus propiedades fundamentales y se caracteriza a aquellos que son *cief*s. El método asegura la corrección de las construcciones. Se incluye su aplicación en varios casos de estudio.

[1] jaguirre@dc.uba.ar
[2] marryo@dc.exa.unrc.edu.ar
[3] Trabajo realizado en el marco de un proyecto subsidiado por la Secretaría de Ciencias y Técnica de la Universidad Nacional de Río Cuarto.

Un método de diseño de autómatas, guiado por invariantes.

1.- Introducción

Frecuentemente, tanto en aplicaciones prácticas como en la obtención de resultados teóricos, se plantea la necesidad o la conveniencia de construir Autómatas Finitos. 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. En tal caso se cuenta con algoritmos para obtener la solución buscada y existen diversas herramientas de soft para hacerlo, como lex y flex 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 o ingeniería, no son capaces de encontrar soluciones correctas para algunos de los casos de estudio presentados en este trabajo.

El método aquí propuesto brinda una estrategia de construcción que facilita la obtención de soluciones y además garantiza la corrección de la solución obtenida no haciendo falta ninguna demostración adicional.

El método procede de atrás hacia delante, determinando primero, a partir del predicado que deben cumplir las cadenas del lenguaje, los estados finales y sus invariantes y retrocediendo a partir de ellos se construyen los estados que serán alcanzados por los distintos prefijos junto con sus invariantes. Finalmente se determina el estado inicial. 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".

En las primeras secciones de este trabajo se introduce el concepto de conjunto de invariantes de estado fuerte y el método propuesto, que se basa en él. En la sección 5 se analizan algunos casos de estudio. La sección 6 contiene la definición de un conjunto de invariantes más general y la demostración de las propiedades de los conceptos introducidos, las cuales garantizan la corrección de los autómatas construidos.

2- Definiciones de conceptos usados

Autómata Finito, AF: $< K, \sum, \delta: K \times (\sum \cup \{\lambda\}) \rightarrow P(K), q_0 \in K, F \subseteq K >$

Donde K es el conjunto de estados, $\sum$ el alfabeto, δ la función de transición, F el conjunto de estados finales y q_o el estado inicial

Configuración: $K \times \sum^*$, transición $(q, a\,\alpha) \succ (r, \alpha)$ sí $r \in \delta(q, a)$ con $a \in \sum \cup \{\lambda\}$

Siendo M: AF, $\alpha \in L(M) \Leftrightarrow \exists\, f \in F$ tal que $(q_0, \alpha) \succ^* (f, \lambda)$

Autómata Finito Determinístico, AFD: AF / $\forall\, q \in K\ (\ \forall\, a \in \sum\ (\ \#\delta\ (q,a) = 1) \wedge \delta(q,\lambda) = \phi\ )$

Se usa esta definición que incluye a los autómatas Determinísticos en los no determinísticos para evitar la duplicación de definiciones y propiedades.

Proyección de una cadena sobre un conjunto: : Π: *P* ($\sum$) x $\sum^* \rightarrow \sum^*$ / $\Pi_C(\lambda) = \lambda$, $\Pi_C(a\alpha) = \Pi_C(\alpha)$ si $a \notin C$ $\Pi_C(a\alpha) = a\,\Pi_C(\alpha)$ si $a \in C$. Ejemplo $\Pi_{\{0,1\}}(01321) = 011$.

3-Conjunto de Invariantes de Estado fuerte

En este trabajo se introduce el concepto de conjunto de invariantes de estado fuerte *cief* para los Autómatas Finitos. Los *ciefs* tienen la importante propiedad siguiente – demostrada en la sec. 6- :

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

Dado un autómata finito M, un conjunto de predicados sobre las cadenas de su alfabeto {Pi} tal que cada Pi está asociado a un correspondiente estado q_i es un - ***cief*** –si cumple las tres condiciones siguientes – demostrado en sec. 6 -:

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 i se satisfaga para la cadena α implica que el invariante 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 $M=<K,\sum, \delta, q_0, F>$:AF, con $K=\{q_0, \ldots, q_n\}$,
Un conjunto de predicados $\{P_i: \sum^* \rightarrow Bool / 0 \le i \le n\}$ es un ***cief*** para M si cumple:
$\forall i,j, \forall \alpha \in \sum^*$

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

4.- El método de diseño propuesto

Este método es útil para construir un autómata finito que acepte al conjunto de cadenas que satisfacen un predicado[4], o sea para dar una solución al siguiente problema:

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

---

[4] Obviamente, dicho conjunto puede no ser regular y en tal caso no existirá ningún AF que lo acepte. El problema de determinar si es regular un conjunto definido por un predicado, 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 y por ende no hay solución posible, usando el lema de pumping. [1]

Este método garantiza la corrección del autómata construido, de manera tal que si se diseña un autómata siguiéndolo, no hace falta ninguna demostración adicional.

El método propuesto es útil tanto para el diseño de autómatas a ser usados en implementaciones concretas de productos de soft, como en la de aquellos requeridos por necesidades teóricas. En los casos de estudio será usado en ambos sentidos.

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 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:

E1: Construcción de los invariantes de los estados finales.
Tratar de expresar P como una disyunción de predicados .

$$P = V_{i=1,\,k}\ P_i$$

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 $q_i \in F \subseteq K$.

E2: Extender I a una familia de predicados $\{P_h\}$ tales que para cada α y para cada P_i de I que se satisfaga para αa haya en ella un P_h que se satisfaga para α. Para cada P_h nuevo incorporado a I debe crearse un estado también nuevo q_h del cual será 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 espontaneas que permitan llegar de q_r a q_i si en I hay algún P_r satisfecho por αa tal que $(q_h, a) \succ^* (q_r, \lambda)$[5],

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 :

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

[5] El uso de esta última forma de asegurar que $(qh, a) \succ^* (qi,\lambda)$ permite reducir la cantidad de transiciones λ introducidas. Será usada en el caso de estudio 4 para arribar a la solución clásica.

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.

E3 Determinar el estado inicial q_0 de M. De la siguiente forma

- si hay solo 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 $Pi(\lambda)$.

5.- Casos de estudio

Caso 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 $\sum = \{0, 1\}$

$$\text{donde } L=\{ \alpha \in \sum^* / \ | \Pi_{\{0\}}(\alpha) | \text{ es impar} \wedge | \Pi_{\{1\}}(\alpha) | \text{ es par } \}$$

Etapa 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

$$I0 = (| \Pi_{\{0\}}(\alpha) | \text{ es impar }) \quad y \quad P1 = (| \Pi_{\{1\}}(\alpha) | \text{ es par })$$

Como no surge ninguna descomposición natural de esta conjunción en una disyunción se toma un sólo $P_1 = P$ y se crea el correspondiente q_1 que lo tiene como invariante.

Etapa 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:
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 de ellas hay que asociar un estado, Resultando pues conveniente definir:

K= {I0,¬I0} x {P1,¬P1} (si q_i = (R,S) su invarainte P_i resulta R ∧ S) y
δ ((R, S), 0) = {(¬R,S)}, δ((R,S), 1) ={(R,¬S)}

resultando:

	δ (q, a) [6]	
q	A=0	A=1
Q_1=(I0 , P1)	Q_3= (¬I0, P1)	Q_2= (I0, ¬ P1)
Q_2=(I0,¬P1)	Q_4= (¬I0,¬P1)	Q_1= (I0 , P1)
Q_3=(¬I0 , P1)	Q_1= (I0, P1)	Q_4= (¬I0,¬ P1)
Q_4=(¬I0,¬P1)	Q_2= (I0 ,¬ P1)	Q_3= (¬I0, P1)

Etapa 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 (¬I0 ∧ P1) ∈ I .

Además (¬I0 ∧ 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 garantizándose que I por construcción constituya un *cief* para M=<∑,K,δ,Q_3,F>
Quedando asegurado que

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

Caso 2: cadenas decimales que representan naturales múltiplos de tres

Definiciones y propiedades del dominio usadas

∑={0,..9}, v: ∑→N / v(0)=0,..v(9)=9; val:∑*→N / $val(a_n .. a_{0)} = \sum_{i=0..n} v(a_i).10^i$ y
val(λ)=0
$s \equiv_3 t$ sii mod(s,3)=mod(t,3); $s +_3 t$ = mod(s+t, 3);
$val(a_n .. a_0) \equiv_3 v(a_n) +_3 . . +_3 v(a_0)$

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

Construcción:
Etapa 1. Se comienza el análisis a partir de P = val (α)$\equiv_3$ 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, que debe ser final resultando F={q_o}⊆ K

Etapa 2. val (αa)$\equiv_3$ 0 entonces si a $\equiv_3$ 0 resulta val (α)$\equiv_3$ 0
si a $\equiv_3$ 1 resulta val (α)$\equiv_3$ 2
si a $\equiv_3$ 2 resulta val (α)$\equiv_3$ 1

[6] Se omiten los delimitadores de conjuntos.

resultando para α tres predicados , el primero de los cuales coincide con P. También las correspondientes transiciones

$P_0 = P = val(\alpha) \equiv_3 0 \quad \delta(q_0, \underline{0}) = \{q_0\}$
$P_1 = val(\alpha) \equiv_3 1 \quad \delta(q_1, \underline{2}) = \{q_0\}$
$P_2 = val(\alpha) \equiv_3 2 \quad \delta(q_2, \underline{1}) = \{q_0\}$

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 haciendo falta extenderlo más, pues se verifica

$\forall\alpha\ \forall a\ \ \forall P_j \in I : (P_j(\alpha a) \Rightarrow \exists P_i : P_i(\alpha))$ 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 puede tomarse $K = \{q_0, q_1, q_2\}$, siendo P_i el invariante de q_i y la función de transición δ queda definida según la tabla siguiente..

	δ [7]		
K	$\underline{0}$	$\underline{1}$	$\underline{2}$
q_0	q_0	q_1	q_2
q_1	q_1	q_2	q_0
q_2	q_2	q_0	q_1

Etapa 3. Como está establecido que val(λ)=0 el estado inicial deberá ser q_0,complentándose la construcción de M.∎

Caso 3: Unión de dos lenguajes regulares

A continuación se aplica el método al clásico problema de construir un autómata para la unión de dos lenguajes regulares [5].

Supónganse que se tienen dos lenguajes regulares L_A y L_B definidos por sus correspondientes autómatas A y B.
$A = \langle K_A, \sum, \delta_A, q_{Ao}, F_A \rangle$, $B = \langle K_B, \sum, \delta_B, q_{Bo}, F_B \rangle$ que cumplen $K_A \cap K_B = \varnothing$

Se quiere obtener un autómata que acepte $L_A \cup L_B$.
$L_A \cup L_B = \{\alpha / \ \exists q_{Af} \in K_A : (q_{Ao}, \alpha) \succ^* (q_{Af}, \lambda) \vee \exists q_{Bf} \in K_B : (q_{Bo}, \alpha) \succ^* (q_{Bf}, \lambda)\}$

Etapa 1. P es $(\exists q_{Af} \in K_A : (q_{Ao}, \alpha) \succ_A^* (q_{Af}, \lambda) \vee \exists q_{Bf} \in K_B : (q_{Bo}, \alpha) \succ_B^* (q_{Bf}, \lambda))$

Como K_A y K_B son finitos

[7] Se omiten los delimitadores de conjuntos

$P \equiv V_{q \in FA}\ ((q_{Ao}, \alpha) \succ_A^* (q, \lambda)) \vee V_{q \in FB}\ ((q_{Bo}, \alpha) \succ_B^* (q, \lambda))$
Resultando I, inicialmente formado por un P_i por cada estado final de A y de B que - los miembros de la unión anterior -, que se denotaran P_{Ai} y P_{Bi} respectivamente.

Etapa 2. Para extender los predicados de manera que I sea cerrado respecto de (3) hay que considerar un predicado por cada estado de cada uno de los dos autómatas
$P_{Ai} = (q_{A0}, \alpha) \succ_A^* (q_{Ai}, \lambda)$ y $P_{Bi} = (q_{Bo}, \alpha) \succ_B^* (q_{Bi}, \lambda)$.

Resultando conveniente definir $K = K_A \cup K_B$. Para cada estado q_{Xi} su invariante es P_{xi}.
Para que se cumpla (2) la función de transición deberá ser

$$\delta(q,a) = \{\delta_A(q,a)\} \text{ si } q \in K_A$$
$$\{\delta_B(q,a)\} \text{ si } q \in K_B$$

Etapa 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
$\delta(q_0, \lambda) = \{q_{Ao}, q_{Bo}\}$
Con lo cual se ha obtenido la solución clásica.

Caso 4: 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 CFG 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 $A \to \alpha \wedge \omega \in V_T^*$
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: $[A \to \rho.\beta] / A \to \rho\beta$, Items: {It :/ It:item}
D4) Item válido para (*iv*) una cadena: dada $\alpha \in V^*$
$[A \to \rho.\beta]$ *iv* α si $\exists\ \delta, \rho, \omega / \alpha = \delta\rho \wedge$
$S \Rightarrow_R^* \delta A\omega \Rightarrow_R \delta\rho\beta\omega$
P1) $\forall X \in V\ (\ [A \to \rho.X\beta]$ *iv* $\delta\rho \Leftrightarrow [A \to \rho X.\beta]$ *iv* $\delta\rho X\ )$
P2) $\forall X \in V_N\ \forall \eta: X \to \eta$ (si $[A \to \rho.X\beta]$ *iv* α entonces $[X \to .\eta]$ *iv* α)

Construcción.

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

Etapa 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

$PV=\{\alpha / \exists [A\rightarrow\rho.\beta]\in$ Iems / $[A\rightarrow\rho.\beta]$ *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
$P(\alpha) = V_{It \in Items}$ It *iv* α o
$P(\alpha) = V_{It \in Items} P_{It}(\alpha)$, llamando $P_{It}(\alpha)$ al predicado It *iv* α

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

Etapa 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 ssi $S\Rightarrow_R^* \alpha aX\omega \Rightarrow_R \alpha a\eta\omega$. Retrocediendo en esta última derivación, alguna vez **a** será reducida, resultando $S \Rightarrow_R^* \alpha' B\omega'' \Rightarrow_R^k \alpha aX\omega \Rightarrow_R \alpha a\eta\omega$, con $\alpha=\alpha'\alpha''$ y $\omega=\omega'\omega''$, para algún $k \geq 0$ y
$B \Rightarrow_R \alpha''aY_1\omega'_1 \Rightarrow_R \alpha''aY_2\omega'_2\omega'_1 \Rightarrow_R \ldots \alpha''aY_{k-1}\omega'_{k-1}..\omega'_1 \Rightarrow_R \alpha''aX\omega_k..\omega_1$ siendo $k \geq 0$, $Y_i\in V_T$ y $\omega_k..\omega_1= \omega'$ [8], de aquí resulta que puede tomarse como It' a $[B\rightarrow\alpha''.\ a\ Y_1\omega'_1]$ ya que es *iv* $\alpha=\alpha'\alpha''$.
A su vez se cumple que $[B \rightarrow \alpha''.\ aY_1\omega'_1]$ *iv* $\alpha=\alpha'\alpha''$ y por la construcción del caso 1:
$([B \rightarrow \alpha''a.Y_1\omega'_1], a) \succ ([B \rightarrow \alpha''.\ aY_1\omega'_1], \lambda)$.
Además, por P2
$[Y_1\rightarrow .Y_2\omega_2]$ *iv* $\alpha a, .., [Y_1\rightarrow .Y_{k-1}\omega_{k-1}]$ *iv* αa
Esto permite definir
$\delta([A\rightarrow\rho.X\beta], \lambda) = \{[X\rightarrow.\eta] / X\rightarrow\eta\}$ para cada $[A\rightarrow\rho.X\beta] / X\in V_N$ (i)
y con esto basta para que $([B\rightarrow\alpha''.\ a\ Y_1\omega'_1], a) \succ^* ([X\rightarrow.\eta],\lambda)$ o sea $(It', a) \succ^* (It, \lambda)$, ya que
$([B\rightarrow\alpha''.\ a\ Y_1\omega'_1], a) \succ ([B\rightarrow\alpha''\ a\ .Y_1\omega'_1], \lambda) \succ^* ([X\rightarrow.\eta], \lambda)$ [9]. ▌

Etapa 3. Como It *iv* λ si es de la forma $[S\rightarrow .\sigma]$ o de la $[X\rightarrow.\eta]$ tal que $\sigma\Rightarrow_R^* X\omega$.
bastará incorporar q_o con invariante $(\alpha=\lambda)$ y definir
$\delta(q_o,\lambda)= \{ [S\rightarrow .\sigma] / [S\rightarrow \sigma] \}$ 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 $[X\rightarrow.\eta]$
▌

6.- Fundamentos – definiciones y propiedades –

Definición 1 Conjunto de invariantes de estado *cie*

Dado $M=<K,\sum, \delta,q_0, F>$: AF (Autómata Finito), con $K=\{q_0, \ldots ,q_n\}$,

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

Un conjunto de predicados $\{P_i: \sum^* \rightarrow Bool / 0 \leq i \leq n\}$es un conjunto de invariantes de estado para M si

$$\forall \alpha \in \sum^* ((q_0,\alpha) \succ^* (q_i,\lambda) \Rightarrow P_i(\alpha))$$

O sea cada propiedad Pi se cumple siempre que el estado qi es alcanzado

Propiedad 1
Dado M=<K,∑, δ,q_0, F>:AF (Autómata Finito), con K={q_0, . . ,q_n}

I= $\{Pi\}_{i=1..n}$ *cie* para M $\Rightarrow$ $L(M) \subseteq \{\alpha / V_{qi\in F}\ Pi(\alpha)\}$

Demostración

$\alpha \in L(M) \Rightarrow (q_0,\alpha) \succ^* (q_f,\lambda)$ para algún $q_f \in F$

entonces por ser I un *cie* para M se cumplirá $P_f(\alpha)$ por lo cual también se cumplirá

$\alpha \in \{\alpha / V_{qi\in F}\ P_i(\alpha)\}$.
∎

Sin embargo que I sea un *cie* para M no implica que $L(M) = \{\alpha / V_{qi\in F}\ P_i(\alpha)\}$
En efecto sea M:AF tal que *L*(M)sea un subconjunto propio de ∑* e

$I = \{P_i(\alpha) / \forall\alpha\ P_i = True\}$.

I es un *cie* para M pero cualquier cadena de ∑* satisface $V_{qi\in F}$ Pi(α)

Por lo cual $L(M) \neq \{\alpha / V_{qi\in F}\ Pi(\alpha)\}$ ∎

Conjunto de Invariantes de estado fuerte

Para lograr que *L*(M) sea igual a $\{\alpha / V_{qi\in F}\ Pi(\alpha)\}$ o sea que la disyunción de los invariantes correspondientes a los estados finales sea la función característica de *L*(M) se introduce una modificación en la definición de invariantes de acuerdo a la siguiente definición.

Definición 2. Conjunto de invariantes de estado fuerte *cief*

Dado M=<K, ∑, δ, q_0, F>:AF (Autómata Finito), con K={q_0, . . ,q_n},
Un conjunto de predicados $\{P_i: \sum^* \rightarrow Bool / 0 \leq i \leq n\}$es un conjunto de invariantes de estado fuerte para M si

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

Propiedad 2

Si I = {Pi} es un cief para M<K,∑, δ,q_0, F>:AF entonces $L(M) = \{\alpha / V_{qi\in F}\ P_i(\alpha)\}$

Demostración

$\subseteq$) $\alpha \in L(M) \Rightarrow \exists\, q_f \in F / (q_0,\alpha) \succ^* (q_f,\lambda) \Rightarrow P_f(\alpha) \Rightarrow V_{qi\in F}\ Pi(\alpha) \Rightarrow$

$$\alpha \in \{\alpha / V_{qi\in F}\ Pi(\alpha)\}$$

$\supseteq$) sea $\alpha / V_{qi\in F}\ Pi(\alpha) \Rightarrow \exists\, P_f / P_f(\alpha) \wedge q_f \in F \Rightarrow (q0,\alpha) \succ^* (q_f,\lambda) \Rightarrow \alpha \in L(M)$

Propiedad 3

Dado M=<K, $\sum$, δ, q_0, F>: AF
Si I={Pi} cumple
$\forall i,j, \forall \alpha \in \sum^*$

$$\begin{array}{lll} (& (1: & P_0(\lambda) \wedge (P_j(\lambda) \Rightarrow (q_0,\lambda) \succ^* (q_j,\lambda)) \quad) \\ & (2: \forall a \in \sum \cup \{\lambda\} & ((q_i,a) \succ^* (q_j,\lambda) \Rightarrow ((P_i(\alpha) \Rightarrow P_j(\alpha a)) \ ) \ ) \wedge \\ & (3: \forall a \in \sum & (P_i(\alpha a) \Rightarrow \exists\, h : (P_h(\alpha) \wedge (q_h, a) \succ^* (q_j,\lambda)) \ ) \ ) \quad) \end{array}$$

entonces I es un *cief* para M

Demostración
Hay que ver que $\forall i\ \forall \alpha\ ((q_o,\alpha) \succ^* (q_i,\lambda) \Leftrightarrow p_i(\alpha))$

$\Rightarrow$) $(q_o,\alpha) \succ^* (q_i,\lambda)$ sii $\exists\, k$ / $(q_o,\alpha) \succ^k (q_i,\lambda)$
se demostrará la propiedad por inducción en k

base k=0
$(q_o,\alpha) \succ^0 (q_i,\lambda) \Rightarrow q_i=q_0 \wedge \alpha = \lambda$ y por (1) $P_0(\lambda)$ que es $P_i(\alpha)$

inducción: k= h+1
$(qo,\alpha a) \succ^{h+1} (qi,\lambda)$ con $a \in \sum \cup \{\lambda\} \Rightarrow$

$\exists\, qj / (q_o,\alpha a) \succ^h (q_j,a) \succ (q_i,\lambda)$

como por hipótesis inductiva $(q_o,\alpha) \succ^k (q_j,\lambda) \Rightarrow P_j(\alpha)$

y por (2) $(q_j,a) \succ^* (q_j,\lambda) \Rightarrow (P_j(\alpha) \Rightarrow Pi(\alpha a))$ resulta

$Pi(\alpha a)$. ▮

$\Leftarrow$) Ahora se demostrará que $Pi(\alpha) \Rightarrow (q_0,\alpha) \succ^* (q_i,\lambda)$
se hará por inducción en $|\alpha|$

Base $\alpha=\lambda$

Se cumple $P_i(\alpha) = P_i(\lambda)$ entonces por (1)
$(q_0, \lambda) \succ^* (q_i, \lambda)$ ▮

Inducción
Se verá que, supuesto que se cumpla la propiedad para α, también se cumple para αa
Se supone $P_i(\alpha a)$

Entonces por (3) $\exists$ h : $P_h(\alpha) \wedge (q_h,a) \succ^* (q_i, \lambda)$ además por hipótesis inductiva
$(q_0, \alpha) \succ^* (q_h, ,\lambda)$ luego
$(q_0, \alpha a) \succ^* (q_h, ,a) \wedge (q_h,a) \succ^* (q_i, \lambda) \Rightarrow (q_0, \alpha a) \succ^* (q_h, ,\lambda)$ ∎

∎

7.- Conclusiones.

El método brinda una manera sistemática de abordar el diseño de un autómata, resultando conveniente su utilización en cualquier contexto donde se requiera la construcción de estos formalismos, ya sea para obtener algoritmos eficientes en aplicaciones prácticas como por necesidades teóricas. Los autómatas construidos siguiéndolo son correctos por construcción. En muchos casos, sirve de guía para encontrar soluciones que no resultan evidentes. Su uso en la educación universitaria puede servir para afianzar la práctica de diseño formal, dando continuidad a la metodología incorporada en los primeros cursos de programación, basada en los trabajos de Dijkstra [6],[7] y desarrollada por Gries,[8].

8.- 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."lex & yacc" J. Levine, T. Mason & D Brown. O'Reilly & Associates 1992

3. "LEX a lexical analyzer generator, Lesk , CSTR 93, Bell Laboratories, 1975

4. Thompson K. , Regular expression search algorithm, Communications ACM 11:6 1968

5. "A Discipline of Programming". Dijkstra. Prentice Hall. 1978

6. "Formal Develop of Programs". Dijkstra. Addison Wesley

7. "The science of Programming". D. Gries. Springer Verlang 1981