Teoría de autómatas para investigadores en XML

Rafael C. Carrasco Jiménez
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante
Febrero 2006

4 de febrero de 2008

Autómatas finitos de cadenas

Un DFA (deterministic finite-state automaton) es una representación (grafo) de un procedimiento computable que requiere memoria finita.

Autómatas finitos de cadenas

Un DFA (deterministic finite-state automaton) es una representación (grafo) de un procedimiento computable que requiere memoria finita.
Ejemplo: determinar la paridad de una cadena binaria.
Contraejemplo: determinar si la entrada es palíndroma.

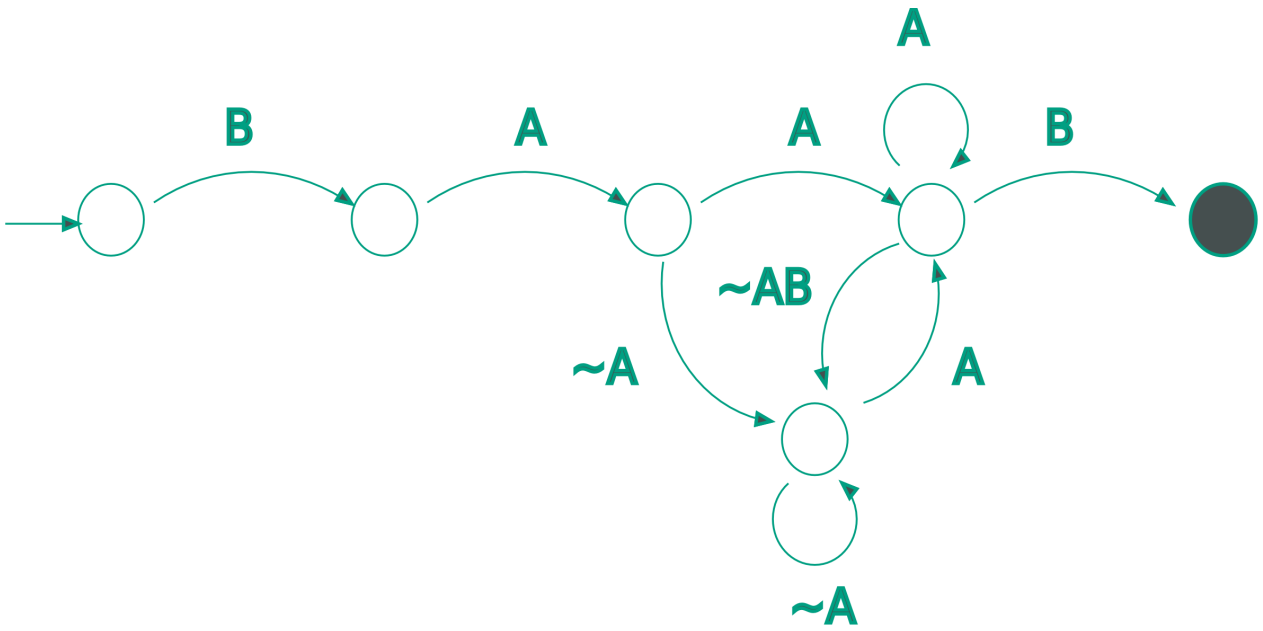
Expresiones regulares

Las expresiones regulares definen lenguajes usando símbolos, paréntesis y operadores de concatenación, elección y repetición.
Comentarios de C:

```
A    [*]
B    [/]
C    [^*/]
Comment {B}{A}{B}*({A}*{C}{B}*)*{A}*{A}{B}
```

Expresiones regulares

La validación de cadenas con respecto a expresiones regulares puede hacerse mediante DFA.



Autómata de Glushkov de una expresión regular

Para cada expresión regular r, se construye el marcado E_r sustituyendo los símbolos por posiciones.
Por ejemplo
$r = BAB^*(A^*CB^*)A^*AB \Rightarrow E_r = 123^*(4^*56^*)^*7^*89$.
Cada posición será un estado del autómata de Glushkov.
Para construir las transiciones se usan 4 funciones: empty, first, last, follow.

Autómata de Glushkov de una expresión regular

empty(E) es cierto si la subexpresión contiene la cadena vacía:

$$\begin{aligned}
\text{empty}(n) &= \text{FALSE} \\
\text{empty}(F|G) &= \text{empty}(F) \vee \text{empty}(G) \\
\text{empty}(F,G) &= \text{empty}(F) \wedge \text{empty}(G) \\
\text{empty}(F*) &= \text{TRUE} \\
\text{empty}(F+) &= \text{empty}(F) \\
\text{empty}(F?) &= \text{TRUE}
\end{aligned}$$

Autómata de Glushkov de una expresión regular

first(E) es el conjunto de símbolos por los que puede empezar una cadena de E:

$$
\begin{aligned}
\text{first}(n) &= \{n\} \\
\text{first}(F|G) &= \text{first}(F) \cup \text{first}(G) \\
\text{first}(F,G) &= \begin{cases} \text{first}(F) \cup \text{first}(G) & \text{if empty}(F) \\ \text{first}(F) & \text{otherwise} \end{cases} \\
\text{first}(F*) &= \text{first}(F) \\
\text{first}(F+) &= \text{first}(F) \\
\text{first}(F?) &= \text{first}(F)
\end{aligned}
$$

Autómata de Glushkov de una expresión regular

last(E) es el conjunto de símbolos por los que puede terminar una cadena de E:

$$
\begin{aligned}
\text{last}(n) &= \{n\} \\
\text{last}(F|G) &= \text{last}(F) \cup \text{last}(G) \\
\text{last}(F,G) &= \begin{cases} \text{last}(F) \cup \text{last}(G) & \text{if empty}(G) \\ \text{last}(G) & \text{otherwise} \end{cases} \\
\text{last}(F*) &= \text{last}(F) \\
\text{last}(F+) &= \text{last}(F) \\
\text{last}(F?) &= \text{last}(F)
\end{aligned}
$$

Autómata de Glushkov de una expresión regular

follow(E) es el conjunto de pares de símbolos que pueden aparecer consecutivos en E:

$$\begin{aligned}
\text{follow}(n) &= \emptyset \\
\text{follow}(F|G) &= \text{follow}(F) \cup \text{follow}(G) \\
\text{follow}(F,G) &= \text{follow}(F) \cup \text{follow}(G) \cup \text{last}(F) \times \text{first}(G) \\
\text{follow}(F*) &= \text{follow}(F) \cup \text{last}(F) \times \text{first}(F) \\
\text{follow}(F+) &= \text{follow}(F) \cup \text{last}(F) \times \text{first}(F) \\
\text{follow}(F?) &= \text{follow}(F)
\end{aligned}$$

Autómata de Glushkov de una expresión regular

El autómata de Glushkov es $(\mathbb{N}, \Sigma, \delta, 0, F)$, con:

- $Q = \{0, 1, ..., N\}$
- $\delta(0, a) = \{n \in \text{first}(E_r) : \Phi_r(n) = a\}$
- $\delta(n, a) = \{m \in Q : (n, m) \in \text{follow}(E_r) \wedge \Phi_r(m) = a\}$
- $F = \begin{cases} \{0\} \cup \text{last}(E_r) & \text{if empty}(E_r) \\ \text{last}(E_r) & \text{otherwise} \end{cases}$

siendo N el número de símbolos de r y Φ el homomorfismo que genera r a partir de E_r.

Autómata de Glushkov de una expresión regular

Construye el autómata de Glushkov para $BAB^*(A^*CB^*)A^*AB$

Autómata de Glushkov de una expresión regular

Si el autómata de Glushkov es determinista, r es 1-inambigua y, por tanto, válida en el estándar SGML.

Aunque todo autómata finito tienen un equivalente determinista, no todas las lenguajes regulares admiten una expresión regular con autómata de Glushkov determinista.

Medidas probabilísticas

En un autómata probabilístico, cada transición (y cada estado de aceptación) tiene una probabilidad asociada.
Algunas distancias

- Cuadrática: $\sum_x (p_A x) - p_B(x))^2$.
- Kullback-Leibler: $\sum_x p_A(x) * \log \frac{p_A(x)}{p_B(x)}$.

La distancia cuadrática es más suave, pero menos sensible a los valores pequeños.

Medidas probabilísticas

La probabilidad de coemisión $C(A,B) = \sum_x p_A(x)p_B(x)$ permite calcular la distancia cuadrática:

$$d_2(A,B) = C(A,A) + C(B,B) - 2C(A,B)$$

Medidas probabilísticas

$$C(A, A') = \sum_{a} \sum_{i \in Q} \sum_{j \in Q'} c_{ij}\ p(i, a) p(j, a)$$

Los coeficientes c_{ij} son el número esperado de "pasos" por i y j.

$$c_{ij} = (i == 0)(j == 0) + \sum_{a} \sum_{k:\delta(k,a)=i} \sum_{l:\delta(l,a)=j} c_{kl} p(k, a) p(l, a)$$

Demostración en: Carrasco 1997.

Árboles

Dado un *alfabeto* $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$:

- Todos los símbolos de Σ son árboles de T_Σ.
- Dado $\sigma \in \Sigma$ y $m > 0$ árboles $t_1, \ldots, t_m$, $\sigma(t_1 \cdots t_m)$ es un árbol de T_Σ.

Lenguajes de árboles

A cualquier subconjunto de árboles se le llama *lenguaje*. En particular, el lenguaje $\mathrm{sub}(t)$ de *subárboles* de t es

$$\mathrm{sub}(t) = \begin{cases} \{\sigma\} & \text{if } t = \sigma \in \Sigma \\ \{t\} \cup \bigcup_{k=1}^{m} \mathrm{sub}(t_k) & \text{if } t = \sigma(t_1 \dots t_m) \in T_\Sigma - \Sigma \end{cases}$$

XHTML es un lenguaje de árboles sobre el alfabeto: $\{$html, head, body, p, a, ul, ol, li, th, tr, td...$\}$

Autómatas de árboles

Un *autómata finito de árboles* es $A = (Q, \Sigma, \Delta, F)$,

- $Q = \{q_1, \ldots, q_{|Q|}\}$ es un conjunto*estados*;
- $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ es el *alfabeto*;
- $F \subseteq Q$ es un subconjunto de *estados de aceptación*,
- $\Delta \subset \cup_{m=0}^{\infty} \Sigma \times Q^{m+1}$ es un conjunto finito de *transiciones*.

Autómatas de árboles

Los autómatas de árboles pueden ser

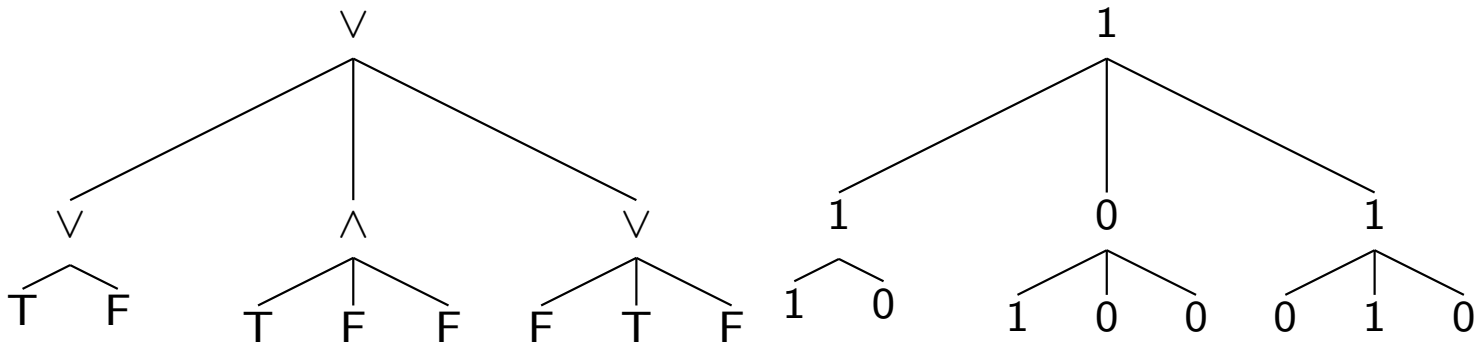
- indeterministas :-|
- deterministas ascendentes :-)
- deterministas descendente :-(

Debemos valorar capacidad expresiva y complejidad de análisis.

Autómatas de árboles

Evaluador de expresiones lógicas:

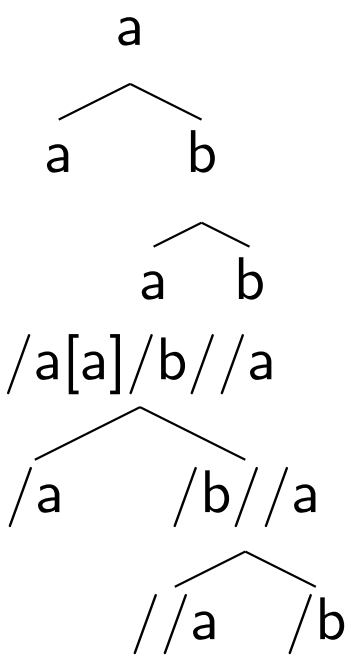
$$\begin{aligned}\Delta \quad = \quad & \{(F,0),(T,1),(\wedge,1^+,1),(\wedge,(0|1)^*0(0|1)^*,0) \\ & (\vee,0^+,0),(\vee,(0|1)^*1(0|1)^*,1)\}\end{aligned}$$



Autómatas de árboles

Evaluador de XPath `/a[a]/b//a` (indeterminista).

$$\Delta = \{ \ (\texttt{a}, Q^*, \texttt{/a}), \ (\texttt{b}, Q^*, \texttt{/b}), \ (\texttt{a}, \ Q^*, \texttt{//a}), \ (\texttt{b}, Q^*\texttt{//a}Q^*, \texttt{/b//a}), \ (\texttt{a}, Q^*\texttt{/a}Q^*\texttt{/b//a}Q^*, \texttt{/a[a]/b//a}), \ldots \ \}$$



Autómatas ascendentes árboles

Las siguientes afirmaciones sobre un lenguaje de árboles L son equivalentes:

- L es reconocible por un autómata de árboles ascendente indeterminista.
- L es reconocible por un autómata de árboles ascendente determinista.
- L es reconocible por un autómata de árboles descendente indeterminista.

En cambio, los autómatas deterministas descendentes (como quiera que se definan) son menos potentes.

Autómatas ascendentes de árboles

Cada transición $(\sigma, i_1, ..., i_m, q)$ de Δ tiene *argumento* $(\sigma, i_1, ..., i_m)$ y *salida* q. El autómata es determinista si no hay más de una salida por cada argumento:

$$\delta_m(\sigma, i_1, ..., i_m) = \begin{cases} q & \text{if } q \in Q \text{ such that } (\sigma, i_1, ..., i_m, q) \in \Delta \\ \perp & \text{if no such } q \text{ exists} \end{cases}$$

$\perp$ es el *estado de absorción*

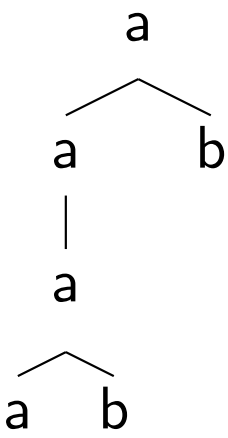
Autómatas ascendentes de árboles

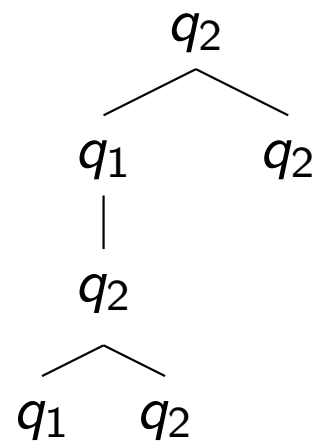
El resultado de A en t es $A(t)$:

$$A(t) = \begin{cases} \delta_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ \delta_m(\sigma, A(t_1), \ldots, A(t_m)) & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma \end{cases}$$

Autómatas ascendentes de árboles

Si $\delta_0(a) = q_1$, $\delta_0(b) = q_2$, $\delta_2(a, q_1, q_2) = q_2$ y $\delta_1(a, q_2) = q_1$,





Autómatas ascendentes de árboles

El lenguaje $L_A(q)$ aceptado por $q \in Q$ es

$$L_A(q) = \{t \in T_\Sigma : A(t) = q\}$$

y el lenguaje $L(A)$ aceptado por A es

$$L(A) = \bigcup_{q \in F} L_A(q).$$

Tipos de autómatas de árboles para XML

Taxonomía (Murata 2001):

- Locales (DTD)
- De tipo único (XML Schema)
- Generales (Relax NG)

Sólo los dos primeros garantizan interpretación única (identificación del estado asociado a la etqiueta) incluso con procesamisnto top-down (por ejemplo, con SAX).

Minimización de autómatas ascendentes de árboles

Eliminación de estados inaccesibles: $L_A(q) = \emptyset$.

- $I \leftarrow Q$
- Mientras existen $q \in I$, $m \geq 0$, $\sigma \in \Sigma$ y $(i_1, ..., i_m) \in (Q - I)^m$ tales que $\delta_m(\sigma, i_1, ..., i_m) = q$, elimínese q de I.

Minimización de autómatas ascendentes de árboles

Dos estados i y j son equivalentes si

1. $i \in F$ y $j \notin F$ o viceversa.
2. Existen $m > 0$, $k \leq m$ y $(\sigma, r_1, ..., r_m) \in \Sigma \times Q^m$ tales que

$$\delta_m(\sigma, r_1, \ldots, r_{k-1}, i, r_{k+1} \ldots, r_m) \not\equiv \delta_m(\sigma, r_1, \ldots, r_{k-1}, j, r_{k+1} \ldots$$

Minimización de autómatas ascendentes de árboles

Sea P_τ la partición de Q en la iteración τ y $E_\tau[i]$ la clase de P_τ que contiene a i.
$i \not\equiv_\tau j$ si existe $m > 0$, $k \leq m$ y $(\sigma, r_1, ..., r_m) \in \Sigma \times Q^m$ tales que

$$E_\tau[\delta_m(\sigma, r_1, \ldots, r_{k-1}, i, r_{k+1} \ldots, r_m)] \neq E_\tau[\delta_m(\sigma, r_1, \ldots, r_{k-1}, j, r_{k+}$$

Minimización de autómatas ascendentes de árboles

Algorithm minimizeDTA
Input: a DTA $A = (Q, \Sigma, \Delta, F)$ *with no inaccessible states.*
Output: a minimal DTA $A^{\text{mín}} = (Q^{\text{mín}}, \Sigma, \Delta^{\text{mín}}, F^{\text{mín}})$.
Method:

1. *Create the initial partition* $P_0 = (F, Q - F)$ *and make* $\tau \leftarrow 0$.
2. *While there exist* $i, j \in Q$ *such that* $E_\tau[i] = E_\tau[j]$ *and* $i \not\equiv_\tau j$
   - *Build the subset* $\mathcal{N} = \{k \in E_\tau[i] : k \equiv_\tau i\}$.
   - *Create* $P_{\tau+1}$ *from* P_τ *by splitting class* $E_\tau[i]$ *into* $\mathcal{N}$ *and* $E_\tau[i] - \mathcal{N}$.
   - *Make* $\tau \leftarrow \tau + 1$.
3. *Output* $(Q^{\text{mín}}, \Sigma, \Delta^{\text{mín}}, F^{\text{mín}})$ *with*
   - $Q^{\text{mín}} = \{E_\tau[i] : i \in Q\}$;
   - $F^{\text{mín}} = \{E_\tau[i] : i \in F\}$;
   - $\delta_m^{\text{mín}}(\sigma, E_\tau[i_1], ..., E_\tau[i_m]) = E_\tau[\delta_m(\sigma, i_1, ..., i_m)]$ *for all* $m \geq 0, \sigma \in \Sigma$, *and* $(i_1, ..., i_m) \in Q^m$.

Gramáticas regulares de árboles

Son equivalentes a los autómatas de árboles. $G = (N, T, S, P)$:

- Σ es un alfabeto de símbolos terminales;
- N es un conjunto finito de variables;
- S es el símbolo inicial;
- P es un conjunto de reglas del tipo $X \longrightarrow ar$, con $X \in N$, $a \in T$ y r una expresión regular sobre N (content model).