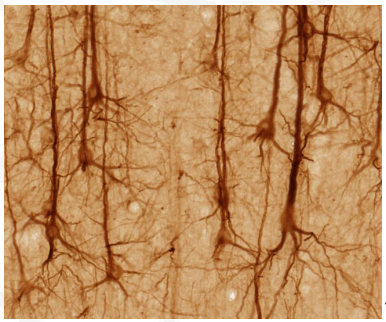
# The Inclusion Problem for Regular Expressions

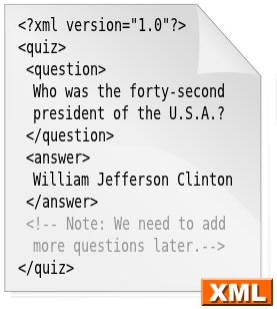
Dag Hovland

Dept. of Informatics, University of Oslo, Norway
hovland@ifi.uio.no

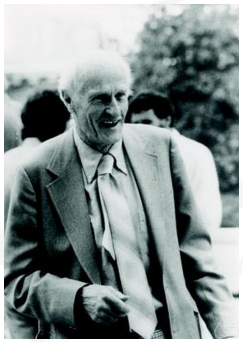
November 10, 2011

# From Neurons to XML





# From Neurons to Automata



- 1880s-1900s: The Neuron as basic unit (Ramòn y Cajal, Golgi)
- 1943: McCulloch and Pitts, "A logical calculus of ideas immanent in nervous activity"
- 1951/56: S.C. Kleene, "Representation of Events in Nerve Nets and Finite Automata"
- 1950s-70s: Tapes, words, events and languages

# From Automata to XML

- 1970s: Ken Thompson: qed with reg.exp support (1971), grep (1973)
- 1980s-90s: Perl and SGML
- XML (1998), XML Schema (2001)

# (Non-empty) Regular Expressions

Alphabet: $\Sigma$

Regular Expressions:

$R_\Sigma ::= R_\Sigma + R_\Sigma \mid R_\Sigma \cdot R_\Sigma \mid R_\Sigma^* \mid \Sigma \mid \epsilon$

$r, r_1, r_2, \ldots$ variables for regular expressions

$\mathrm{sym}(r) \subseteq \Sigma$ is the set of letters occurring in $r$.

# Regular Languages

$\|r\| \subseteq \Sigma^*$, denoted by $r$, a regular language.
$\epsilon$: the empty word

# Regular Languages

$\|r\| \subseteq \Sigma^*$, denoted by $r$, a regular language.

$\epsilon$: the empty word

$r \in \Sigma \cup \{\epsilon\} \Rightarrow \|r\| = \{r\}$

$\|r_1\| + \|r_2\| = \|r_1\| \cup \|r_2\|$

$\|r_1 \cdot r_2\| = \|r_1\| \cdot \|r_2\|$

(where $\|r_1\| \cdot \|r_2\| = \{w_1 \cdot w_2 \mid w_1 \in \|r_1\| \wedge w_2 \in \|r_2\|\}$ and $\epsilon \cdot w = w \cdot \epsilon = w$).

$\|r^*\| = \bigcup_{0 \leq i} \|r\|^i$

(where $A^0 = \{\epsilon\}$ and for $i > 0$, $A^i = A \cdot A^{i-1}$)

# Example Regular Expressions

- $\|ac^*b\| = \{ab, acb, accb, acccb, \ldots\}$.
- $\|(a+c)^*b\| = \{b, ab, cb, aab, cab, acb, ccb \ldots\}$.

# Nullable Expressions

Nullable Expressions (Glushkov (1961), McNaughton & Yamada (1960))

$$\mathfrak{N} ::= \mathfrak{N} + R_\Sigma \mid R_\Sigma + \mathfrak{N} \mid \mathfrak{N} \cdot \mathfrak{N} \mid R_\Sigma^* \mid \epsilon$$

$$\epsilon \in \|r\| \Leftrightarrow r \in \mathfrak{N}$$

# The First-set

The First-set (Glushkov (1961), McNaughton & Yamada (1960)) of a regular expression is the set of letters that can occur first in a word in the language, that is, $\text{first}(r) = \{l \in \Sigma \mid \exists w : lw \in \|r\|\}$.

$$\frac{\text{first}(\epsilon) = \varnothing, \qquad r \in \Sigma \Rightarrow \text{first}(r) = \{r\}}{r = r_1 + r_2 \Rightarrow \text{first}(r) = \text{first}(r_1) \cup \text{first}(r_2)}$$

$$\frac{r = r_1 \cdot r_2 \wedge r_1 \in \mathfrak{N} \Rightarrow \text{first}(r) = \text{first}(r_1) \cup \text{first}(r_2)}{r = r_1 \cdot r_2 \wedge r_1 \notin \mathfrak{N} \Rightarrow \text{first}(r) = \text{first}(r_1)}$$

$$\frac{}{r = r_1^* \Rightarrow \text{first}(r) = \text{first}(r_1)}$$

# Language Inclusion

- INCLUSION $= \{(r_1, r_2) \mid \|r_1\| \subseteq \|r_2\|\}$
- E.g. $\|ac^*b\| \subseteq \|(a+c)^*b\|$.
- PSPACE-complete, [Stockmeyer and Meyer, 1972].
- Note that $\|r_1\| \subseteq \|r_2\|$ iff $\|r_1\| \cap \overline{\|r_2\|} = \varnothing$.
- Usual algorithm creates an NFA recognizing $\|r_1\| \cap \overline{\|r_2\|}$ and checks reachability of final states from initial state
- Super-polynomial blowup in creation of DFA recognizing $\|r_2\|$

# 1-Unambiguity

- Some applications, e.g., SGML and XML Schema: Restrict scope to *1-unambiguous* regular expressions.
- Construction of DFAs from 1-unambiguous regular expressions is polynomial-time, so INCLUSION moves to P.

# 1-unambiguous regular expressions

- From a 1-unambiguous regular expression we can in polynomial time construct a DFA recognizing the same language, and where every non-initial state corresponds to a letter occurrence in the expression (Brüggemann-Klein and Wood (1993))
- 1-unambiguous: $b^*a(b^*a)^*$.
- 1-ambiguous: $(a+b)^*a$ and $(a+\epsilon)a$.

# Problems

- Hard to extend the algorithm to more operators.
- All parts (also superfluous ones) of the right-hand expression must be treated. For example:
  $\|ab\| \subseteq \|(a + (b + c)^* c(b + c) \cdots (b + c))b\|$

# Derivatives

- Derivatives of regular expressions (Brzozowski (1964)) lend themselves to another algorithm for inclusion, described first by Brzozowski in 1967, and rediscovered by Antimirov in 1995 and applied to 1-unambiguous expressions by Chen & Chen in 2008.
- $a\backslash(a+c)b = b$, $a\backslash a* = a*$.
- $\bigcup_{l \in \mathsf{first}(r)}(\{l\} \cdot \|l\backslash r\|) = \|r\|$.
- Polynomial run-time when *both* expressions are 1-unambiguous
- Unneeded parts of the right-hand expression are discarded

# Goal

Goal: Combine the best from both approaches:

- Takes two arbitrary expressions $r_1$ and $r_2$ as input
- Gives output "Yes", "No" or "1-ambiguous" in time polynomial in the size of the input
- If output is "Yes", then $\|r_1\| \subseteq \|r_2\|$, if output is "No" then $\|r_1\| \not\subseteq \|r_2\|$
- If output is "1-ambiguous", then $r_2$ is not 1-unambiguous
- Discards unneeded parts of $r_2$

# Header Normal Form

Using associativity of concatenation, and that $\epsilon$ is the neutral element for concatenation, all regular expressions can in linear time be translated to one of the forms:

- $l \cdot r$
- $(r_1 + r_2) \cdot r_3$
- $r_1^* \cdot r_2$
- $\epsilon$

# About the Rules

- Binary relation $\sqsubseteq$ over regular expressions
- We say that $r_1 \sqsubseteq r_2$ *holds* iff $\|r_1\| \subseteq \|r_2\|$.
- Rules of the form $\frac{over}{under}[side]$:
  *over* is 0, 1 or 2 premises, while *under* is the conclusion.
- Intended semantics: Assuming *side* holds for exactly one rule instance, the conjunction of the premises in *over* hold if and only if *under* holds.

# Decision Rules

$$\text{(Axm)}\quad \frac{}{\epsilon \sqsubseteq r}\ [r \in \mathfrak{N}] \qquad \text{(Letter)}\quad \frac{r_1 \sqsubseteq r_2}{l \cdot r_1 \sqsubseteq l \cdot r_2}$$

# More About the Rules

- Used bottom-up, depth-first, with a store $S$ of already treated decisions.
- If several rule instances fit, return "1-ambiguous"
- If no rule instance fits, return "No"
- Otherwise, return "Yes" when search is finished

# Decision Rules cont'd

$$\text{(LetterStar)}\quad \frac{l \cdot r_1 \sqsubseteq r_2 r_2^* r_3}{l \cdot r_1 \sqsubseteq r_2^* r_3}\ [l \in \text{first}(r_2)]$$

$$\text{(LetterChoice)}\quad \frac{l \cdot r_1 \sqsubseteq r_i r_4}{l \cdot r_1 \sqsubseteq (r_2 + r_3) r_4}\ \left[\begin{array}{c} i \in \{2,3\} \\ l \in \text{first}(r_i) \end{array}\right]$$

$$\text{(ElimCat)}\quad \frac{r_1 \sqsubseteq r_3}{r_1 \sqsubseteq r_2 r_3}\ \left[\begin{array}{c} \exists l, r_4, r_5 : r_1 = l \cdot r_4 \vee r_1 = r_4^* r_5 \\ r_2 \in \mathfrak{N} \\ \text{first}(r_1) \subseteq \text{first}(r_3) \end{array}\right]$$

# Example

$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{Axm})}{(\text{Letter})\ \ \epsilon \sqsubseteq \epsilon}}{(\text{Letter})\ \ b \sqsubseteq b}}{(\text{LetterChoice})\ \ ab \sqsubseteq ab}}{ab \sqsubseteq (a + (b + c)^* c(b + c) \cdots (b + c))b}$$

$$\text{(LeftChoice)}\ \frac{r_1 r_3 \sqsubseteq r_4 \qquad r_2 r_3 \sqsubseteq r_4}{(r_1 + r_2) r_3 \sqsubseteq r_4}$$

# Decision Rules cont'd

$$\text{(LeftStar)}\quad \frac{r_1 r_1^* r_2 \sqsubseteq r_3 r_4 \qquad r_2 \sqsubseteq r_3 r_4}{r_1^* r_2 \sqsubseteq r_3 r_4} \quad \left[\begin{array}{c} \mathsf{first}(r_1) \cap \mathsf{first}(r_3) \neq \varnothing \\ r_4 \neq \epsilon \vee r_2 \neq \epsilon \\ \exists l, r_5 : r_3 = l \vee r_3 = r_5^* \end{array}\right]$$

$$\text{(StarStarE)}\quad \frac{r_1 \sqsubseteq r_2^*}{r_1^* \sqsubseteq r_2^*}$$

# "No" Example

$$\dfrac{\dfrac{\dfrac{\dfrac{\textit{Fail}}{(\text{Letter})\ \ \epsilon \sqsubseteq a(aa)^*}}{(\text{LetterStar})\ \ a \sqsubseteq aa(aa)^*}}{(\text{StarStarE})\ \ a \sqsubseteq (aa)^*}}{a^* \sqsubseteq (aa)^*}$$

# Basic Properties of the Rules

- Preservation of 1-Unambiguity of *right-hand* expression.
- Several rule instances possible only if right-hand expression 1-ambiguous.
- If only one rule conclusion matches a pair, then the conclusion holds if and only if the conjunction of the premises hold
- A rule instance fitting any $r_1 \sqsubseteq r_2$ where $\|r_1\| \subseteq \|r_2\|$

# Store Example

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{(Letter)}\ \ a^*b^* \sqsubseteq (a+b)^*}\,\textit{Store}}{\text{(LetterChoice)}\ \ aa^*b^* \sqsubseteq a(a+b)^*}}{\text{(LetterStar)}\ \ aa^*b^* \sqsubseteq (a+b)(a+b)^*} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\text{(Letter)}\ \ \epsilon \sqsubseteq (a+b)^*}\,\text{(Axm)}}{\text{(LetterChoice)}\ \ b \sqsubseteq b(a+b)^*}}{\text{(LetterStar)}\ \ b \sqsubseteq (a+b)(a+b)^*}}{\text{(StarStarE)}\ \ b \sqsubseteq (a+b)^*}}{\text{(LeftStar)}\ \ aa^*b^* \sqsubseteq (a+b)^*} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\text{(Letter)}\ \ \epsilon \sqsubseteq (a+b)^*}\,\text{(Axm)}}{\text{(LetterChoice)}\ \ b \sqsubseteq b(a+b)^*}}{\text{(LetterStar)}\ \ b \sqsubseteq (a+b)(a+b)^*}}{\text{(StarStarE)}\ \ b \sqsubseteq (a+b)^*}}{b^* \sqsubseteq (a+b)^*}}{a^*b^* \sqsubseteq (a+b)^*}$$

# Safe Loop Checking

- By induction on the length of a word $w$, for all $r_1 \sqsubseteq r_2$ occurring in the execution, $w \in \|r_1\|$ implies $w \in \|r_2\|$.
- For all $l \cdot w \in \|r_1\|$, follow the upward path to an instance of (Letter).
- Use the induction hypothesis on the premise of the (Letter) rule.
- Follow the path down again.

# Conclusion

- A sound algorithm for inclusion of regular expressions
- Run-time always polynomial
- Always returns yes or no when the right-hand expression is 1-Unambiguous
- When the right-hand expression is 1-ambiguous the algorithm might decide that this is a problem
- Not needed parts of the right-hand expression are not treated

Valentin M. Antimirov.

Rewriting regular inequalities (extended abstract).

In Horst Reichel, editor, *FCT*, volume 965 of *Lecture Notes in Computer Science*, pages 116–125. Springer, 1995.

Janusz A. Brzozowski.

Derivatives of regular expressions.

*J. ACM*, 11(4):481–494, 1964.

Janusz A. Brzozowski.

Roots of star events.

*J. ACM*, 14(3):466–477, 1967.

Haiming Chen and Lei Chen.

Inclusion test algorithms for one-unambiguous regular expressions.

In John S. Fitzgerald, Anne Elisabeth Haxthausen, and Hüsnü Yenigün, editors, *ICTAC*, volume 5160 of *LNCS*, pages 96–110. Springer, 2008.


From xkcd.com

# Grep-Style Regular Expressions

- A killer has followed your friend on vacation. The address is somewhere in her thousands of e-mails.
- (A—a)dresse.*[0-9]{4} [A-ZÆ-Å][a-zæ-å]* (Adresse . . . 5081 Bergen, adresse . . . 7070 Bosberg, . . . )
- 385 matches. $4,2$ GB mail. about 46 sec.

**Input**: Two regular expressions $r_1$ and $r_2$
Initialize stack T and set S both consisting of pairs of regular expressions
push $(r_1, r_2)$ on T
**while** T *not empty* **do**
  pop $(r_1, r_2)$ from T
  **if** $(r_1, r_2) \notin$ S **then**
    **if** first$(r_1) \not\subseteq$ first$(r_2)$ *or* $r_1 \in \mathfrak{N} \wedge r_2 \notin \mathfrak{N}$ *or* $r_2 = \epsilon \wedge r_1 \neq \epsilon$ **then**
      **return** *"No"*
    **end**
    **if** $r_1 \sqsubseteq r_2$ *matches conclusion of more than one rule instance* **then**
      **return** *"1-ambiguous"*
    **end**
    add $(r_1, r_2)$ to S
    **for** *all premises* $r_3 \sqsubseteq r_4$ *of the rule instance matching* $r_1 \sqsubseteq r_2$
    **do**
      push $(\text{hdf}(r_3), \text{hdf}(r_4))$ on T
    **end**
  **end**
**end**
**return** *"Yes"*