

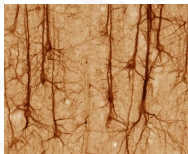
The Membership Problem for Regular Expressions with Unordered Concatenation and Numerical Constraints

Dag Hovland

University of Oslo, Norway

March, 2012

From Neurons to XML



```
<?xml version="1.0"?>
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

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)



From xkcd.com

Numerical Constraints

- ▶ Expresses iterations (as the $*$) but with more control: bounds the number of iterations
- ▶ $abbac \in \|((a + b)^{1..40}c)\|$
- ▶ Standard in text-search regular expressions like grep, perl
- ▶ Used in Extensible Markup Language (XML) Schema.

Unordered Concatenation

- ▶ $\&(ya, basta)$ matches both *yabasta* and *bastaya*
- ▶ Defined in terms of permutations (reorderings)
- ▶ A word w matches an unordered concatenation $\&(r_1, \dots, r_n)$ if:
 - ▶ $r'_1 \cdots r'_n$ is a permutation of $r_1 \cdots r_n$
 - ▶ and $w = w_1 \cdots w_n$ where w_1 matches r'_1 , w_2 matches r_2 etc.
- ▶ $\{abc, acb, bca, cba\} = \|\&(a, \&(b, c))\|$

Unordered Concatenation

- ▶ $\&(ya, basta)$ matches both *yabasta* and *bastaya*
- ▶ Defined in terms of permutations (reorderings)
- ▶ A word w matches an unordered concatenation $\&(r_1, \dots, r_n)$ if:
 - ▶ $r'_1 \cdots r'_n$ is a permutation of $r_1 \cdots r_n$
 - ▶ and $w = w_1 \cdots w_n$ where w_1 matches r'_1 , w_2 matches r_2 etc.
- ▶ $\{abc, acb, bca, cba\} = \|\&(a, \&(b, c))\|$
- ▶ Not associative
- ▶ $\{abc, bac, cab, cba\} = \|\&(\&(a, b), c)\|$
- ▶ First used in Standard Generalized Markup Language (SGML), later in a limited form in XML Schema

Regular Expressions with Numerical Constraints and Unordered Concatenation

$$\mathbb{N} = \{1, 2, \dots\} \text{ and } \mathbb{N}_1 = \{2, 3, 4, \dots\} \cup \{\infty\}$$

$$R_\Sigma ::= R_\Sigma + R_\Sigma \mid R_\Sigma \cdot R_\Sigma \mid R_\Sigma^{\mathbb{N}..N_1} \mid \&(R_\Sigma, \dots, R_\Sigma) \mid \Sigma \mid \epsilon$$

$$(r^* = r^{1..∞} + \epsilon.)$$

The language of a regular expression with numerical constraints and unordered concatenation

- ▶ $\|r_1 + r_2\| = \|r_1\| \cup \|r_2\|$
- ▶ $\|r_1 \cdot r_2\| = \|r_1\| \cdot \|r_2\|$
- ▶ $\|\&(r_1, \dots, r_n)\| = \bigcup_{\sigma \in \text{Perm}(\{1, \dots, n\})} \|r_{\sigma_1}\| \cdot \dots \cdot \|r_{\sigma_n}\|$
- ▶ $\|r^{l..u}\| = \bigcup_{i=l}^u \|r\|^i$ (where $A^0 = \{\epsilon\}$ and for $i > 0$, $A^i = A \cdot A^{i-1}$)
- ▶ $\|a\| = \{a\}$, when $a \in \Sigma \cup \{\epsilon\}$,

Numerical Constraints and Unordered Concatenation as Short-Hand

- ▶ $r^l : \underbrace{r \cdots r}_l$
- ▶ $r^{l..∞} : r^l \cdot r^*$
- ▶ $r^{l..u} : r^l (r + \epsilon)^{u-l}$, when $u \neq \infty$
- ▶ $\&(r_1, \dots, r_n) : r_1 \cdot \&(r_2, \dots, r_n) + \dots + r_n \cdot \&(r_1, \dots, r_{n-1})$

Complexity of Membership for the Usual Regular Expressions

- ▶ Polynomial time
- ▶ Streaming / Linear time in word: DFAs
- ▶ Constructing DFAs is “PSPACE-hard”

1-Unambiguity

- ▶ Many applications only work with a subset, the *1-unambiguous* regular expressions.
- ▶ Construction of DFAs from 1-unambiguous regular expressions is polynomial-time.

With Numerical Constraints

- ▶ Regular expressions with numerical constraints: P
- ▶ Linear in the length of the word: Strongly 1-unambiguous
- ▶ Regular expressions with unordered concatenation: NP-complete

Goal

- ▶ Find a subset of the regular expressions with numerical constraints and unordered concatenation, such that membership can be decided in polynomial time
- ▶ For fixed regular expression: runtime linear in the length of the word.

Finite Automata with Counters (FACs)

- ▶ Based on classical finite automata
- ▶ A counter for each subexpression with numerical constraints
- ▶ A counter for each of the arguments to the unordered concatenations
- ▶ Counter States: $\gamma : \mathcal{C} \rightarrow \{0, 1, 2, \dots\}$.
- ▶ min and max upper and lower limits for final value of each counter
- ▶ *update instruction* : $\psi : \mathcal{C} \rightarrow \{\text{inc, res, one}\}$
- ▶ inc for *increment*
- ▶ res for *reset* (to 0)
- ▶ one for *setting to 1*

An FAC is a Finite Automaton with Counters

An FAC is a tuple $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, \min, \max, q^I, \mathcal{F})$.

- ▶ Σ is the alphabet, Q is the *states*, and \mathcal{C} is the *counters*.
- ▶ $q^I \in Q$ is the *initial state*.
- ▶ $\mathcal{A} : Q - \{q^I\} \rightarrow \Sigma$: the letter matched when entering a non-initial state.
- ▶ Φ maps each state to a set of pairs of a state and an update instruction.

$$\Phi : Q \rightarrow \wp(Q \times (\mathcal{C} \rightarrow \{\text{inc, res, one}\})).$$

- ▶ \min and \max are the counter conditions.
- ▶ $\mathcal{F} \subset Q \times (\mathcal{C} \rightarrow \{\text{res}\})$: the *final configurations*.

Satisfaction of update instructions

$$\begin{array}{lcl}
 \gamma \models_{\min}^{\max} \emptyset & & \\
 \gamma \models_{\min}^{\max} \psi_1 \cup \{p \mapsto \text{inc}\} & \Leftrightarrow & \gamma \models_{\min}^{\max} \psi_1 \wedge \gamma(p) < \max(p) \\
 \gamma \models_{\min}^{\max} \psi_1 \cup \{p \mapsto \text{res}\} & \Leftrightarrow & \gamma \models_{\min}^{\max} \psi_1 \wedge \gamma(p) \geq \min(p) \\
 \gamma \models_{\min}^{\max} \psi_1 \cup \{p \mapsto \text{one}\} & \Leftrightarrow & \gamma \models_{\min}^{\max} \psi_1 \wedge \gamma(p) \geq \min(p)
 \end{array}$$

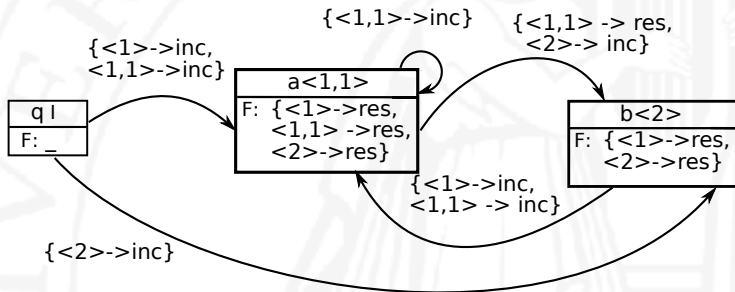
A counter can only be increased if the value is smaller than the maximum, while a value can only be reset if it's value is at least as large as the minimum.

Running an FAC

- ▶ A *configuration* of an FAC: a pair (q, γ)
- ▶ $q \in Q$
- ▶ $\gamma : \mathcal{C} \rightarrow \{0, 1, 2, \dots\}$
- ▶ δ : the transition function
- ▶ δ is calculated from Φ , \mathcal{A} , min, and max.
- ▶ Initial configuration: $(q^I, \{c \mapsto 0 \mid c \in \mathcal{C}\})$
- ▶ (q, γ) is a final configuration if $\gamma \models_{\min}^{\max} \psi$ for some $(q, \psi) \in \mathcal{F}$.
- ▶ If δ never returns more than 1 configuration, the FAC is deterministic

FAC recognizing $\| \&(a^{2..3}, b) \|$

C: $\langle 1 \rangle \langle 1, 1 \rangle \langle 2 \rangle$
 max: 1 3 1
 min: 1 2 1

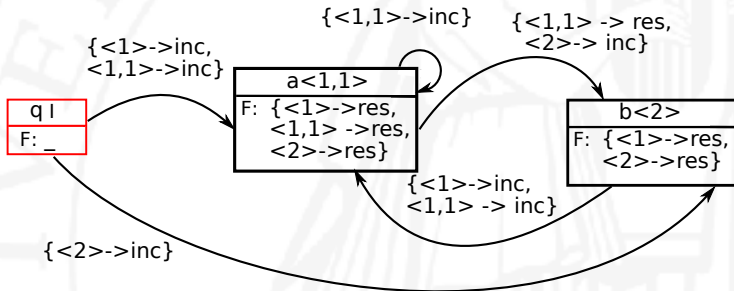


- ▶ qI initial
- ▶ The other states final

$baa \in \|\&(a^{2..3}, b)\| ?$

baa

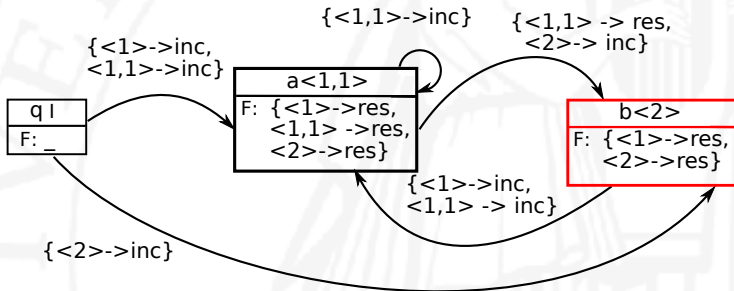
C: $\langle 1 \rangle \langle 1, 1 \rangle \langle 2 \rangle$
 max: 1 3 1
 min: 1 2 1
 value: 0 0 0



$baa \in \|\&(a^{2..3}, b)\| ?$

baa

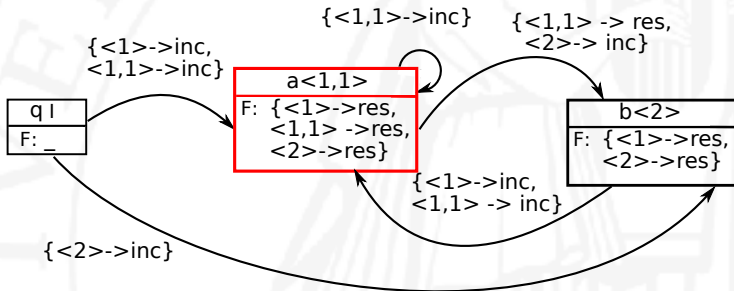
C: $\langle 1 \rangle \langle 1, 1 \rangle \langle 2 \rangle$
 max: 1 3 1
 min: 1 2 1
 value: 0 0 1



$baa \in \|\&(a^{2..3}, b)\| ?$

baa

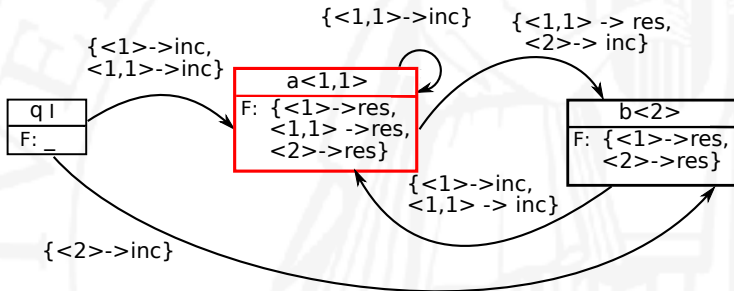
C: <1><1,1><2>
 max: 1 3 1
 min: 1 2 1
 value: 1 1 1



$baa \in \|\&(a^{2..3}, b)\| ?$

baa

C: <1><1,1><2>
 max: 1 3 1
 min: 1 2 1
 value: 1 2 1



- ▶ Construction based on *first*, *last*, and *follow* sets.
- ▶ first-set becomes $\Phi(q^l)$.
- ▶ last-set becomes the final configurations
- ▶ follow-set becomes the rest of Φ .
- ▶ Construction of first, last, and follow is polynomial-time

Result

- ▶ For any regular expression with numerical constraints and unordered concatenation, we can in polynomial time construct an FAC recognizing the language of the regular expression
- ▶ If the expression is *strongly 1-unambiguous*, the constructed FAC is deterministic
- ▶ A deterministic FAC can accept or reject a word in time linear in the length of the word.