XML Schema, UNIX Grep and Automata

Dag Hovland

October 29, 2008
Regular Expressions with Numerical Constraints

Usual Regular Expressions
Numerical Constraints

Applications

XML Schema
UNIX grep

Unambiguity

Finite Automata with Counters
From xkcd.com (1)
From xkcd.com (2)
From xkcd.com (3)
Applications of Regular Expressions

- Searching (UNIX grep, editors)
- Programming language compilers (Lexical Analyzers, e.g. flex)
- Document formats (XML Schema, SGML)
Usual Regular Expressions:

\[ r ::= r + r \mid r \cdot r \mid r^* \mid \Sigma \mid \epsilon \mid \emptyset \]

\( \Sigma \) alphabet
Regular Languages

$L(r)$: the regular language denoted by $r$

- $r \in \Sigma \cup \{\epsilon\} \Rightarrow L(r) = \{r\}$
- $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
- $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$

where $L(r_1) \cdot L(r_2) = \{w_1 \cdot w_2 | w_1 \in L(r_1) \land w_2 \in L(r_2)\}$

and $\epsilon \cdot w = w \cdot \epsilon = w$

- $L(r^*) = \bigcup_{0 \leq i} L(r)^i$

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

- $L(\emptyset) = \emptyset$
Regular Expressions with Numerical Constraints

- $r ::= r + r \mid r \cdot r \mid r^{N\ldots N} \mid r^* \mid \Sigma \mid \epsilon \mid \emptyset$
- $N = \{1, 2, 3, \ldots\}$
- $L(r_1^{l\ldots u}) = \bigcup_{l \leq i \leq u} L(r_1)^i$
- E.g.: $bab \in L((a + b)^{1\ldots 4})$
- In grep: $(a|b){1, 4}$
- $L(r^{l\ldots u}) = L(r^l(r + \epsilon)^{u-l})$
XML Schema

- XML document: a tree (labelled, ordered and rooted)
- Labels on children of internal node: a word
- XML Schema: specifies valid XML documents:
  - A set of types, for each type: a regular expression
  - Assigns a type to each node in valid XML documents
  - the “children-word” must be in the regular expression connected with the type
(a + b)\(^1\ldots4\) in XML Schema:

```xml
<xsd:choice minOccurs="1" maxOccurs="4">
  <xsd:element name="a" />
  <xsd:element name="b" />
</xsd:choice>
```
“grepping my mail”

- grep: Searches each line in files for matches to a regular expression
- Examples, GNU grep: ‘[0-9]{4} [A-Z][a-z]+’
- 4822 matches, 761 MB mail, ca. 50 secs. on 2.4 GHz processor
- Even better: ’Add?ress.*[0-9]{4} [A-Z][a-z]+’, ca. 2 secs, 48 matches
Faster, faster!

- \( \text{match} = \{ (r, w) \mid r \text{ is a regular expression matching } w \} \)
- \( \text{match} \in P \) (Stockmeyer and Meyer, 1973, Kilpeläinen and Tuhkanen, 2003)
- Searching: quadratic number of executions of match-algorithm
- Kilpeläinen and Tuhkanen: matching in quadratic space and time
Deterministic Finite Automata (DFA)

- $r$ regular expression: $\exists$ DFA $A$: recognizes $L(r)$ in linear time
- $\text{prefix} = \{(r, w) | \exists u, v : w = u \cdot v \land u \in L(r)\}$
- Searching in word $w$: $O(|w|)$ executions of prefix
- Deterministic Finite Automata: decides prefix in time linear in $|w|$
Super-polynomial behaviour of grep

- Translating usual regular expressions to DFA is super-polynomial
- From numerical constraints even worse
- GNU grep and Apache Xerces for XML Schema > 2GB
“fast-matcher” for MATCH. Polynomial time in regular expression, linear time in word-length

1-unambiguity (Brüggeammann-Klein, 1992): Polynomial-time construction of DFA from 1-unambiguous regular expressions without numerical constraints

In XML Schema: Element Declarations Consistent

No polynomial-time construction from 1-unambiguous regular expressions with numerical constraints known (Equivalent to NP=P, Kilpeläinen, 2004)
- Constraint normal form: polynomial-time decidable subclass of regular expressions
- Counter-unambiguous: polynomial-time decidable subclass of constraint normal form
- Finite Automata with Counters (FAC): polynomial-time construction from Constraint Normal Form
- For Counter-unambiguous: gives deterministic FAC
- Deterministic FAC: linear time matching, quadratic time searching
Comparing greps

- Example: '^[0-9]{4} [A-Z][a-z]+'
- 761 MB mail. ca. 70 secs on a 2, 4 GHz processor
- \( r = ((0 + \cdots + 9)^1\cdots^2 m((0 + \cdots + 9)^1\cdots^2 s)^0\cdots^60)^0\cdots^60 \)
- 2MB memory, less than 1 second