Formal Language Foundations and Schema Languages

Stefan Tittel

University of Dortmund

Seminar: Theoretical Foundations of XML Data Processing, February 2006

Overview

Overview

Overview

Overview

Motivation

XML:

- general-purpose markup language widely in use,
- syntactic structure described by XML schema languages.
  - Schema languages (like DTD) define the relative positions of pairs of corresponding tags.

What we do now:

- characterize the language class generated by DTDs,
  - What can we do with XML languages generated by a DTD?
  - What can we not do?
- transform (rather naively) DTDs to string grammars,
- analyze the languages created by these grammars.
  - How can we determine if a given language is in this language class?

Motivation

XML:

- general-purpose markup language widely in use,
- syntactic structure described by XML schema languages.
    - Schema languages (like DTD) define the relative positions of pairs of corresponding tags.

What we do now:

- characterize the language class generated by DTDs,
    - What can we do with XML languages generated by a DTD?
    - What can we not do?
- transform (rather naively) DTDs to string grammars,
- analyze the languages created by these grammars.
    - How can we determine if a given language is in this language class?

Definition of XML Grammars

A is the set of opening tags, $\overline{A}$ is the set of closing tags, r_a is a regular expression for each tag sort a.

Definition: XML Grammars

Grammar $G = (N, T, S, P)$ with:

- $N = X_a$ for all $a \in A$,
- $T = A \cup \overline{A}$,
- some $S \in N$,
- $P = \{X_a \rightarrow a r_a \overline{a}\}$ with $a \in A$, $\overline{a} \in \overline{A}$, $X_a \in N$.

Constraint: No empty tags and attributes, only reduced grammars.

Note

XML grammars as defined above only cover XML languages generated by DTDs.

Definition of XML Grammars

A is the set of opening tags, $\overline{A}$ is the set of closing tags, r_a is a regular expression for each tag sort a.

Definition: XML Grammars

Grammar $\mathrm{G} = (N, T, S, P)$ with:

- $N = X_a$ for all $a \in A$,
- $T = A \cup \overline{A}$,
- some $S \in N$,
- $P = \{X_a \rightarrow a r_a \overline{a}\}$ with $a \in A$, $\overline{a} \in \overline{A}$, $X_a \in N$.

Constraint: No empty tags and attributes, only reduced grammars.

Note

XML grammars as defined above only cover XML languages generated by DTDs.

Examples of XML Grammars and Dyck Primes

Example

$\{a^n\overline{a}^n\}$ is an XML language generated by $X \to a(X|\varepsilon)\overline{a}$.

Definition

The language D_A (or just D) of Dyck primes over $T = A \cup \overline{A}$ is generated by:

$$\begin{array}{lll} X & \to & \Sigma_{a\in A} X_a \\ X_a & \to & aX^*\overline{a}, \quad \text{for } a \in A \end{array}$$

D_A is the language of properly tag-parenthesized words. D_A is not an XML language (but $bD_A\overline{b}$ is).

D_a $(a \in A)$ is the subset of D_A, where each word starts with a and ends with $\overline{a}$. D_a is an XML language.

Examples of XML Grammars and Dyck Primes

Example

$\{a^n\overline{a}^n\}$ is an XML language generated by $X \to a(X|\varepsilon)\overline{a}$.

Definition

The language D_A (or just D) of Dyck primes over $T = A \cup \overline{A}$ is generated by:

$$\begin{array}{rcl} X & \to & \Sigma_{a \in A} X_a \\ X_a & \to & aX^*\overline{a}, \quad \text{for } a \in A \end{array}$$

D_A is the language of properly tag-parenthesized words. D_A is not an XML language (but $bD_A\overline{b}$ is).

D_a ($a \in A$) is the subset of D_A, where each word starts with a and ends with $\overline{a}$. D_a is an XML language.

$L_G(X)$ and Contexts

Definition

$L_G(X)$ is the language generated by a grammar G if X has been chosen as start symbol.

Hence $L_G(X)$ is the set of all words that can be generated from the non-terminal symbol X in the grammar G.

Definition: Contexts in L of Word w

$C_L(w)$ is the set of pairs of words (x, y) such that $xwy \in L$.

Example: $L = \{abc^n \mid n \in \mathbb{N}\}$

$C_L(b) = \{(a, c^n) \mid n \in \mathbb{N}\}$

$L_G(X)$ and Contexts

Definition

$L_G(X)$ is the language generated by a grammar G if X has been chosen as start symbol.

Hence $L_G(X)$ is the set of all words that can be generated from the non-terminal symbol X in the grammar G.

Definition: Contexts in L of Word w

$C_L(w)$ is the set of pairs of words (x, y) such that $xwy \in L$.

Example: $L = \{abc^n \mid n \in \mathbb{N}\}$

$C_L(b) = \{(a, c^n) \mid n \in \mathbb{N}\}$

$L_G(X)$ and Contexts

Definition

$L_G(X)$ is the language generated by a grammar G if X has been chosen as start symbol.

Hence $L_G(X)$ is the set of all words that can be generated from the non-terminal symbol X in the grammar G.

Definition: Contexts in L of Word w

$C_L(w)$ is the set of pairs of words (x, y) such that $xwy \in L$.

Example: $L = \{abc^n \mid n \in \mathbb{N}\}$

$C_L(b) = \{(a, c^n) \mid n \in \mathbb{N}\}$

Overview

Conditions for a Language to Be XML 1/4

At first we need to introduce the following definitions.

Definition

$F_a(L) := D_a \cap F(L)$ for each $a \in A$, where $F(L)$ is the set of factors of L.

Example: $L = \{a(b\overline{b})^n(c\overline{c})^n\overline{a} \mid n \geq 1\}$

$$F_a(L) = L, \quad F_b(L) = \{b\overline{b}\}, \quad F_c(L) = \{c\overline{c}\}.$$

Definition

If w is a Dyck prime in D_a it can be uniquely factorized as $au_{a_1}u_{a_2}\cdots u_{a_n}\overline{a}$ with $u_{a_i} \in D_{a_i}$ for $i = 1, \ldots, n$. Then $a_1a_2\cdots a_n \in A^*$ is what is called the trace of the word w.

Conditions for a Language to Be XML 1/4

At first we need to introduce the following definitions.

Definition

$F_a(L) := D_a \cap F(L)$ for each $a \in A$, where $F(L)$ is the set of factors of L.

Example: $L = \{a(b\overline{b})^n(c\overline{c})^n\overline{a} \mid n \geq 1\}$

$$F_a(L) = L, \quad F_b(L) = \{b\overline{b}\}, \quad F_c(L) = \{c\overline{c}\}.$$

Definition

If w is a Dyck prime in D_a it can be uniquely factorized as $au_{a_1}u_{a_2}\cdots u_{a_n}\overline{a}$ with $u_{a_i} \in D_{a_i}$ for $i = 1, \ldots, n$. Then $a_1a_2\cdots a_n \in A^*$ is what is called the trace of the word w.

Conditions for a Language to Be XML 1/4

At first we need to introduce the following definitions.

Definition

$F_a(L) := D_a \cap F(L)$ for each $a \in A$, where $F(L)$ is the set of factors of L.

Example: $L = \{a(b\overline{b})^n(c\overline{c})^n\overline{a} \mid n \geq 1\}$

$$F_a(L) = L, \quad F_b(L) = \{b\overline{b}\}, \quad F_c(L) = \{c\overline{c}\}.$$

Definition

If w is a Dyck prime in D_a it can be uniquely factorized as $au_{a_1}u_{a_2}\cdots u_{a_n}\overline{a}$ with $u_{a_i} \in D_{a_i}$ for $i = 1, \ldots, n$. Then $a_1a_2\cdots a_n \in A^*$ is what is called the trace of the word w.

Conditions for a Language to Be XML 2/4

Example

bd is the trace of $abc\overline{c}\overline{b}d\overline{d}\overline{a}$,
c is the trace of $bc\overline{cb}$.

Definition: Surface

$S_a(L)$ = set of all traces of words in $F_a(L)$.

Example: $L = \{a(b\overline{b})^n(c\overline{c})^n\overline{a} \mid n \geq 1\}$

- $S_a(L) = \{b^n c^n \mid n \geq 1\}$
- $S_b(L) = S_c(L) = \{\varepsilon\}$

Conditions for a Language to Be XML 2/4

Example

bd is the trace of $abc\bar{c}\bar{b}d\bar{d}\bar{a}$,
c is the trace of $bc\bar{c}\bar{b}$.

Definition: Surface

$S_a(L)$ = set of all traces of words in $F_a(L)$.

Example: $L = \{a(b\bar{b})^n(c\bar{c})^n\bar{a} \mid n \geq 1\}$

- $S_a(L) = \{b^n c^n \mid n \geq 1\}$
- $S_b(L) = S_c(L) = \{\varepsilon\}$

Conditions for a Language to Be XML 2/4

Example

bd is the trace of $abc\overline{c}\overline{b}d\overline{d}\overline{a}$,
c is the trace of $bc\overline{c}\overline{b}$.

Definition: Surface

$S_a(L)$ = set of all traces of words in $F_a(L)$.

Example: $L = \{a(b\overline{b})^n(c\overline{c})^n\overline{a} \mid n \geq 1\}$

- $S_a(L) = \{b^n c^n \mid n \geq 1\}$
- $S_b(L) = S_c(L) = \{\varepsilon\}$

Conditions for a Language to Be XML 3/4

Definition

1. $\emptyset$ (the empty set) is a regular set.
2. $\{\varepsilon\}$ is a regular set.
3. Every finite set is a regular set.
4. If R and S are regular sets, then $R \cup S$, RS, and R^* also are.

Theorem

A language L over $A \cup \overline{A}$ is an XML language if and only if the following three conditions hold true:

1. $L \subset D_\alpha$ *for some* $\alpha \in A$,
2. $C_L(w) = C_L(w')$ *for all* $a \in A$ *and* $w, w' \in F_a(L)$,
3. $S_a(L)$ *is a regular set for all* $a \in A$.

Conditions for a Language to Be XML 3/4

Definition

1. $\emptyset$ (the empty set) is a regular set.
2. $\{\varepsilon\}$ is a regular set.
3. Every finite set is a regular set.
4. If R and S are regular sets, then $R \cup S$, RS, and R^* also are.

Theorem

A language L over $A \cup \overline{A}$ is an XML language if and only if the following three conditions hold true:

1. $L \subset D_\alpha$ *for some* $\alpha \in A$,
2. $C_L(w) = C_L(w')$ *for all* $a \in A$ *and* $w, w' \in F_a(L)$,
3. $S_a(L)$ *is a regular set for all* $a \in A$.

Conditions for a Language to Be XML 4/4

Example

Non-XML grammar with start symbol S:

$$\begin{aligned} S &\rightarrow aTT\bar{a} \\ T &\rightarrow aTT\bar{a} \mid b\bar{b} \end{aligned}$$

- $L \subset D_a$ and $F_a(L) = L$,
- all $w \in L$ share the same $C_L(w)$ (by construction),
- $S_a(L) = (a \cup b)^2$ and $S_b(L) = \{\varepsilon\}$, i. e. both surfaces are regular.

All three conditions are satisfied. $\Rightarrow$ This grammar describes an XML language. $\Rightarrow$ There must be an XML grammar generating this language:

$$\begin{aligned} S &\rightarrow a(S|T)(S|T)\bar{a} \\ T &\rightarrow b\bar{b} \end{aligned}$$

Conditions for a Language to Be XML 4/4

Example

Non-XML grammar with start symbol S:

$$\begin{array}{lcl} S & \rightarrow & aTT\overline{a} \\ T & \rightarrow & aTT\overline{a} \mid b\overline{b} \end{array}$$

- $L \subset D_a$ and $F_a(L) = L$,
- all $w \in L$ share the same $C_L(w)$ (by construction),
- $S_a(L) = (a \cup b)^2$ and $S_b(L) = \{\varepsilon\}$, i. e. both surfaces are regular.

All three conditions are satisfied. $\Rightarrow$ This grammar describes an XML language. $\Rightarrow$ There must be an XML grammar generating this language:

$$\begin{array}{lcl} S & \rightarrow & a(S|T)(S|T)\overline{a} \\ T & \rightarrow & b\overline{b} \end{array}$$

Closure Under Union and Difference

Theorem

XML languages are closed neither under union nor difference.

Proof by counter-example:

- Consider $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$,
- $\{cL\overline{c}\}$ and $\{cM\overline{c}\}$ both are XML languages,
- $cab\overline{b}\overline{a}\overline{c}$ and $ca\overline{a}d\overline{d}\overline{c}$ are in H,
- $(c, d\overline{d}\overline{c})$ is in $C_H(a\overline{a})$, so it also has to be in $C_H(ab\overline{b}\overline{a})$,
- but $cab\overline{b}\overline{a}d\overline{d}\overline{c}$ is not in $H \Rightarrow$ XML languages are not closed under union,
- then (as direct consequence of De Morgan's theorem) XML languages are not closed under difference either.

Closure Under Union and Difference

Theorem

XML languages are closed neither under union nor difference.

Proof by counter-example:

- Consider $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$,
- $\{cL\overline{c}\}$ and $\{cM\overline{c}\}$ both are XML languages,
- $cab\overline{bac}$ and $ca\overline{a}d\overline{dc}$ are in H,
- $(c, d\overline{dc})$ is in $C_H(a\overline{a})$, so it also has to be in $C_H(ab\overline{ba})$,
- but $cab\overline{ba}d\overline{dc}$ is not in $H \Rightarrow$ XML languages are not closed under union,
- then (as direct consequence of De Morgan's theorem) XML languages are not closed under difference either.

Closure Under Union and Difference

Theorem

XML languages are closed neither under union nor difference.

Proof by counter-example:

- Consider $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$,
- $\{cL\overline{c}\}$ and $\{cM\overline{c}\}$ both are XML languages,
- $cab\overline{bac}$ and $ca\overline{a}d\overline{dc}$ are in H,
- $(c, d\overline{dc})$ is in $C_H(a\overline{a})$, so it also has to be in $C_H(ab\overline{ba})$,
- but $cab\overline{ba}d\overline{dc}$ is not in $H \Rightarrow$ XML languages are not closed under union,
- then (as direct consequence of De Morgan's theorem) XML languages are not closed under difference either.

Closure Under Union and Difference

Theorem

XML languages are closed neither under union nor difference.

Proof by counter-example:

- Consider $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$,
- $\{cL\overline{c}\}$ and $\{cM\overline{c}\}$ both are XML languages,
- $cab\overline{bac}$ and $ca\overline{a}d\overline{dc}$ are in H,
- $(c, d\overline{dc})$ is in $C_H(a\overline{a})$, so it also has to be in $C_H(ab\overline{ba})$,
- but $cab\overline{ba}d\overline{dc}$ is not in $H \Rightarrow$ XML languages are not closed under union,
- then (as direct consequence of De Morgan's theorem) XML languages are not closed under difference either.

Closure Under Union and Difference

Theorem

XML languages are closed neither under union nor difference.

Proof by counter-example:

- Consider $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$,
- $\{cL\overline{c}\}$ and $\{cM\overline{c}\}$ both are XML languages,
- $cab\overline{ba}\overline{c}$ and $ca\overline{a}d\overline{dc}$ are in H,
- $(c, d\overline{dc})$ is in $C_H(a\overline{a})$, so it also has to be in $C_H(ab\overline{ba})$,
- but $cab\overline{ba}d\overline{dc}$ is not in $H \Rightarrow$ XML languages are not closed under union,
- then (as direct consequence of De Morgan's theorem) XML languages are not closed under difference either.

Closure Under Union and Difference

Theorem

XML languages are closed neither under union nor difference.

Proof by counter-example:

- Consider $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$,
- $\{cL\overline{c}\}$ and $\{cM\overline{c}\}$ both are XML languages,
- $cab\overline{bac}$ and $ca\overline{a}d\overline{dc}$ are in H,
- $(c, d\overline{dc})$ is in $C_H(a\overline{a})$, so it also has to be in $C_H(ab\overline{ba})$,
- but $cab\overline{ba}d\overline{dc}$ is not in $H \Rightarrow$ XML languages are not closed under union,
- then (as direct consequence of De Morgan's theorem) XML languages are not closed under difference either.

Closure Under Union and Difference

Theorem

XML languages are closed neither under union nor difference.

Proof by counter-example:

- Consider $L = D^*_{\{a,b\}}$, $M = D^*_{\{a,d\}}$, and $H = \{cL\overline{c}\} \cup \{cM\overline{c}\}$,
- $\{cL\overline{c}\}$ and $\{cM\overline{c}\}$ both are XML languages,
- $cab\overline{bac}$ and $ca\overline{a}d\overline{dc}$ are in H,
- $(c, d\overline{dc})$ is in $C_H(a\overline{a})$, so it also has to be in $C_H(ab\overline{ba})$,
- but $cab\overline{ba}d\overline{dc}$ is not in $H \Rightarrow$ XML languages are not closed under union,
- then (as direct consequence of De Morgan's theorem) XML languages are not closed under difference either.

More Results

- XML languages are closed under intersection.
- For each XML language L there is exactly one reduced XML grammar generating L if variable names and entities are ignored.
- It is decidable if an XML language L is included in or equal to another XML language M.
- It is also decidable if a regular language $L \subset D_A$ is an XML language.
- It is however undecidable if a context-free language is an XML language.

More Results

- XML languages are closed under intersection.
- For each XML language L there is exactly one reduced XML grammar generating L if variable names and entities are ignored.
- It is decidable if an XML language L is included in or equal to another XML language M.
- It is also decidable if a regular language $L \subset D_A$ is an XML language.
- It is however undecidable if a context-free language is an XML language.

More Results

- XML languages are closed under intersection.
- For each XML language L there is exactly one reduced XML grammar generating L if variable names and entities are ignored.
- It is decidable if an XML language L is included in or equal to another XML language M.
- It is also decidable if a regular language $L \subset D_A$ is an XML language.
- It is however undecidable if a context-free language is an XML language.

More Results

- XML languages are closed under intersection.
- For each XML language L there is exactly one reduced XML grammar generating L if variable names and entities are ignored.
- It is decidable if an XML language L is included in or equal to another XML language M.
- It is also decidable if a regular language $L \subset D_A$ is an XML language.
- It is however undecidable if a context-free language is an XML language.

More Results

- XML languages are closed under intersection.
- For each XML language L there is exactly one reduced XML grammar generating L if variable names and entities are ignored.
- It is decidable if an XML language L is included in or equal to another XML language M.
- It is also decidable if a regular language $L \subset D_A$ is an XML language.
- It is however undecidable if a context-free language is an XML language.

Overview

Motivation

Why should we care about one-unambiguous regular languages?

Because in SGML the regular expressions (more precisely: model groups) on the right-hand side of productions have to be one-unambiguous.

But who cares about SGML?

The W3C does in its recommendation for XML:

> *For compatibility [with SGML], it is an error if the content model allows an element to match more than one occurrence of an element type in the content model.*

Furthermore one-unambiguity helps to efficiently parse a document.

Motivation

Why should we care about one-unambiguous regular languages?

Because in SGML the regular expressions (more precisely: model groups) on the right-hand side of productions have to be one-unambiguous.

But who cares about SGML?

The W3C does in its recommendation for XML:

> *For compatibility [with SGML], it is an error if the content model allows an element to match more than one occurrence of an element type in the content model.*

Furthermore one-unambiguity helps to efficiently parse a document.

Motivation

Why should we care about one-unambiguous regular languages?

Because in SGML the regular expressions (more precisely: model groups) on the right-hand side of productions have to be one-unambiguous.

But who cares about SGML?

The W3C does in its recommendation for XML:

> *For compatibility [with SGML], it is an error if the content model allows an element to match more than one occurrence of an element type in the content model.*

Furthermore one-unambiguity helps to efficiently parse a document.

Motivation

Why should we care about one-unambiguous regular languages?

Because in SGML the regular expressions (more precisely: model groups) on the right-hand side of productions have to be one-unambiguous.

But who cares about SGML?

The W3C does in its recommendation for XML:

> *For compatibility [with SGML], it is an error if the content model allows an element to match more than one occurrence of an element type in the content model.*

Furthermore one-unambiguity helps to efficiently parse a document.

Description of (One-)Unambiguous Regular Expressions

Informal Description

- If we can determine uniquely which symbol of a regular expression corresponds to a symbol in the input word (while knowing the whole word), the regular expression is unambiguous.
- If we can do so without looking beyond that symbol, the regular expression is one-unambiguous.

Example

- $(bc) + (bd)$ is unambiguous, but not one-unambiguous,
- $b(c + d)$ is one-unambiguous (hence also unambiguous).

A more formal way to distinguish between symbols is needed $\Rightarrow$ marking (soon).

Description of (One-)Unambiguous Regular Expressions

Informal Description

- If we can determine uniquely which symbol of a regular expression corresponds to a symbol in the input word (while knowing the whole word), the regular expression is unambiguous.
- If we can do so without looking beyond that symbol, the regular expression is one-unambiguous.

Example

- $(bc) + (bd)$ is unambiguous, but not one-unambiguous,
- $b(c + d)$ is one-unambiguous (hence also unambiguous).

A more formal way to distinguish between symbols is needed ⇒ marking (soon).

Description of (One-)Unambiguous Regular Expressions

Informal Description

- If we can determine uniquely which symbol of a regular expression corresponds to a symbol in the input word (while knowing the whole word), the regular expression is unambiguous.
- If we can do so without looking beyond that symbol, the regular expression is one-unambiguous.

Example

- $(bc) + (bd)$ is unambiguous, but not one-unambiguous,
- $b(c + d)$ is one-unambiguous (hence also unambiguous).

A more formal way to distinguish between symbols is needed $\Rightarrow$ marking (soon).

Description of (One-)Unambiguous Regular Expressions

Informal Description

- If we can determine uniquely which symbol of a regular expression corresponds to a symbol in the input word (while knowing the whole word), the regular expression is unambiguous.
- If we can do so without looking beyond that symbol, the regular expression is one-unambiguous.

Example

- $(bc) + (bd)$ is unambiguous, but not one-unambiguous,
- $b(c + d)$ is one-unambiguous (hence also unambiguous).

A more formal way to distinguish between symbols is needed $\Rightarrow$ marking (soon).

A New Perspective on the First Section

How is one-unambiguity incorporated into the XML grammars and languages of the previous section?

It is not. Thus the XML languages of the previous section are not even proper DTD languages.

Example: an XML language lacking one-unambiguity

$$\begin{aligned} N &= \{X_a, X_b\} \\ T &= \{a, \overline{a}, b, \overline{b}\} \\ S &= X_a \\ P &= \{X_a \rightarrow a X_b^* X_b^* \overline{a}, \\ &\quad X_b \rightarrow b \text{ something } \overline{b}\} \end{aligned}$$

A New Perspective on the First Section

How is one-unambiguity incorporated into the XML grammars and languages of the previous section?

It is not. Thus the XML languages of the previous section are not even proper DTD languages.

Example: an XML language lacking one-unambiguity

$$
\begin{aligned}
N &= \{X_a, X_b\} \\
T &= \{a, \overline{a}, b, \overline{b}\} \\
S &= X_a \\
P &= \{X_a \to a X_b^* X_b^* \overline{a}, \\
&\quad\; X_b \to b \text{ something } \overline{b}\}
\end{aligned}
$$

Marking of Regular Expressions

Example: $(a + b)^* a(ab)^*$

- $(a_1 + b_1)^* a_2(a_3 b_2)^*$ is a marking,
- $(a_4 + b_2)^* a_1(a_5 b_1)^*$ is a marking,
- $(a_1 + b_2)^* a_3(a_1 b_1)^*$ is not a marking.

Definition

- Assigning subscripts to occurrences of symbols,
- subscript is unique for each sort of symbols,
- marking of a regular expression E over alphabet Σ denoted by E' over the alphabet Π,
- dropping of subscripts denoted by $^\natural$, i. e. $(E')^\natural = E$ and $(w')^\natural = w$.

Marking of Regular Expressions

Example: $(a+b)^*a(ab)^*$

- $(a_1+b_1)^*a_2(a_3b_2)^*$ is a marking,
- $(a_4+b_2)^*a_1(a_5b_1)^*$ is a marking,
- $(a_1+b_2)^*a_3(a_1b_1)^*$ is not a marking.

Definition

- Assigning subscripts to occurrences of symbols,
- subscript is unique for each sort of symbols,
- marking of a regular expression E over alphabet Σ denoted by E' over the alphabet Π,
- dropping of subscripts denoted by $^\natural$, i. e. $(E')^\natural = E$ and $(w')^\natural = w$.

Marking of Regular Expressions

Example: $(a+b)^*a(ab)^*$

- $(a_1 + b_1)^* a_2 (a_3 b_2)^*$ is a marking,
- $(a_4 + b_2)^* a_1 (a_5 b_1)^*$ is a marking,
- $(a_1 + b_2)^* a_3 (a_1 b_1)^*$ is not a marking.

Definition

- Assigning subscripts to occurrences of symbols,
- subscript is unique for each sort of symbols,
- marking of a regular expression E over alphabet Σ denoted by E' over the alphabet Π,
- dropping of subscripts denoted by $^\natural$, i. e. $(E')^\natural = E$ and $(w')^\natural = w$.

Marking of Regular Expressions

Example: $(a+b)^*a(ab)^*$

- $(a_1+b_1)^*a_2(a_3b_2)^*$ is a marking,
- $(a_4+b_2)^*a_1(a_5b_1)^*$ is a marking,
- $(a_1+b_2)^*a_3(a_1b_1)^*$ is not a marking.

Definition

- Assigning subscripts to occurrences of symbols,
- subscript is unique for each sort of symbols,
- marking of a regular expression E over alphabet Σ denoted by E' over the alphabet Π,
- dropping of subscripts denoted by $^\natural$, i. e. $(E')^\natural = E$ and $(w')^\natural = w$.

Definition of One-Unambiguous Regular Languages

Definition

Let t, u, v, w be words over Π and $x, y \in \Pi$. A regular expr. E is one-unambiguous iff

$$uxv, uyw \in L(E') \wedge x \neq y \Rightarrow x^\natural \neq y^\natural.$$

If $\exists$ one-unambiguous E for $L \Rightarrow L$ is one-unambiguous.

Examples

- $E = (bc) + (bd)$, $E' = (b_1c_1) + (b_2d_1)$, $b_1c_1 \in L(E')$, $b_2d_1 \in L(E')$: $b_1 \neq b_2$, but $b = b$ therefore E is not one-unambiguous.
- $F = b(c + d)$, $F' = b_1(c_1 + d_1)$ satisfies the conditions $\Rightarrow F$ is one-unambiguous.

Definition of One-Unambiguous Regular Languages

Definition

Let t, u, v, w be words over Π and $x, y \in \Pi$. A regular expr. E is one-unambiguous iff

$$uxv, uyw \in L(E') \wedge x \neq y \Rightarrow x^\natural \neq y^\natural.$$

If $\exists$ one-unambiguous E for $L \Rightarrow L$ is one-unambiguous.

Examples

- $E = (bc) + (bd)$, $E' = (b_1c_1) + (b_2d_1)$, $b_1c_1 \in L(E')$, $b_2d_1 \in L(E')$: $b_1 \neq b_2$, but $b = b$ therefore E is not one-unambiguous.
- $F = b(c + d)$, $F' = b_1(c_1 + d_1)$ satisfies the conditions $\Rightarrow F$ is one-unambiguous.

Definition of One-Unambiguous Regular Languages

Definition

Let t, u, v, w be words over Π and $x, y \in \Pi$. A regular expr. E is one-unambiguous iff

$$uxv, uyw \in L(E') \land x \neq y \Rightarrow x^\natural \neq y^\natural.$$

If $\exists$ one-unambiguous E for $L \Rightarrow L$ is one-unambiguous.

Examples

- $E = (bc) + (bd)$, $E' = (b_1c_1) + (b_2d_1)$, $b_1c_1 \in L(E')$, $b_2d_1 \in L(E')$: $b_1 \neq b_2$, but $b = b$ therefore E is not one-unambiguous.
- $F = b(c + d)$, $F' = b_1(c_1 + d_1)$ satisfies the conditions $\Rightarrow F$ is one-unambiguous.

Definition of first, last and follow

Definition

Let L be a language.

$$\begin{array}{lcl} \text{first}(L) & := & \{b \mid \text{there is a word } w \text{ such that } bw \in L\} \\ \text{last}(L) & := & \{b \mid \text{there is a word } w \text{ such that } wb \in L\} \\ \text{follow}(L, a) & := & \{b \mid \text{there are words } v \text{ and } w \text{ such that} \\ & & vabw \in L\}, \text{ for each symbol } a \end{array}$$

For a regular expression E we define set(E) as set$(L(E))$.

Example: $E = b(c + d)$

$$\text{first}(E) = \{b\}, \quad \text{last}(E) = \text{follow}(E, b) = \{c, d\},$$
$$\text{follow}(E, c) = \text{follow}(E, d) = \emptyset$$

Definition of first, last and follow

Definition

Let L be a language.

$$\begin{array}{lcl} \text{first}(L) & := & \{b \mid \text{there is a word } w \text{ such that } bw \in L\} \\ \text{last}(L) & := & \{b \mid \text{there is a word } w \text{ such that } wb \in L\} \\ \text{follow}(L, a) & := & \{b \mid \text{there are words } v \text{ and } w \text{ such that} \\ & & vabw \in L\}, \text{ for each symbol } a \end{array}$$

For a regular expression E we define set(E) as set$(L(E))$.

Example: $E = b(c + d)$

$$\text{first}(E) = \{b\}, \quad \text{last}(E) = \text{follow}(E, b) = \{c, d\},$$
$$\text{follow}(E, c) = \text{follow}(E, d) = \emptyset$$

An Alternative Definition of One-Unambiguity

Theorem

A regular expression E is one-unambiguous iff

1. $\forall x, y \in \mathit{first}(E') : x \neq y \Rightarrow x^\natural \neq y^\natural$,
2. $\forall z \in \mathit{sym}(E') \wedge x, y \in \mathit{follow}(E', z) : x \neq y \Rightarrow x^\natural \neq y^\natural$,

where $\mathit{sym}(E')$ is the set of symbols occurring in E'.

Example: $E = b(c + d)$ marked as $b_1(c_1 + d_1)$

- $\text{first}(E') = \{b_1\}$ (condition 1 is satisfied),
- $\text{follow}(E, c_1) = \text{follow}(E, d_1) = \emptyset$, $\text{follow}(E, b) = \{c_1, d_1\}$; $c_1 \neq d_1 \Rightarrow c \neq d$ (condition 2 is satisfied).

E is one-unambiguous.

An Alternative Definition of One-Unambiguity

Theorem

A regular expression E is one-unambiguous iff

1. $\forall x, y \in first(E') : x \neq y \Rightarrow x^\natural \neq y^\natural$,
2. $\forall z \in sym(E') \wedge x, y \in follow(E', z) : x \neq y \Rightarrow x^\natural \neq y^\natural$,

where $sym(E')$ is the set of symbols occurring in E'.

Example: $E = b(c + d)$ marked as $b_1(c_1 + d_1)$

- first$(E') = \{b_1\}$ (condition 1 is satisfied),
- follow(E, c_1) = follow$(E, d_1) = \emptyset$, follow$(E, b) = \{c_1, d_1\}$; $c_1 \neq d_1 \Rightarrow c \neq d$ (condition 2 is satisfied).

E is one-unambiguous.

An Alternative Definition of One-Unambiguity

Theorem

A regular expression E is one-unambiguous iff

1. $\forall x, y \in \mathit{first}(E') : x \neq y \Rightarrow x^\natural \neq y^\natural$,
2. $\forall z \in \mathit{sym}(E') \wedge x, y \in \mathit{follow}(E', z) : x \neq y \Rightarrow x^\natural \neq y^\natural$,

where $\mathit{sym}(E')$ is the set of symbols occurring in E'.

Example: $E = b(c + d)$ marked as $b_1(c_1 + d_1)$

- $\text{first}(E') = \{b_1\}$ (condition 1 is satisfied),
- $\text{follow}(E, c_1) = \text{follow}(E, d_1) = \emptyset$, $\text{follow}(E, b) = \{c_1, d_1\}$; $c_1 \neq d_1 \Rightarrow c \neq d$ (condition 2 is satisfied).

E is one-unambiguous.

Glushkov Automata 1/4

Definition

Let E be a regular expression. The corresponding Glushkov automaton $G_E = (Q_E, \Sigma, \delta_E, q_I, F_E)$ is defined by:

1. $Q_E :=$ all symbols of E' and a new, initial state q_I,
2. for $a \in \Sigma$: $\delta_E(q_I, a) := \{x \mid x \in \text{first}(E'), x^\natural = a\}$,
3. for $x \in \text{sym}(E')$ and $a \in \Sigma$: $\delta_E(x, a) = \{y \mid y \in \text{follow}(E', x), y^\natural = a\}$,
4. $F_E = \begin{cases} \text{last}(E') \cup \{q_I\}, & \text{if } \varepsilon \in L(E) \\ \text{last}(E'), & \text{otherwise.} \end{cases}$

Glushkov Automata 1/4

Definition

Let E be a regular expression. The corresponding Glushkov automaton $G_E = (Q_E, \Sigma, \delta_E, q_I, F_E)$ is defined by:

1. $Q_E :=$ all symbols of E' and a new, initial state q_I,
2. for $a \in \Sigma$: $\delta_E(q_I, a) := \{x \mid x \in \text{first}(E'), x^\natural = a\}$,
3. for $x \in \text{sym}(E')$ and $a \in \Sigma$:
   $\delta_E(x, a) = \{y \mid y \in \text{follow}(E', x), y^\natural = a\}$,
4. $F_E = \begin{cases} \text{last}(E') \cup \{q_I\}, & \text{if } \varepsilon \in L(E) \\ \text{last}(E'), & \text{otherwise.} \end{cases}$

Glushkov Automata 1/4

Definition

Let E be a regular expression. The corresponding Glushkov automaton $G_E = (Q_E, \Sigma, \delta_E, q_I, F_E)$ is defined by:

1. $Q_E :=$ all symbols of E' and a new, initial state q_I,
2. for $a \in \Sigma$: $\delta_E(q_I, a) := \{x \mid x \in \text{first}(E'), x^\natural = a\}$,
3. for $x \in \text{sym}(E')$ and $a \in \Sigma$:
   $\delta_E(x, a) = \{y \mid y \in \text{follow}(E', x), y^\natural = a\}$,
4. $F_E = \begin{cases} \text{last}(E') \cup \{q_I\}, & \text{if } \varepsilon \in L(E) \\ \text{last}(E'), & \text{otherwise.} \end{cases}$

Glushkov Automata 1/4

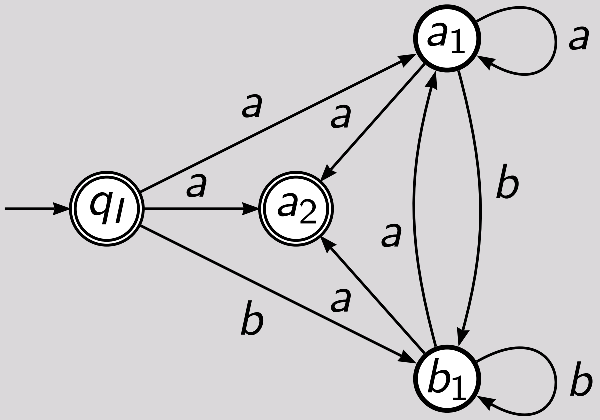
Definition

Let E be a regular expression. The corresponding Glushkov automaton $G_E = (Q_E, \Sigma, \delta_E, q_I, F_E)$ is defined by:

1. $Q_E :=$ all symbols of E' and a new, initial state q_I,
2. for $a \in \Sigma$: $\delta_E(q_I, a) := \{x \mid x \in \text{first}(E'), x^\natural = a\}$,
3. for $x \in \text{sym}(E')$ and $a \in \Sigma$:
   $\delta_E(x, a) = \{y \mid y \in \text{follow}(E', x), y^\natural = a\}$,
4. $F_E = \begin{cases} \text{last}(E') \cup \{q_I\}, & \text{if } \varepsilon \in L(E) \\ \text{last}(E'), & \text{otherwise.} \end{cases}$

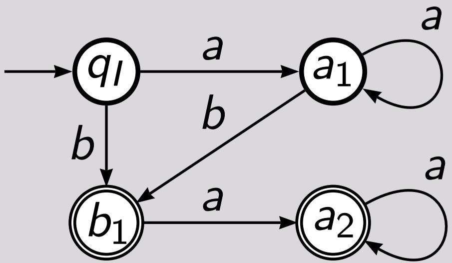
Glushkov Automata 2/4

Example: $(a+b)^*a+\varepsilon$ marked as $(a_1+b_1)^*a_2+\varepsilon$



Glushkov Automata 3/4

Example: a^*ba^* marked as $a_1^*b_1a_2^*$



Glushkov Automata 4/4

- No transition leads back to the initial state.
- Two transitions that lead to the same state have identical labels.
- G_E can be computed in time quadratic in the size of E.

Theorem

A regular expression E is one-unambiguous iff G_E is a DFA.

With Glushkov automata we can decide rather efficiently if a regular expression is one-unambiguous.

Glushkov Automata 4/4

- No transition leads back to the initial state.
- Two transitions that lead to the same state have identical labels.
- G_E can be computed in time quadratic in the size of E.

Theorem

A regular expression E is one-unambiguous iff G_E is a DFA.

With Glushkov automata we can decide rather efficiently if a regular expression is one-unambiguous.

Glushkov Automata 4/4

- No transition leads back to the initial state.
- Two transitions that lead to the same state have identical labels.
- G_E can be computed in time quadratic in the size of E.

Theorem

A regular expression E is one-unambiguous iff G_E is a DFA.

With Glushkov automata we can decide rather efficiently if a regular expression is one-unambiguous.

Overview

Initial Considerations 1/2

We know (mostly from the GTI lecture) . . .

- . . . that for each regular language L the corresponding minimum-state DFA $MS(L)$ is uniquely determined.
- . . . how minimizing a DFA can be achieved by equivalence-class construction.
- . . . that we can transform an NFA to an equivalent DFA using subset construction.
- . . . how to transform a regular expression to a Glushkov automaton.

Initial Considerations 1/2

We know (mostly from the GTI lecture) . . .

- . . . that for each regular language L the corresponding minimum-state DFA $MS(L)$ is uniquely determined.
- . . . how minimizing a DFA can be achieved by equivalence-class construction.
- . . . that we can transform an NFA to an equivalent DFA using subset construction.
- . . . how to transform a regular expression to a Glushkov automaton.

Initial Considerations 1/2

We know (mostly from the GTI lecture) ...

- ... that for each regular language L the corresponding minimum-state DFA $MS(L)$ is uniquely determined.
- ... how minimizing a DFA can be achieved by equivalence-class construction.
- ... that we can transform an NFA to an equivalent DFA using subset construction.
- ... how to transform a regular expression to a Glushkov automaton.

Initial Considerations 1/2

We know (mostly from the GTI lecture) . . .

- . . . that for each regular language L the corresponding minimum-state DFA $MS(L)$ is uniquely determined.
- . . . how minimizing a DFA can be achieved by equivalence-class construction.
- . . . that we can transform an NFA to an equivalent DFA using subset construction.
- . . . how to transform a regular expression to a Glushkov automaton.

Initial Considerations 2/2

- Idea: Examine the structural properties of $MS(L)$ that characterize an one-unambiguous language L.
- If E is a regular expression, $MS(L(E))$ can be achieved by minimizing G_E.
- If E is one-unambiguous, we do not need to use subset construction on G_E, because G_E already is a DFA.
- Question: What properties of Glushkov automata are preserved under minimization, but not necessarily under subset construction?

Initial Considerations 2/2

- Idea: Examine the structural properties of $MS(L)$ that characterize an one-unambiguous language L.
- If E is a regular expression, $MS(L(E))$ can be achieved by minimizing G_E.
- If E is one-unambiguous, we do not need to use subset construction on G_E, because G_E already is a DFA.
- Question: What properties of Glushkov automata are preserved under minimization, but not necessarily under subset construction?

Initial Considerations 2/2

- Idea: Examine the structural properties of $MS(L)$ that characterize an one-unambiguous language L.
- If E is a regular expression, $MS(L(E))$ can be achieved by minimizing G_E.
- If E is one-unambiguous, we do not need to use subset construction on G_E, because G_E already is a DFA.
- Question: What properties of Glushkov automata are preserved under minimization, but not necessarily under subset construction?

Initial Considerations 2/2

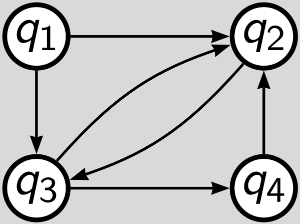
- Idea: Examine the structural properties of $MS(L)$ that characterize an one-unambiguous language L.
- If E is a regular expression, $MS(L(E))$ can be achieved by minimizing G_E.
- If E is one-unambiguous, we do not need to use subset construction on G_E, because G_E already is a DFA.
- Question: What properties of Glushkov automata are preserved under minimization, but not necessarily under subset construction?

Orbits

Definition: Orbit

For q being a state of an NFA, $\mathcal{O}(q)$ is the strongly connected component of q.

Example



$\mathcal{O}(q_1) = \{q_1\}$ $\quad$ $\mathcal{O}(q_2) = \{q_2, q_3, q_4\}$

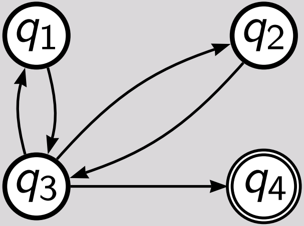
$\mathcal{O}(q_3) = \{q_2, q_3, q_4\}$ $\quad$ $\mathcal{O}(q_4) = \{q_2, q_3, q_4\}$

Gates

Definition

If $q \in F$ or $\exists q' \notin \mathcal{O}(q) : ((q, a), q') \in \delta$, then q is a gate of $\mathcal{O}(q)$.

Example



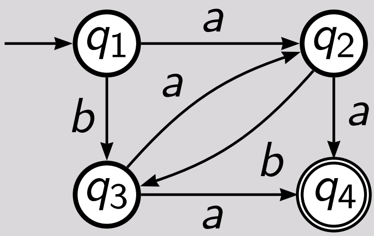
- q_1 and q_2 are not gates of their orbits.
- q_3 and q_4 are gates of their orbits.

Orbit Property

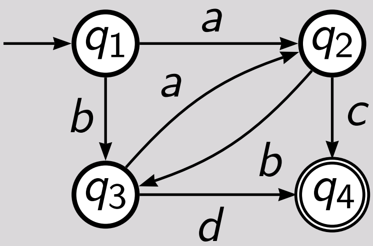
Definition

An NFA has the orbit property if all gates of each orbit have identical connections to the outside world.

Example



has orbit property



doesn't have it

Orbit Automata and Orbit Languages 1/2

Definition: Orbit Automaton

1. For a state q, restrict state set to $\mathcal{O}(q)$,
2. set q as the initial state,
3. set the gates of $\mathcal{O}(q)$ as the final states,
4. denote the resulting automaton as M_q.

Definition

- The language of M_q is called the orbit language of q.
- The languages $L(M_q)$, $q \in Q_M$ are called the orbit languages of M.

Orbit Automata and Orbit Languages 1/2

Definition: Orbit Automaton

1. For a state q, restrict state set to $\mathcal{O}(q)$,
2. set q as the initial state,
3. set the gates of $\mathcal{O}(q)$ as the final states,
4. denote the resulting automaton as M_q.

Definition

- The language of M_q is called the orbit language of q.
- The languages $L(M_q), q \in Q_M$ are called the orbit languages of M.

Orbit Automata and Orbit Languages 1/2

Definition: Orbit Automaton

1. For a state q, restrict state set to $\mathcal{O}(q)$,
2. set q as the initial state,
3. set the gates of $\mathcal{O}(q)$ as the final states,
4. denote the resulting automaton as M_q.

Definition

- The language of M_q is called the orbit language of q.
- The languages $L(M_q), q \in Q_M$ are called the orbit languages of M.

Orbit Automata and Orbit Languages 1/2

Definition: Orbit Automaton

1. For a state q, restrict state set to $\mathcal{O}(q)$,
2. set q as the initial state,
3. set the gates of $\mathcal{O}(q)$ as the final states,
4. denote the resulting automaton as M_q.

Definition

- The language of M_q is called the orbit language of q.
- The languages $L(M_q), q \in Q_M$ are called the orbit languages of M.

Orbit Automata and Orbit Languages 1/2

Definition: Orbit Automaton

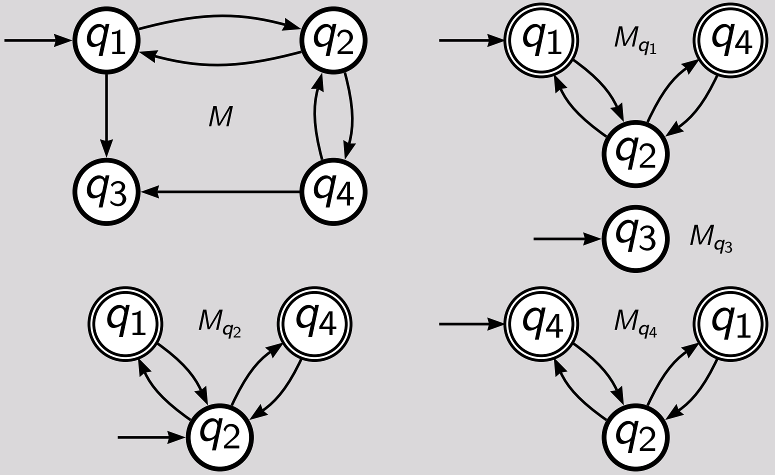
1. For a state q, restrict state set to $\mathcal{O}(q)$,
2. set q as the initial state,
3. set the gates of $\mathcal{O}(q)$ as the final states,
4. denote the resulting automaton as M_q.

Definition

- The language of M_q is called the orbit language of q.
- The languages $L(M_q), q \in Q_M$ are called the orbit languages of M.

Orbit Automata and Orbit Languages 2/2

Example



Characterization of One-Unambiguous Regular Languages

Theorem

M is a minimal DFA. If and only if

- *M has the orbit property,*
- *all orbit languages of M are one-unambiguous,*

then $L(M)$ is one-unambiguous.

An one-unambiguous regular expression for $L(M)$ is constructable from the one-unambiguous regular expressions for the orbit languages.

Definition

$\mathcal{O}(q)$ is trivial if $\mathcal{O}(q) = \{q\}$ and $(q, q) \notin \delta$.

Question: How can we decide if an orbit language is one-unambiguous if the orbit is not trivial?

Characterization of One-Unambiguous Regular Languages

Theorem

M is a minimal DFA. If and only if

- *M has the orbit property,*
- *all orbit languages of M are one-unambiguous,*

then $L(M)$ is one-unambiguous.

An one-unambiguous regular expression for $L(M)$ is constructable from the one-unambiguous regular expressions for the orbit languages.

Definition

$\mathcal{O}(q)$ is trivial if $\mathcal{O}(q) = \{q\}$ and $(q, q) \notin \delta$.

Question: How can we decide if an orbit language is one-unambiguous if the orbit is not trivial?

Characterization of One-Unambiguous Regular Languages

Theorem

M is a minimal DFA. If and only if

- *M has the orbit property,*
- *all orbit languages of M are one-unambiguous,*

then $L(M)$ is one-unambiguous.

An one-unambiguous regular expression for $L(M)$ is constructable from the one-unambiguous regular expressions for the orbit languages.

Definition

$\mathcal{O}(q)$ is trivial if $\mathcal{O}(q) = \{q\}$ and $(q, q) \notin \delta$.

Question: How can we decide if an orbit language is one-unambiguous if the orbit is not trivial?

Characterization of One-Unambiguous Regular Languages

Theorem

M is a minimal DFA. If and only if

- *M has the orbit property,*
- *all orbit languages of M are one-unambiguous,*

then $L(M)$ is one-unambiguous.

An one-unambiguous regular expression for $L(M)$ is constructable from the one-unambiguous regular expressions for the orbit languages.

Definition

$\mathcal{O}(q)$ is trivial if $\mathcal{O}(q) = \{q\}$ and $(q, q) \notin \delta$.

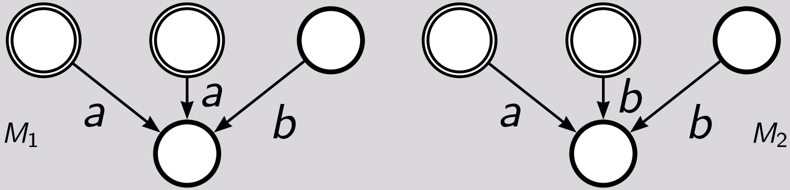
Question: How can we decide if an orbit language is one-unambiguous if the orbit is not trivial?

M-Consistency

Definition

- M is a DFA,
- $s \in \Sigma_M$ is *M-consistent* if
  $$\exists\ f(s) \in Q_M : \forall q \in F_M : ((q, s), f(s)) \in \delta_M,$$
- $S \subseteq \Sigma_M$ is M-consistent if $\forall s \in S : s$ is M-consistent.

Example



a is M_1-consistent

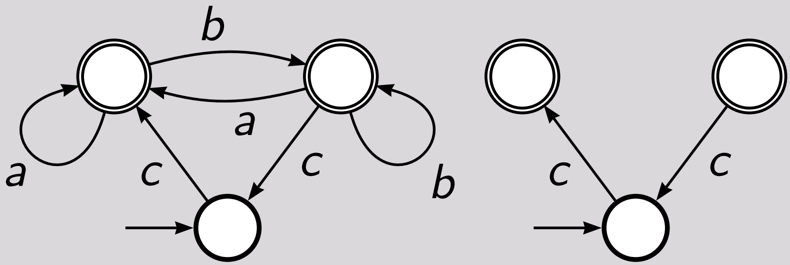
a is not M_2-consistent

S-Cut

Definition: S-Cut M_S of M

$\forall a \in S : \forall q \in Q_M : \forall q' \in F_M$: remove $((q, a), q')$ from δ_M

Example



M

$\{a, b\}$-cut of M

Conditions for a DFA to Be One-Unambiguous 1/2

Theorem

Let

- *M be a minimal DFA,*
- *S be an M-consistent set of symbols,*

now iff

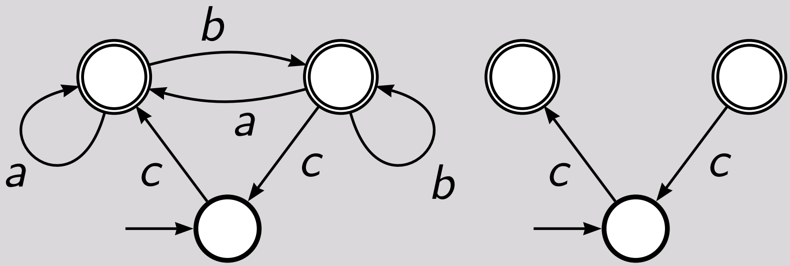
- *M_S satisfies the orbit property,*
- *all orbit languages of M_S are one-unambiguous,*

then $L(M)$ is one-unambiguous.

We will extend this theorem to a decision algorithm very soon.

Conditions for a DFA to Be One-Unambiguous 2/2

Example



M $\{a, b\}$-cut of M

The $\{a, b\}$-cut of M has only one-unambiguous orbits. Hence $L(M)$ is one-unambiguous and can be denoted by the one-unambiguous regular expression $c(a + b(\varepsilon + cc))^*$.

Decision Algorithm

```
boolean one-unambiguous (MinimalDFA M) {
   compute S := {a ∈ Σ | a is M-consistent};
   if (M has a single, trivial orbit) {return true;}
   if (M has a single, nontrivial orbit && S = ∅) {return false;}
   compute the orbits of M_S;
   if (!OrbitProperty(M_S)) {return false;}
   for (each orbit K of M_S) {
      choose x ∈ K;
      if (!one-unambiguous((M_S)_x) {return false;}
   }
   return true;
}
```

Overview

Closure

Definition

- L is a language,
- w is a word,
- $\{v \mid wv \in L\}$ is the derivative of L with respect to w and denoted by $w \backslash L$.

- The family of one-unambiguous regular languages is closed under derivatives.
- One-unambiguous regular expressions are not closed under derivatives, unless they are in a star normal form.
- The family of one-unambigous regular languages is not closed under union, concatenation or star.

Closure

Definition

- L is a language,
- w is a word,
- $\{v \mid wv \in L\}$ is the derivative of L with respect to w and denoted by $w \backslash L$.

- The family of one-unambiguous regular languages is closed under derivatives.
- One-unambiguous regular expressions are not closed under derivatives, unless they are in a star normal form.
- The family of one-unambigous regular languages is not closed under union, concatenation or star.

Closure

Definition

- L is a language,
- w is a word,
- $\{v \mid wv \in L\}$ is the derivative of L with respect to w and denoted by $w \backslash L$.

- The family of one-unambiguous regular languages is closed under derivatives.
- One-unambiguous regular expressions are not closed under derivatives, unless they are in a star normal form.
- The family of one-unambigous regular languages is not closed under union, concatenation or star.

Closure

Definition

- L is a language,
- w is a word,
- $\{v \mid wv \in L\}$ is the derivative of L with respect to w and denoted by $w \backslash L$.

- The family of one-unambiguous regular languages is closed under derivatives.
- One-unambiguous regular expressions are not closed under derivatives, unless they are in a star normal form.
- The family of one-unambigous regular languages is not closed under union, concatenation or star.

Overview

XML Schema Languages 1/2

Definition

- An XML schema describes constraints on the structure and content beyond the basic syntax constraints of XML itself.
- It is specified by an XML schema language.

Examples of XML schema languages

DTD, XML Schema, RELAX (NG), DSD, XDuce

Attention

"XML schema" $\neq$ "XML Schema"

XML Schema Languages 1/2

Definition

- An XML schema describes constraints on the structure and content beyond the basic syntax constraints of XML itself.
- It is specified by an XML schema language.

Examples of XML schema languages

DTD, XML Schema, RELAX (NG), DSD, XDuce

Attention

"XML schema" $\neq$ "XML Schema"

XML Schema Languages 1/2

Definition

- An XML schema describes constraints on the structure and content beyond the basic syntax constraints of XML itself.
- It is specified by an XML schema language.

Examples of XML schema languages

DTD, XML Schema, RELAX (NG), DSD, XDuce

Attention

"XML schema" $\neq$ "XML Schema"

XML Schema Languages 2/2

Example: XML Schema Specification of a Business Card (Extract)

```
<schema [...]
  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="logo" type="b:logo_type"/>
  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="logo_type">
    <attribute name="url" type="anyURI"/>
...
```

Motivation

We are interested in . . .

1. . . . expression power.
2. . . . closure properties.
3. . . . document validation.

Examples

1. Can I model my constraints with a certain XML schema language?
2. What XHTML 1.0 documents are still valid XHTML 1.1 documents?
3. Can I efficiently check if a document conforms to an XML schema?

Motivation

We are interested in ...

1. ... expression power.
2. ... closure properties.
3. ... document validation.

Examples

1. Can I model my constraints with a certain XML schema language?
2. What XHTML 1.0 documents are still valid XHTML 1.1 documents?
3. Can I efficiently check if a document conforms to an XML schema?

Motivation

We are interested in . . .

1. . . . expression power.
2. . . . closure properties.
3. . . . document validation.

Examples

1. Can I model my constraints with a certain XML schema language?
2. What XHTML 1.0 documents are still valid XHTML 1.1 documents?
3. Can I efficiently check if a document conforms to an XML schema?

Motivation

We are interested in ...

1. ... expression power.
2. ... closure properties.
3. ... document validation.

Examples

1. Can I model my constraints with a certain XML schema language?
2. What XHTML 1.0 documents are still valid XHTML 1.1 documents?
3. Can I efficiently check if a document conforms to an XML schema?

Motivation

We are interested in . . .

1. . . . expression power.
2. . . . closure properties.
3. . . . document validation.

Examples

1. Can I model my constraints with a certain XML schema language?
2. What XHTML 1.0 documents are still valid XHTML 1.1 documents?
3. Can I efficiently check if a document conforms to an XML schema?

Motivation

We are interested in ...

1. ... expression power.
2. ... closure properties.
3. ... document validation.

Examples

1. Can I model my constraints with a certain XML schema language?
2. What XHTML 1.0 documents are still valid XHTML 1.1 documents?
3. Can I efficiently check if a document conforms to an XML schema?

Regular Tree Grammars 1/3

Definition

A model group is a regular expression in which the following additional operators are allowed:

- ? – where $E?$ denotes $L(E + \varepsilon)$
- & – where $F\&G$ denotes $L(FG + GF)$
- $^+$ – where E^+ denotes $L(EE^*)$

Definition: Regular Tree Grammar $G = (N, T, P, S)$

- N = non-terminal symbols,
- T = terminal symbols,
- P = productions of the form $X \rightarrow a\ Expression$ with $X \in N$, $a \in T$ and $Expression$ model group over N,
- S = start symbols.

Regular Tree Grammars 1/3

Definition

A model group is a regular expression in which the following additional operators are allowed:

- ? – where $E?$ denotes $L(E + \varepsilon)$
- & – where $F\&G$ denotes $L(FG + GF)$
- $^+$ – where E^+ denotes $L(EE^*)$

Definition: Regular Tree Grammar $G = (N, T, P, S)$

- N = non-terminal symbols,
- T = terminal symbols,
- P = productions of the form $X \rightarrow a\ Expression$ with $X \in N$, $a \in T$ and $Expression$ model group over N,
- S = start symbols.

Regular Tree Grammars 2/3

Example: A Tree Grammar for a DTD

```
<!DOCTYPE book [
   <!ELEMENT book (author+, publisher) >
   <!ELEMENT author (#PCDATA) >
   <!ELEMENT publisher (EMPTY) >
   <!ATTLIST publisher Name CDATA #IMPLIED >
]>
```

$$
\begin{aligned}
N &= \{Book, Author, Publisher, Pcdata\},\\
T &= \{book, author, publisher, pcdata\},\\
S &= \{Book\},\\
P &= \{Book \to book(Author^{+}, Publisher),\\
&\quad Author \to author(Pcdata),\\
&\quad Publisher \to publisher(\varepsilon),\\
&\quad Pcdata \to pcdata(\varepsilon)\}.
\end{aligned}
$$

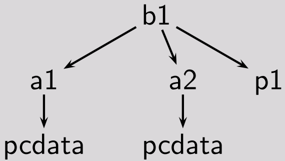
Regular Tree Grammars 3/3

Example

A possible document complying with this DTD:

```
<book>
   <author>J. E. Hopcroft</author>
   <author>J. D. Ullman</author>
   <publisher Name="Addison-Wesley"/>
</book>
```

An instance tree for this document:



Normal Form 1 (NF1) 1/2

Definition: Grammar in Normal Form 1 (NF1)

Grammar $G = (N1, N2, T, P1, P2, S)$ with

- T and S as usual,
- $N1$ = non-terminal symbols used for deriving trees,
- $N2$ = non-terminal symbols used for content-model spec.,
- $P1$ = productions of the form $A \rightarrow aX$ with $A \in N1, X \in N2$, $a \in T$ (only one production per symbol $\in N1$),
- $P2$ = prod. of the form $X \rightarrow Exp$ with $X \in N2$, Exp model group over $N1$ (only one production per symbol $\in N2$).

Definition

contentModel(A) ($A \in N1$) is the model group over $N1$ denoting the content of A.

Normal Form 1 (NF1) 1/2

Definition: Grammar in Normal Form 1 (NF1)

Grammar $G = (N1, N2, T, P1, P2, S)$ with

- T and S as usual,
- $N1$ = non-terminal symbols used for deriving trees,
- $N2$ = non-terminal symbols used for content-model spec.,
- $P1$ = productions of the form $A \rightarrow aX$ with $A \in N1, X \in N2$, $a \in T$ (only one production per symbol $\in N1$),
- $P2$ = prod. of the form $X \rightarrow Exp$ with $X \in N2$, Exp model group over $N1$ (only one production per symbol $\in N2$).

Definition

$contentModel(A)$ ($A \in N1$) is the model group over $N1$ denoting the content of A.

Normal Form 1 (NF1) 1/2

Definition: Grammar in Normal Form 1 (NF1)

Grammar $G = (N1, N2, T, P1, P2, S)$ with

- T and S as usual,
- $N1$ = non-terminal symbols used for deriving trees,
- $N2$ = non-terminal symbols used for content-model spec.,
- $P1$ = productions of the form $A \rightarrow aX$ with $A \in N1, X \in N2$, $a \in T$ (only one production per symbol $\in N1$),
- $P2$ = prod. of the form $X \rightarrow Exp$ with $X \in N2$, Exp model group over $N1$ (only one production per symbol $\in N2$).

Definition

$contentModel(A)$ ($A \in N1$) is the model group over $N1$ denoting the content of A.

Normal Form 1 (NF1) 1/2

Definition: Grammar in Normal Form 1 (NF1)

Grammar $G = (N1, N2, T, P1, P2, S)$ with

- T and S as usual,
- $N1$ = non-terminal symbols used for deriving trees,
- $N2$ = non-terminal symbols used for content-model spec.,
- $P1$ = productions of the form $A \rightarrow aX$ with $A \in N1, X \in N2$, $a \in T$ (only one production per symbol $\in N1$),
- $P2$ = prod. of the form $X \rightarrow Exp$ with $X \in N2$, Exp model group over $N1$ (only one production per symbol $\in N2$).

Definition

$contentModel(A)$ ($A \in N1$) is the model group over $N1$ denoting the content of A.

Normal Form 1 (NF1) 1/2

Definition: Grammar in Normal Form 1 (NF1)

Grammar $G = (N1, N2, T, P1, P2, S)$ with

- T and S as usual,
- $N1$ = non-terminal symbols used for deriving trees,
- $N2$ = non-terminal symbols used for content-model spec.,
- $P1$ = productions of the form $A \rightarrow aX$ with $A \in N1, X \in N2$, $a \in T$ (only one production per symbol $\in N1$),
- $P2$ = prod. of the form $X \rightarrow Exp$ with $X \in N2$, Exp model group over $N1$ (only one production per symbol $\in N2$).

Definition

$contentModel(A)$ $(A \in N1)$ is the model group over $N1$ denoting the content of A.

Normal Form 1 (NF1) 1/2

Definition: Grammar in Normal Form 1 (NF1)

Grammar $G = (N1, N2, T, P1, P2, S)$ with

- T and S as usual,
- $N1$ = non-terminal symbols used for deriving trees,
- $N2$ = non-terminal symbols used for content-model spec.,
- $P1$ = productions of the form $A \rightarrow aX$ with $A \in N1, X \in N2$, $a \in T$ (only one production per symbol $\in N1$),
- $P2$ = prod. of the form $X \rightarrow Exp$ with $X \in N2$, Exp model group over $N1$ (only one production per symbol $\in N2$).

Definition

$contentModel(A)$ ($A \in N1$) is the model group over $N1$ denoting the content of A.

Normal Form 1 (NF1) 2/2

Example: The Grammar of the Last Example in NF1

$$
\begin{aligned}
N1 &= \{Book, Author, Publisher, Pcdata\},\\
N2 &= \{BOOK, AUTHOR, PUBLISHER, PCDATA\},\\
T &= \{book, author, publisher, pcdata\},\\
P1 &= \{Book \rightarrow book\ BOOK, Author \rightarrow author\ AUTHOR,\\
&\quad Publisher \rightarrow publisher\ PUBLISHER, Pcdata \rightarrow\\
&\quad pcdata\ PCDATA\},\\
P2 &= \{BOOK \rightarrow (Author^+, Publisher), AUTHOR \rightarrow Pcdata,\\
&\quad PUBLISHER \rightarrow \varepsilon, PCDATA \rightarrow \varepsilon\},\\
S &= \{Book\}.
\end{aligned}
$$

$$contentModel(Book) = (Author^+, Publisher)$$

From now on upper- and lower-casing will be used like in this example to distinguish between symbols in $N1, N2$ and T.

Normal Form 1 (NF1) 2/2

Example: The Grammar of the Last Example in NF1

$$
\begin{aligned}
N1 &= \{Book, Author, Publisher, Pcdata\},\\
N2 &= \{BOOK, AUTHOR, PUBLISHER, PCDATA\},\\
T &= \{book, author, publisher, pcdata\},\\
P1 &= \{Book \rightarrow book\ BOOK, Author \rightarrow author\ AUTHOR,\\
&\quad Publisher \rightarrow publisher\ PUBLISHER, Pcdata \rightarrow\\
&\quad pcdata\ PCDATA\},\\
P2 &= \{BOOK \rightarrow (Author^{+}, Publisher), AUTHOR \rightarrow Pcdata,\\
&\quad PUBLISHER \rightarrow \varepsilon, PCDATA \rightarrow \varepsilon\},\\
S &= \{Book\}.
\end{aligned}
$$

$$contentModel(Book) = (Author^{+}, Publisher)$$

From now on upper- and lower-casing will be used like in this example to distinguish between symbols in $N1, N2$ and T.

Overview

Local Tree Grammars

Definition: Tree-Locality Constraint

$\forall a \in T$ there is no more than one rule of the form $A \rightarrow aX$ in $P1$.

Definition: Local Tree Grammar (LTG)

A regular tree grammar that satisfies the tree-locality constraint.

Example

$$
\begin{aligned}
N1 &= \{Out, In, Pcd\} \\
N2 &= \{OUT, IN, PCD\} \\
T &= \{out, in, pcd\} \\
P1_a &= \{Out \rightarrow out\ OUT, In \rightarrow in\ IN, Pcd \rightarrow pcd\ PCD\} \\
P1_b &= \{Out \rightarrow out\ OUT, In \rightarrow out\ IN, Pcd \rightarrow pcd\ PCD\} \\
P2 &= \{OUT \rightarrow In, IN \rightarrow Pcd, PCD \rightarrow \varepsilon\}
\end{aligned}
$$

$(N1, N2, T, P1_a, P2)$ is an LTG, $(N1, N2, T, P1_b, P2)$ is not.

Local Tree Grammars

Definition: Tree-Locality Constraint

$\forall a \in T$ there is no more than one rule of the form $A \rightarrow aX$ in $P1$.

Definition: Local Tree Grammar (LTG)

A regular tree grammar that satisfies the tree-locality constraint.

Example

$$\begin{aligned}
N1 &= \{Out, In, Pcd\} \\
N2 &= \{OUT, IN, PCD\} \\
T &= \{out, in, pcd\} \\
P1_a &= \{Out \rightarrow out\ OUT, In \rightarrow in\ IN, Pcd \rightarrow pcd\ PCD\} \\
P1_b &= \{Out \rightarrow out\ OUT, In \rightarrow out\ IN, Pcd \rightarrow pcd\ PCD\} \\
P2 &= \{OUT \rightarrow In, IN \rightarrow Pcd, PCD \rightarrow \varepsilon\}
\end{aligned}$$

$(N1, N2, T, P1_a, P2)$ is an LTG, $(N1, N2, T, P1_b, P2)$ is not.

Single-Type Constraint Languages 1/2

Definition

Two different non-terminals A and B are called competing with each other if

- one production rule has A in the left-hand side,
- another production rule has B in the left-hand side, and
- these two production rules share the same terminal in the right-hand side.

Definition: Single-Type Constraint Grammar

- For each production rule, non-terminals in its content model do not compete with each other,
- start symbols do not compete with each other.

Single-Type Constraint Languages 1/2

Definition

Two different non-terminals A and B are called competing with each other if

- one production rule has A in the left-hand side,
- another production rule has B in the left-hand side, and
- these two production rules share the same terminal in the right-hand side.

Definition: Single-Type Constraint Grammar

- For each production rule, non-terminals in its content model do not compete with each other,
- start symbols do not compete with each other.

Single-Type Constraint Languages 2/2

Definition

A tree language is a single-type constraint language if it is generated by a single-type constraint grammar.

Example

- $P_1 = \{A \rightarrow B, A \rightarrow C, B \rightarrow a, C \rightarrow b\}$ satisfies the s.-t. c.,
- $P_2 = \{A \rightarrow B, A \rightarrow C, B \rightarrow a, C \rightarrow a\}$ doesn't.

Single-type constraint languages and local tree languages are ...

- ... closed under intersection.
- ... not closed under union.
- ... not closed under difference.

Single-Type Constraint Languages 2/2

Definition

A tree language is a single-type constraint language if it is generated by a single-type constraint grammar.

Example

- $P_1 = \{A \rightarrow B, A \rightarrow C, B \rightarrow a, C \rightarrow b\}$ satisfies the s.-t. c.,
- $P_2 = \{A \rightarrow B, A \rightarrow C, B \rightarrow a, C \rightarrow a\}$ doesn't.

Single-type constraint languages and local tree languages are ...

- ... closed under intersection.
- ... not closed under union.
- ... not closed under difference.

Local Tree Languages $\subset$ Single Type Constraint Languages

Theorem

Local tree languages form a proper subclass of single-type constraint languages.

Proof:

$\Longrightarrow$: A local tree language satisfies the single-type constraint by definition.

$\Longleftarrow$:

- Consider a regular tree grammar with $A, B \in N1 \wedge A \neq B \wedge root(A) = root(B)$.
- This grammar can satisfy the single-type constraint.
- This grammar is not a local tree grammar.

Local Tree Languages ⊂ Single Type Constraint Languages

Theorem

Local tree languages form a proper subclass of single-type constraint languages.

Proof:

$\Longrightarrow$: A local tree language satisfies the single-type constraint by definition.

$\Longleftarrow$:

- Consider a regular tree grammar with $A, B \in N1 \wedge A \neq B \wedge root(A) = root(B)$.
- This grammar can satisfy the single-type constraint.
- This grammar is not a local tree grammar.

Local Tree Languages ⊂ Single Type Constraint Languages

Theorem

Local tree languages form a proper subclass of single-type constraint languages.

Proof:

$\Longrightarrow$: A local tree language satisfies the single-type constraint by definition.

$\Longleftarrow$:

- Consider a regular tree grammar with $A, B \in N1 \wedge A \neq B \wedge root(A) = root(B)$.
- This grammar can satisfy the single-type constraint.
- This grammar is not a local tree grammar.

Overview

DTD and DSD

DTD

- TDLL(1),
- local tree grammar.

DSD

- No constraints on the production rules,
- theoretically any regular tree grammar can be expressed in DSD,
- parsing algorithm uses greedy technique with one vertical and horizontal lookahead,
- acceptance of all and only TDLL(1) languages is suspected.

DTD and DSD

DTD

- TDLL(1),
- local tree grammar.

DSD

- No constraints on the production rules,
- theoretically any regular tree grammar can be expressed in DSD,
- parsing algorithm uses greedy technique with one vertical and horizontal lookahead,
- acceptance of all and only TDLL(1) languages is suspected.

XML Schema and RELAX

XML Schema

- TDLL(1) with single-type constraint,
- group definitions allowed to contain other group definitions without restriction ⇒ context-free content models possible (specification mistake?).

RELAX

- Any regular tree grammar.

XML Schema and RELAX

XML Schema

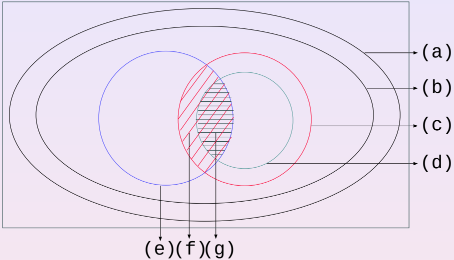
- TDLL(1) with single-type constraint,
- group definitions allowed to contain other group definitions without restriction ⇒ context-free content models possible (specification mistake?).

RELAX

- Any regular tree grammar.

Expression Power

This figure is from [3].



(a) regular tree grammars (RELAX, XDuce)
(b) TD(1) grammars
(c) single-type constraint grammars
(d) local tree grammars
(e) TDLL(1) grammars
(f) TDLL(1) w/ single-type constraint (XML Schema, DSD?)
(g) TDLL(1) w/ tree-locality constraint (DTD)

References 1/2

- Jean Berstel, Luc Boasson. *Formal Properties of XML Grammars and Languages*. Acta Informatica, 38:649–671, 2002.
- Anne Brüggemann-Klein, Derick Wood. *One-Unambiguous Regular Languages*. Information and Computation, 140:229–253, 1998.

[3] Dongwon Lee, Murali Mani, Makoto Murata. *Reasoning about XML Schema Languages using Formal Language Theory*. Technical Report, IBM Almaden Research Center, 2000. Log #95071.

References 2/2

- Thomas Schwentick. *Formal Methods for XML: Algorithms & Complexity*. Internet: <http://lrb.cs.uni-dortmund.de/~tick/Talks/edbtp.pdf>, 2004 (cited 2006–01–26).
- Anders Møller, Michael I. Schwartzbach. *The XML Revolution: Technologies for the future Web*. Internet: <http://www.brics.dk/~amoeller/XML/>, 2003 (cited 2006–01–26).
- Dongwon Lee, Murali Mani, Makoto Murata. *Taxonomy of XML Schema Languages Using Formal Language Theory*. Proceedings of the 2001 Conference on Extreme Markup Languages, 2001.