Simplifying XML Schema: Effortless Handling of Nondeterministic Regular Expressions

Geert Jan Bex[1] and Wouter Gelade[1] and Wim Martens[2] and Frank Neven[1]

[1]Hasselt University
[2]University of Dortmund

July, 2009

XML Schema

XML Schema is ...

- A language for defining the structure of XML documents.
- W3C Standard
- Successor of DTD

XML Schema

XML Schema is ...

- A language for defining the structure of XML documents.
- W3C Standard
- Successor of DTD

Why a schema for XML documents?

- Provides semantics to the data
- Very useful for optimization
- Necessary for data integration
- ...

XML Schema: Abstract Syntax

XSD

```
<xsd:element name="store" type="store"/>

<xsd:complexType name="store">
  <xsd:sequence>
    <xsd:element name="order" type="order" minOccurs="0" maxOccurs="unbour
    <xsd:element name="stock" type="stock"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="order">
  <xsd:sequence>
    <xsd:element name="customer" type="customer"/>
    <xsd:element name="item" type="item1" minOccurs="1" maxOccurs="unbounc
  </xsd:sequence>
</xsd:complexType>
```

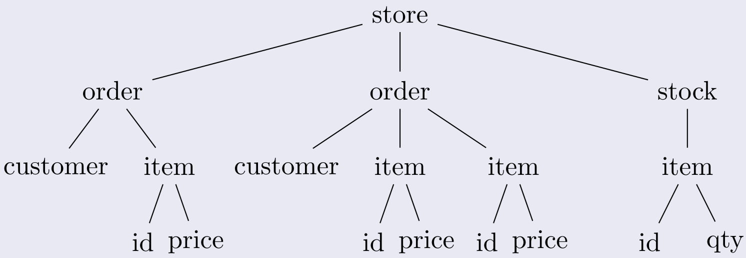
$$\begin{aligned} \textit{root} &\rightarrow \textit{store} \\ \textit{store} &\rightarrow \textit{order}^* \ \textit{stock} \\ \textit{order} &\rightarrow \textit{customer} \ \textit{item}_1^+ \end{aligned}$$

XML Schema

XSD

root	$\rightarrow$	store	stock	$\rightarrow$	item_2^*
store	$\rightarrow$	order^* stock	item_1	$\rightarrow$	id price
order	$\rightarrow$	customer item_1^+	item_2	$\rightarrow$	id qty

XML Document: Tree

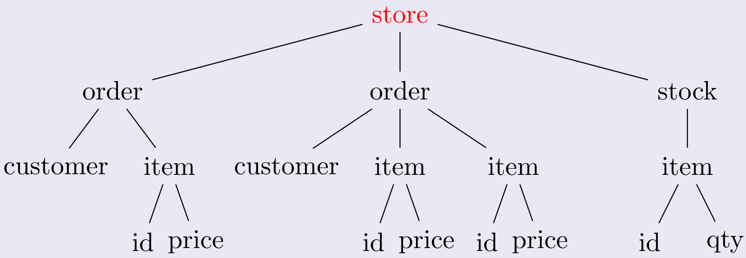


XSD Validation

XSD

root	$\rightarrow$	store	stock	$\rightarrow$	item_2^*
store	$\rightarrow$	order^* stock	item_1	$\rightarrow$	id price
order	$\rightarrow$	customer item_1^+	item_2	$\rightarrow$	id qty

XML Document: Tree

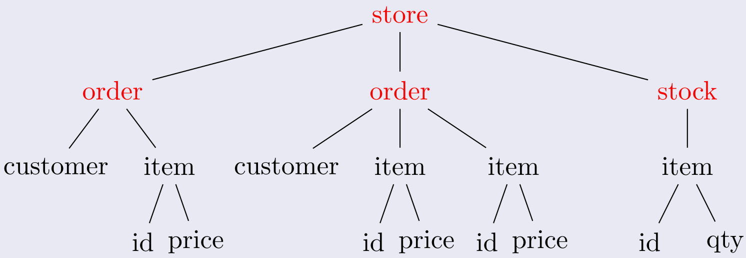


XSD Validation

XSD Validation

root	$\rightarrow$	store	stock	$\rightarrow$	item_2^*
store	$\rightarrow$	order^* stock	item_1	$\rightarrow$	id price
order	$\rightarrow$	customer item_1^+	item_2	$\rightarrow$	id qty

XML Document: Tree

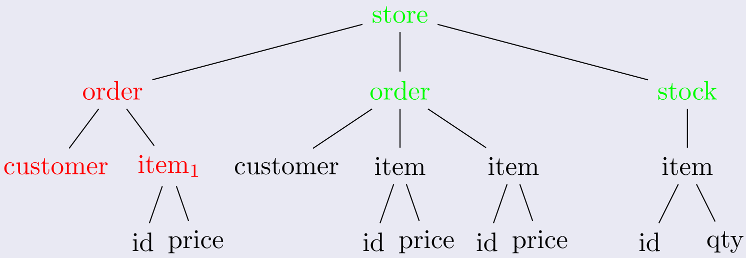


XSD Validation

XSD Validation

root	$\rightarrow$	store	stock	$\rightarrow$	item_2^*
store	$\rightarrow$	order^* stock	item_1	$\rightarrow$	id price
order	$\rightarrow$	customer item_1^+	item_2	$\rightarrow$	id qty

XML Document: Tree

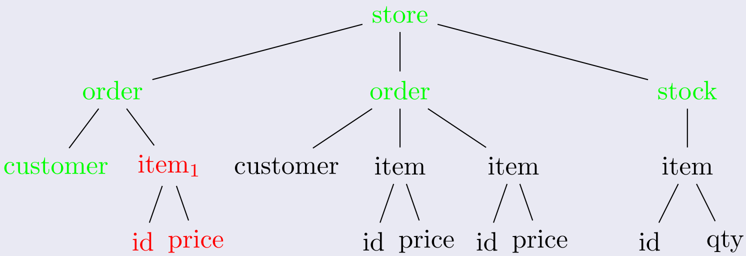


XSD Validation

XSD Validation

root	$\rightarrow$	store	stock	$\rightarrow$	item_2^*
store	$\rightarrow$	order^* stock	item_1	$\rightarrow$	id price
order	$\rightarrow$	customer item_1^+	item_2	$\rightarrow$	id qty

XML Document: Tree

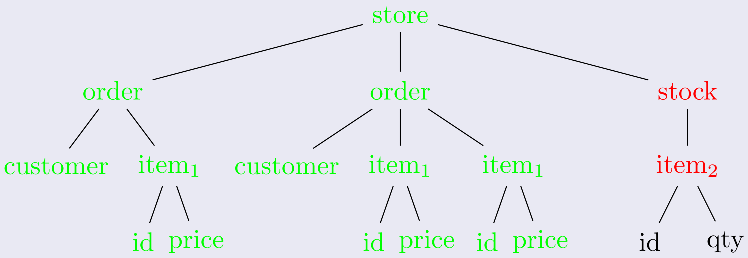


XSD Validation

XSD Validation

root	$\rightarrow$	store	stock	$\rightarrow$	item_2^*
store	$\rightarrow$	order^* stock	item_1	$\rightarrow$	id price
order	$\rightarrow$	customer item_1^+	item_2	$\rightarrow$	id qty

XML Document: Tree

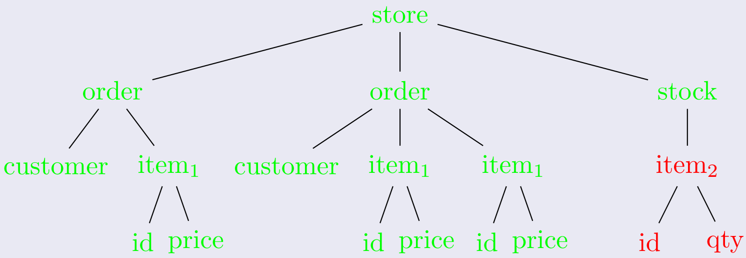


XSD Validation

XSD Validation

root	$\rightarrow$	store	stock	$\rightarrow$	item_2^*
store	$\rightarrow$	order^* stock	item_1	$\rightarrow$	id price
order	$\rightarrow$	customer item_1^+	item_2	$\rightarrow$	id qty

XML Document: Tree



XML Schema

XML Schema is ...

a simple grammar-based formalism using regular expressions

Regular expressions are great

- Easy to use
- Robust class of languages: closed under union, intersection, complement, . . .
- Very well understood

Deterministic Regular Expressions

UPA constraint

All content models must be deterministic regular expressions.

Definition

A regular expression r is deterministic if when matching any string from left to right against r, we can deterministically match every symbol against a position in r, without looking ahead in the string.

Deterministic Regular Expressions

UPA constraint

All content models must be deterministic regular expressions.

Definition

A regular expression r is deterministic if when matching any string from left to right against r, we can deterministically match every symbol against a position in r, without looking ahead in the string.

Example

- $(ab)^*$ is deterministic.
- $(ab)^*a$ is not deterministic

Deterministic Regular Expressions

UPA constraint

All content models must be deterministic regular expressions.

Definition

A regular expression r is deterministic if when matching any string from left to right against r, we can deterministically match every symbol against a position in r, without looking ahead in the string.

Example

- $(ab)^*$ is deterministic. Example: $abab$
- $(ab)^*a$ is not deterministic

Deterministic Regular Expressions

UPA constraint

All content models must be deterministic regular expressions.

Definition

A regular expression r is deterministic if when matching any string from left to right against r, we can deterministically match every symbol against a position in r, without looking ahead in the string.

Example

- $(ab)^*$ is deterministic. Example: $abab$
- $(ab)^*a$ is not deterministic

Deterministic Regular Expressions

UPA constraint

All content models must be deterministic regular expressions.

Definition

A regular expression r is deterministic if when matching any string from left to right against r, we can deterministically match every symbol against a position in r, without looking ahead in the string.

Example

- $(ab)^*$ is deterministic. Example: $abab$
- $(ab)^*a$ is not deterministic

Deterministic Regular Expressions

UPA constraint

All content models must be deterministic regular expressions.

Definition

A regular expression r is deterministic if when matching any string from left to right against r, we can deterministically match every symbol against a position in r, without looking ahead in the string.

Example

- $(ab)^*$ is deterministic. Example: $abab$
- $(ab)^* a$ is not deterministic

Deterministic Regular Expressions

UPA constraint

All content models must be deterministic regular expressions.

Definition

A regular expression r is deterministic if when matching any string from left to right against r, we can deterministically match every symbol against a position in r, without looking ahead in the string.

Example

- $(ab)^*$ is deterministic.
- $(ab)^*a$ is not deterministic. Examples: aba and a

Deterministic Regular Expressions

Deterministic regular expressions are ugly

- ~~Easy to use~~
- ~~Robust~~ class of languages: closed under ~~union~~, ~~intersection~~, ~~complement~~, ~~...~~
- ~~Very well~~ Partially understood

UPA Constraint

W3C XML Schema Standard

A content model must be formed such that during validation of an element information item sequence, the particle component contained directly, indirectly or implicitly therein with which to attempt to validate each item in the sequence in turn can be uniquely determined without examining the content or attributes of that item, and without any information about the items in the remainder of the sequence.

XML Schema Validator

Scenario

- User writes XML Schema Definition containing non-deterministic expression, say $(a+b)^*a$, and tries to validate it.
- Validator response: **ERROR: non-deterministic content model** $(a+b)^*a$.

Smart XML Schema Validator

Scenario

- User writes XML Schema Definition containing non-deterministic expression, say $(a+b)^*a$, and tries to validate it.
- Smart validator response: **PROBLEM: non-deterministic content model $(a+b)^*a$. However, the content model $b^*a(b^*a)^*$ describes the same content and is deterministic. Would you like to use it instead?**

Too optimistic ...

Theorem: Bruggemann-Klein and Wood

Some regular languages are not definable by a deterministic regular expression.

Too optimistic ...

Theorem: Bruggemann-Klein and Wood

Some regular languages are not definable by a deterministic regular expression.

Scenario

- User writes XML Schema Definition containing expression $(ab)^*a$ and tries to validate it.
- Smart validator response: **PROBLEM: non-deterministic content model for $(ab)^*a$. Moreover, there is no deterministic content model describing exactly this content. However, the content model $a(b?a)^*$ is deterministic and describes the same content plus some additional strings. Would you like to use it instead?**

Goal

Overall Goal

Develop the tools for a smart schema validator.

Technical goals

Given a non-deterministic regular expression,

- decide whether its language can be defined by a deterministic expression
- if possible, construct equivalent deterministic expression
- otherwise, construct deterministic overapproximation

Goal

Overall Goal

Develop the tools for a smart schema validator.

Technical goals

Given a non-deterministic regular expression,

- decide whether its language can be defined by a deterministic expression
- if possible, construct equivalent deterministic expression
- otherwise, construct deterministic overapproximation

Remark

All results apply to DTDs

Deciding Determinism

Deciding Determinism Problem

Given non-deterministic expression r, decide whether there exists a deterministic expression s, such that $L(r) = L(s)$.

Deciding Determinism

Deciding Determinism Problem

Given non-deterministic expression r, decide whether there exists a deterministic expression s, such that $L(r) = L(s)$.

Bruggemann-Klein and Wood 1998

Deciding Determinism can be done in time exponential in the size of r.

Deciding Determinism

Deciding Determinism Problem

Given non-deterministic expression r, decide whether there exists a deterministic expression s, such that $L(r) = L(s)$.

Bruggemann-Klein and Wood 1998

Deciding Determinism can be done in time exponential in the size of r.

Theorem

Deciding Determinism is PSPACE-hard.

Constructing Deterministic Expressions

Problem

Given a non-deterministic expression r, construct a deterministic expression s, such that $L(r) = L(s)$.

Construct Deterministic Expressions: BKW

Algorithm Bruggemann-Klein and Wood

- Construct minimal DFA.
- Construct deterministic expression by induction on DFA.
- Note: Added a few optimizations.

Construct Deterministic Expressions: BKW

Algorithm Bruggemann-Klein and Wood

- Construct minimal DFA.
- Construct deterministic expression by induction on DFA.
- Note: Added a few optimizations.

BKW

- + : If possible always return an equivalent deterministic expression.
- - : Can create very big expressions (possibly double exponential)

Example: $(a^*b?c?d?e?f^*g^*h^*i^*j^*k^*a^*)$

(. (* (. (a))) (| (| (. (d) (. (. (. (. (. (. (? (. (| (e) (f)) (* (. (f))))))) (? (. (g) (*
(. (g))))))) (? (. (h) (* (. (h))))))) (? (. (i) (* (. (i))))))) (? (. (j) (* (. (j)))))))
(? (. (k) (* (. (k))))))) (? (. (a) (* (. (a)))))))) (| (. (j) (. (. (* (. (j)))) (? (.
(k) (* (. (k))))))) (? (. (a) (* (. (a)))))))) (| (. (b) (. (. (. (. (. (. (. (. (? (. (c)
))) (? (. (d)))) (? (. (| (e) (f)) (* (. (f))))))) (? (. (g) (* (. (g))))))) (? (. (h)
(* (. (h))))))) (? (. (i) (* (. (i))))))) (? (. (j) (* (. (j))))))) (? (. (k) (* (. (k)
)))))) (? (. (a) (* (. (a)))))))) (| (. (g) (. (. (. (. (. (* (. (g)))) (? (. (h) (* (.
(h))))))) (? (. (i) (* (. (i))))))) (? (. (j) (* (. (j))))))) (? (. (k) (* (. (k)))))))
(? (. (a) (* (. (a)))))))) (| (. (e) (. (. (. (. (. (. (* (. (f)))) (? (. (g) (* (. (g)
)))))) (? (. (h) (* (. (h))))))) (? (. (i) (* (. (i))))))) (? (. (j) (* (. (j))))))) (? (.
(k) (* (. (k))))))) (? (. (a) (* (. (a)))))))) (| (. (c) (. (. (. (. (. (. (. (? (. (d)
))) (? (. (| (e) (f)) (* (. (f))))))) (? (. (g) (* (. (g))))))) (? (. (h) (* (. (h)))))))
(? (. (i) (* (. (i))))))) (? (. (j) (* (. (j))))))) (? (. (k) (* (. (k))))))) (? (. (a) (*
(. (a)))))))) (| (. (k) (. (* (. (k)))) (? (. (a) (* (. (a)))))))) (| (. (h) (. (. (. (.
(* (. (h)))) (? (. (i) (* (. (i))))))) (? (. (j) (* (. (j))))))) (? (. (k) (* (. (k)))))))
(? (. (a) (* (. (a)))))))) (| (. (f) (. (. (. (. (. (. (* (. (f)))) (? (. (g) (* (. (g)

Constructing Deterministic Expressions: GROW

Goal

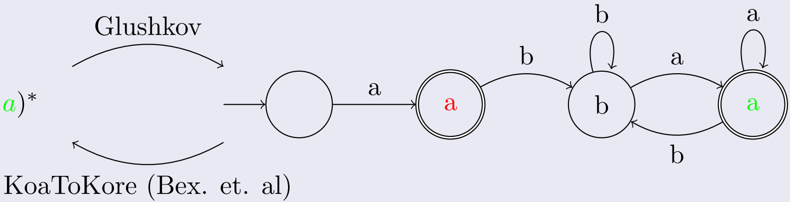
Find concise deterministic expressions.

Goal

Find concise deterministic expressions.

Glushkov Automata

$a(b^*a)^*$



Constructing Deterministic Expressions: GROW

Input Expression

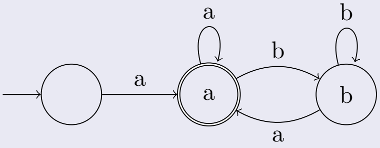
$a(a+b)^*a$

Constructing Deterministic Expressions: GROW

Input Expression

$a(a+b)^*a$

Minimal DFA



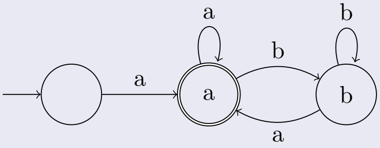
KoaToKore: Fail

Constructing Deterministic Expressions: GROW

Input Expression

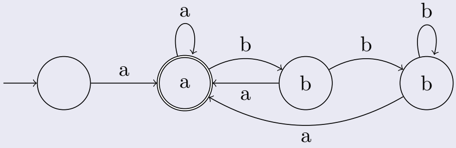
$a(a+b)^*a$

Minimal DFA



KoaToKore: Fail

Expansion 1



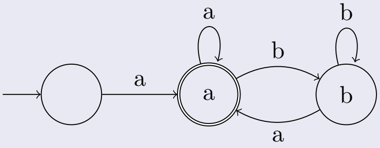
KoaToKore: Fail

Constructing Deterministic Expressions: GROW

Input Expression

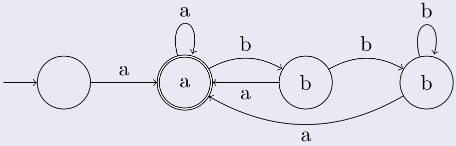
$a(a+b)^*a$

Minimal DFA



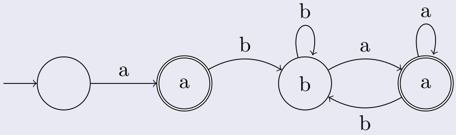
KoaToKore: Fail

Expansion 1



KoaToKore: Fail

Expansion 2



KoaToKore: $a(b^*a)^*$

Constructing Deterministic Expressions: GROW

Algorithm

- Enumerate all (non-isomorphic) deterministic automata equivalent to r, up to a given size.
- Check whether one of these automata is a Glushkov automaton; and construct equivalent expression.

Constructing Deterministic Expressions: GROW

Algorithm

- Enumerate all (non-isomorphic) deterministic automata equivalent to r, up to a given size.
- Check whether one of these automata is a Glushkov automaton; and construct equivalent expression.

GROW

- + : Returns concise, readable expressions.
- - : Not always returns an expression

Approximating Deterministic Expressions

Problem

Given a non-deterministic expression r, construct a deterministic expression s, such that $L(r) \subset L(s)$.

Approximating Deterministic Expressions

Problem

Given a non-deterministic expression r, construct a deterministic expression s, such that $L(r) \subset L(s)$.

Optimal Approximations

- An approximation s is optimal if there does not exist a deterministic expression s' such that $L(r) \subset L(s') \subset L(s)$.

Approximating Deterministic Expressions

Problem

Given a non-deterministic expression r, construct a deterministic expression s, such that $L(r) \subset L(s)$.

Optimal Approximations

- An approximation s is optimal if there does not exist a deterministic expression s' such that $L(r) \subset L(s') \subset L(s)$.

Theorem

Let r be an expression such that no equivalent deterministic expression exists. Then, there does not exist an optimal deterministic approximation of r.

Approximating Deterministic Expressions

Theorem

Let r be an expression such that no equivalent deterministic expression exists. Then, there does not exist an optimal deterministic approximation of r.

Proof

- Suppose s is optimal approximation of r.
- Take w in $L(s)$, not in $L(r)$
- $L(s) \setminus \{w\}$ also definable by deterministic expression s', but better approximation than s.

Approximating Deterministic Expressions: Ahonen

Algorithm by Ahonen: Ahonen-BKW

1. Given non-deterministic expression r, construct its minimal DFA.
2. "Simulate" BKW algorithm. Stuck $\Rightarrow$ merge states and add transitions.
3. Construct deterministic expression using BKW algorithm

Approximating Deterministic Expressions: Ahonen

Algorithm by Ahonen: Ahonen-BKW

1. Given non-deterministic expression r, construct its minimal DFA.
2. "Simulate" BKW algorithm. Stuck $\Rightarrow$ merge states and add transitions.
3. Construct deterministic expression using BKW algorithm

Ahonen-GROW

Alternative: apply GROW instead of BKW in step 3.

Approximating Deterministic Expressions: Ahonen

Ahonen-BKW

- + : Always returns an expression.
- - : Big expressions.

Ahonen-GROW

- + : Small expressions.
- - : Not always returns an expression

Approximating Deterministic Expressions: SHRINK

Goal

Algorithm that always returns small, readable expression.

Approximating Deterministic Expressions: SHRINK

Goal

Algorithm that always returns small, readable expression.

KoaToKore (Bex. et. al)

- When automaton is Glushkov automaton, returns corresponding expression (of equal size)
- Can also return overapproximation (of equal size)

Approximating Deterministic Expressions: SHRINK

Input Expression

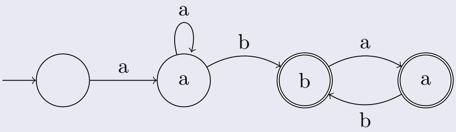
$a^+(ba)^*b?$

Approximating Deterministic Expressions: SHRINK

Input Expression

$a^{+}(ba)^{*}b?$

Minimal DFA



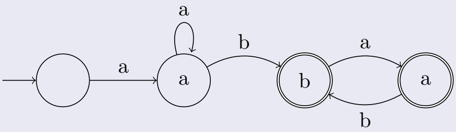
KoaToKore: Fail

Approximating Deterministic Expressions: SHRINK

Input Expression

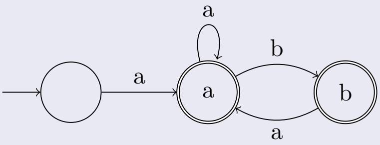
$a^{+}(ba)^{*}b?$

Minimal DFA



KoaToKore: Fail

Merged States

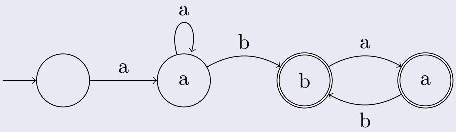


KoaToKore: $(ab?)^{+}$

Input Expression

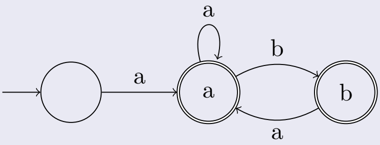
$a^+(ba)^*b?$

Minimal DFA



KoaToKore: Fail

Merged States



KoaToKore: $(ab?)^+$

Approximating Deterministic Expressions: SHRINK

Algorithm

- Shrink minimal DFA by merging states (trying to add as little as possible)
- Each DFA: check whether DFA is glushkov, or let koaToKore overapproximate (by adding transitions)

Experiments: Setup

Expressions

- Randomly generated.
- 2100 non-deterministic expressions.
- Number of alphabet symbols ranging from 5 to 50.

Experiments: Setup

Expressions

- Randomly generated.
- 2100 non-deterministic expressions.
- Number of alphabet symbols ranging from 5 to 50.

Repeatability and Workability

We participated in the ACM SIGMOD 2009 Repeatability and Workability Evaluation. The reviewers were able to repeat all the experiments presented in our paper, yielding results that match the ones published in our paper, except from insignificant and to be expected variation due to randomness and-or hardware-software differences. The detailed reports will shortly be made publicly available by ACM SIGMOD.

Experiments: Deciding Determinism

Deciding Determinism

- Very efficient (up to 50 milliseconds for largest ones)
- Minimal DFAs are small!

Experiments: Constructing Deterministic Expressions

Size of output expressions (and success rate)

input size	BKW	GROW
5	7	3 (89%)
10	95	6 (66%)
15	394	9 (43%)
20	/	12 (31%)
25-30	/	13 (21%)
35-50	/	23 (7%)

Running times

- GROW and BKW: Less than a second for small expressions.
- GROW: up to 20 seconds for biggest

Experiments: Approximating Deterministic Expressions

Measure of Quality

Ratio of number of strings defined by original expression over number by det. approximation: Close to 1 is good

Quality of Approximations

input size	Ahonen-BKW	Ahonen-GROW	SHRINK
5	0.73 (100%)	0.71 (75%)	0.75 (100%)
10	0.81 (100%)	0.79 (56%)	0.78 (100%)
15	0.84 (100%)	0.88 (40%)	0.79 (100%)
20	/	0.89 (18%)	0.76 (100%)
25-30	/	0.89 (8%)	0.71 (100%)
35-50	/	0.75 (4%)	0.68 (100%)

Experiments: Approximating Deterministic Expressions

Expression sizes (and success rate)

input size	Ahonen-BKW	Ahonen-GROW	SHRINK
5	8 (100%)	3 (75%)	3 (100%)
10	28 (100%)	6 (56%)	6 (100%)
15	73 (100%)	8 (40%)	8 (100%)
20	/	11 (18%)	10 (100%)
25-30	/	11 (8%)	13 (100%)
35-50	/	14 (4%)	18 (100%)

SUPAC

Supportive UPA Checker

Input regular expression

1. If r is deterministic, return r
2. Else If $L(r)$ is deterministic
   1. If GROW(r) succeeds, return GROW(r)
   2. Else return best from BKW(r) and SHRINK(r)
3. Else return best from Ahonen-GROW(r) and SHRINK(r)

Future and Current Work

Future and Current Work

- Minimization of deterministic expressions
- Experiments using real-world expressions
- Take into account counting operator