

# Inclusion and Membership for a Class of XML Types with Interleaving and Counting

Phd Lunchtime Seminar

Luca Pardini

University of Pisa

November 3, 2009



## Summary

Introduction

Types and constraints

Types

Constraints

Inclusion

Symmetric algorithm

Asymmetric algorithm

Structural algorithm

Experiments

Membership

Membership

XML types

Experiments

Conclusions

References

# Summary

## Introduction

### Types and constraints

Types

Constraints

### Inclusion

Symmetric algorithm

Asymmetric algorithm

Structural algorithm

Experiments

### Membership

Membership

XML types

Experiments

### Conclusions

### References

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

Interleaving and counting

$$(a \& b \& c \& d)$$

$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

Interleaving and counting

$$(a \& b \& c \& d)$$

$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

Interleaving and counting

$$(a \& b \& c \& d)$$

$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

Interleaving and counting

$$(a \& b \& c \& d)$$
$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

Interleaving and counting

$$(a \& b \& c \& d)$$
$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

### Interleaving and counting

$$(a\&b\&c\&d)$$
$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

### Interleaving and counting

$$(a\&b\&c\&d)$$
$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

## Introduction

XML is an important language used for sharing data on the web.  
XML types are needed to define the format of documents.

- ▶ inclusion  $T \subseteq U$ ;
- ▶ membership  $w \in T$ .

We reduce these problem on XML types to the same problem on Regular Expressions.

### Interleaving and counting

$$(a\&b\&c\&d)$$
$$a[3 : 5], b[1 : *]$$

The interleave operator increase the complexity of these algorithms!

# Summary

## Introduction

## Types and constraints

Types

Constraints

## Inclusion

Symmetric algorithm

Asymmetric algorithm

Structural algorithm

Experiments

## Membership

Membership

XML types

Experiments

## Conclusions

## References

# Types

## Interleaving

$$(ab)\&(XY) = \{abXY, aXbY, aXYb, XabY, XaYb, XYab\}$$

## Syntax

$$T ::= \epsilon \mid a \mid T[m..n] \mid T + T \mid T \cdot T \mid T\&T \mid T!$$

# Types

## Interleaving

$$(ab)\&(XY) = \{abXY, aXbY, aXYb, XabY, XaYb, XYab\}$$

## Syntax

$$T ::= \epsilon \mid a \mid T[m..n] \mid T + T \mid T \cdot T \mid T\&T \mid T!$$

# Types

## Symbol counting

A type  $T$  satisfy *symbol counting* if only symbols can be counted or subject to Kleene-star.

## Single occurrence

A type  $T$  is *single occurrence* if no symbol appears twice.

More than 95% of real-world XML Types satisfy these constraints.

# Types

## Symbol counting

A type  $T$  satisfy *symbol counting* if only symbols can be counted or subject to Kleene-star.

## Single occurrence

A type  $T$  is *single occurrence* if no symbol appears twice.

More than 95% of real-world XML Types satisfy these constraints.

# Types

## Symbol counting

A type  $T$  satisfy *symbol counting* if only symbols can be counted or subject to Kleene-star.

## Single occurrence

A type  $T$  is *single occurrence* if no symbol appears twice.

More than 95% of real-world XML Types satisfy these constraints.

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

### Syntax

$F ::= A^+$	$a \& b$	$\models$	$\{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c$	$\models$	$\{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3]$	$\models$	$a?[1..2]$
$\text{upper}(A)$	$a + b$	$\models$	$\text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b$	$\models$	$a \prec b$
$F \wedge F'$			
<b>true</b>			

$$a \prec \succ b = a \prec b \wedge b \prec a$$

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

### Syntax

$F ::= A^+$	$a \& b \models \{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c \models \{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3] \models a?[1..2]$
$\text{upper}(A)$	$a + b \models \text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b \models a \prec b$
$F \wedge F'$	
<b>true</b>	

$$a \prec \succ b = a \prec b \wedge b \prec a$$

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

### Syntax

$F ::= A^+$	$a \& b \models \{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c \models \{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3] \models a?[1..2]$
$\text{upper}(A)$	$a + b \models \text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b \models a \prec b$
$F \wedge F'$	
<b>true</b>	

$$a \prec \succ b = a \prec b \wedge b \prec a$$

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

### Syntax

$F ::= A^+$	$a \& b \models \{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c \models \{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3] \models a?[1..2]$
$\text{upper}(A)$	$a + b \models \text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b \models a \prec b$
$F \wedge F'$	
<b>true</b>	

$$a \prec\prec b = a \prec b \wedge b \prec a$$

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

### Syntax

$F ::= A^+$	$a \& b$	$\models$	$\{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c$	$\models$	$\{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3]$	$\models$	$a?[1..2]$
$\text{upper}(A)$	$a + b$	$\models$	$\text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b$	$\models$	$a \prec b$
$F \wedge F'$			
<b>true</b>			

$$a \prec \succ b = a \prec b \wedge b \prec a$$

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

### Syntax

$F ::= A^+$	$a \& b$	$\models$	$\{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c$	$\models$	$\{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3]$	$\models$	$a?[1..2]$
$\text{upper}(A)$	$a + b$	$\models$	$\text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b$	$\models$	$a \prec b$
$F \wedge F'$			
<b>true</b>			

$$a \prec\prec b = a \prec b \wedge b \prec a$$

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

### Syntax

$F ::= A^+$	$a \& b \models \{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c \models \{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3] \models a?[1..2]$
$\text{upper}(A)$	$a + b \models \text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b \models a \prec b$
$F \wedge F'$	
<b>true</b>	

$$a \prec\prec b = a \prec b \wedge b \prec a$$

## Constraints

From each type we extract a set of constraints  $F$  which it is satisfied by all words  $w$  of the type.

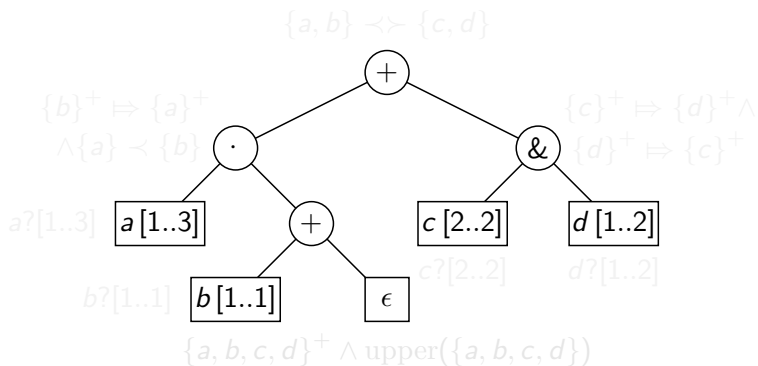
### Syntax

$F ::= A^+$	$a \& b \models \{a, b\}^+$
$A^+ \Rightarrow B^+$	$(a \& b) + c \models \{a\}^+ \Rightarrow \{b\}^+$
$a?[m..n]$	$a[1..2] + b[2..3] \models a?[1..2]$
$\text{upper}(A)$	$a + b \models \text{upper}(\{a, b\})$
$a \prec b$	$a + b + a \cdot b \models a \prec b$
$F \wedge F'$	
<b>true</b>	

$$a \prec\prec b = a \prec b \wedge b \prec a$$

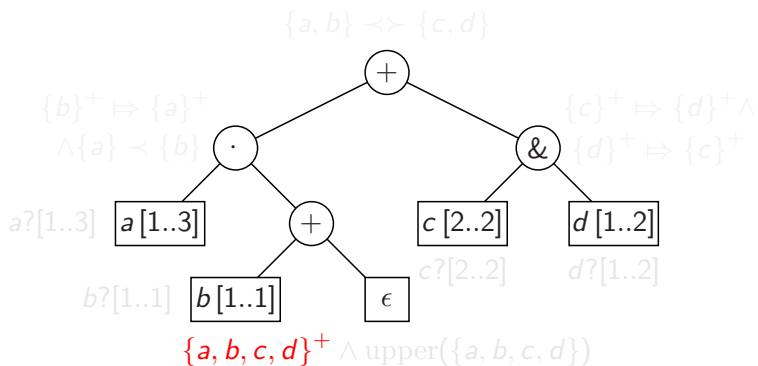
## Constraints extraction

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



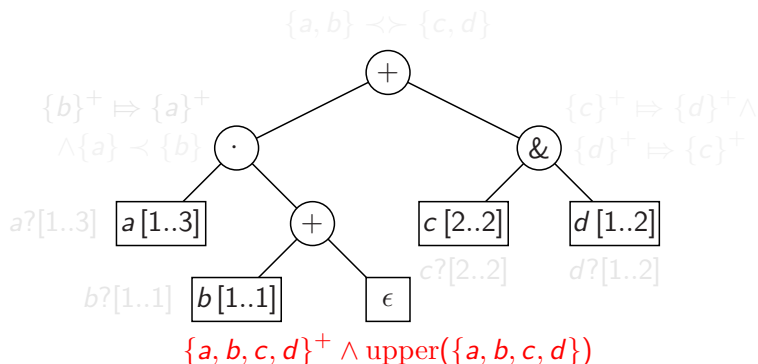
## Constraints extraction

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



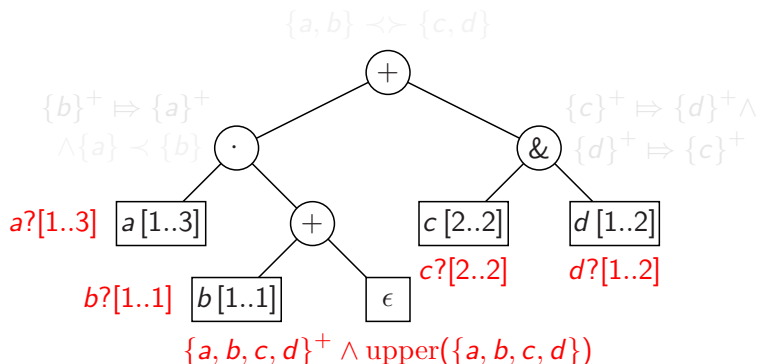
## Constraints extraction

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



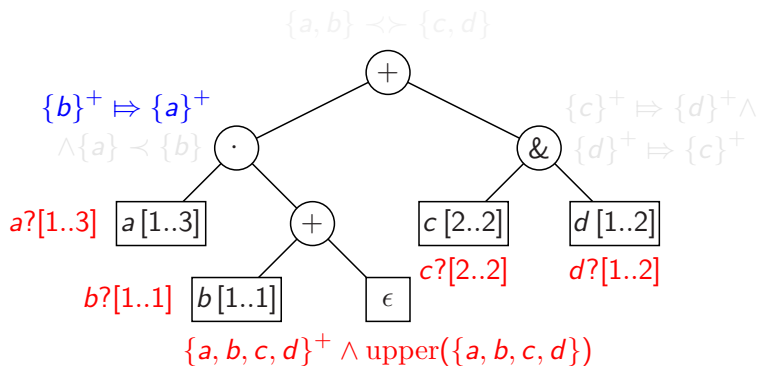
## Constraints extraction

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



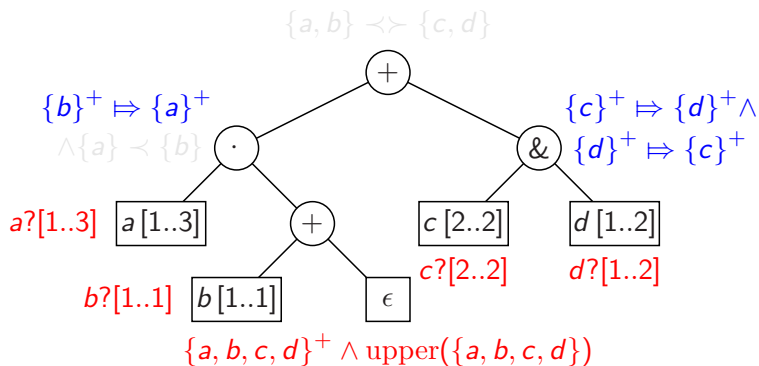
## Constraints extraction

$$T = (a [1..3] \cdot (b [1..1] + \epsilon)) + (c [2..2] \& d [1..2])$$



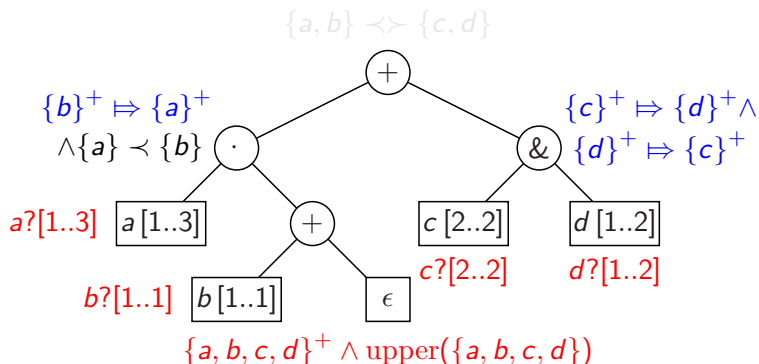
## Constraints extraction

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



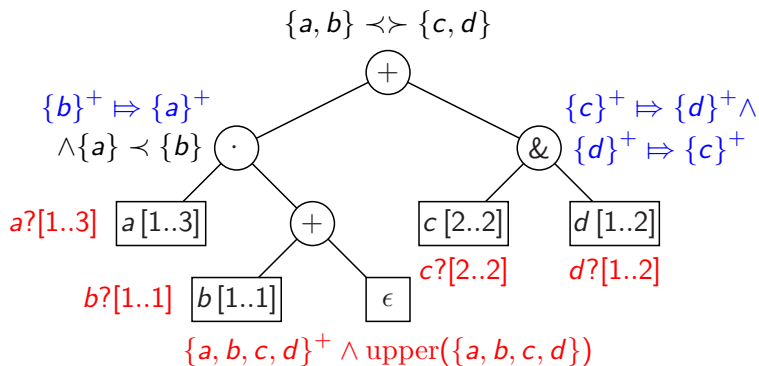
## Constraints extraction

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



## Constraints extraction

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



## Summary

Introduction

Types and constraints

Types

Constraints

**Inclusion**

Symmetric algorithm

Asymmetric algorithm

Structural algorithm

Experiments

Membership

Membership

XML types

Experiments

Conclusions

References

## The Idea

We want to check if  $T \subseteq U$  when both  $T$  and  $U$  satisfy symbol counting and single occurrence constraints.

### Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

We check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ;
- ▶  $T \models \mathcal{OC}(U)$ .

## The Idea

We want to check if  $T \subseteq U$  when both  $T$  and  $U$  satisfy symbol counting and single occurrence constraints.

### Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

We check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ;
- ▶  $T \models \mathcal{OC}(U)$ .

## The Idea

We want to check if  $T \subseteq U$  when both  $T$  and  $U$  satisfy symbol counting and single occurrence constraints.

### Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

We check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ;
- ▶  $T \models \mathcal{OC}(U)$ .

## Flat Constraints

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

atom	$Min_U$	$Max_U$
$a[1..3]$	1	3
$b[1..1]$	1	1
$c[2..2]$	2	2
$d[1..2]$	1	2

$T$  doesn't contain the empty string.

## Flat Constraints

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

atom	$Min_U$	$Max_U$
$a[1..3]$	1	3
$b[1..1]$	1	1
$c[2..2]$	2	2
$d[1..2]$	1	2

$T$  doesn't contain the empty string.

## Flat Constraints

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

atom	$Min_U$	$Max_U$
$a[1..3]$	1	3
$b[1..1]$	1	1
$c[2..2]$	2	2
$d[1..2]$	1	2

$T$  doesn't contain the empty string.

## Flat Constraints

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

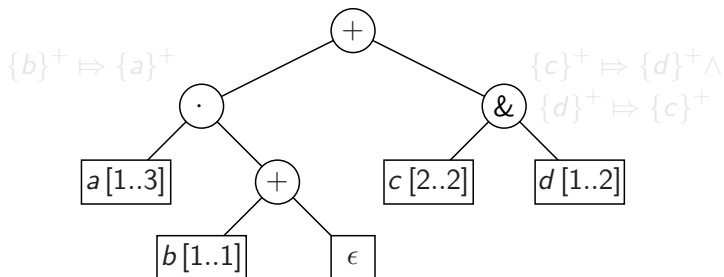
atom	$Min_U$	$Max_U$
$a[1..3]$	1	3
$b[1..1]$	1	1
$c[2..2]$	2	2
$d[1..2]$	1	2

$T$  doesn't contain the empty string.

## Co-Occurrence Constraints

$$T = c[2..2] \cdot d[1..1]$$

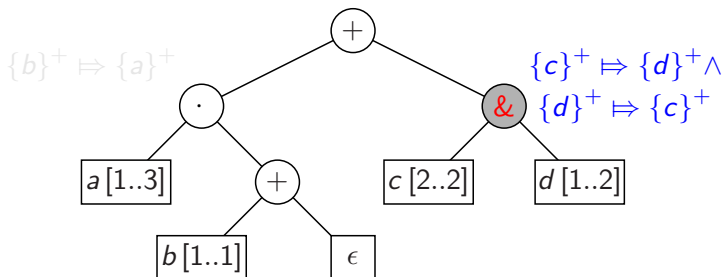
$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



## Co-Occurrence Constraints

$$T = c[2..2] \cdot d[1..1]$$

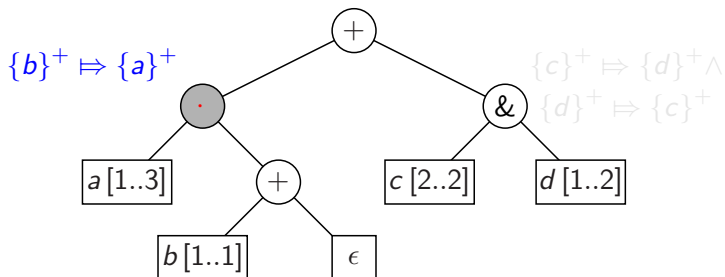
$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



## Co-Occurrence Constraints

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



## Order Constraints

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$

$a$	$b$	$LCA_U(a, b)$	$a \in S(T)$	$b \in S(T)$	$LCA_T(a, b)$
$a$	$b$	$\cdot$	<b>false</b>	<b>false</b>	
$a$	$c$	$+$	<b>false</b>	<b>true</b>	
$a$	$d$	$+$	<b>false</b>	<b>true</b>	
$b$	$c$	$+$	<b>false</b>	<b>true</b>	
$b$	$d$	$+$	<b>false</b>	<b>true</b>	
$c$	$d$	$\&$	<b>true</b>	<b>true</b>	$\cdot$

## Order Constraints

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$

$a$	$b$	$LCA_U(a, b)$	$a \in S(T)$	$b \in S(T)$	$LCA_T(a, b)$
$a$	$b$	$\cdot$	<b>false</b>	<b>false</b>	
$a$	$c$	$+$	<b>false</b>	<b>true</b>	
$a$	$d$	$+$	<b>false</b>	<b>true</b>	
$b$	$c$	$+$	<b>false</b>	<b>true</b>	
$b$	$d$	$+$	<b>false</b>	<b>true</b>	
$c$	$d$	$\&$	<b>true</b>	<b>true</b>	$\cdot$

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

As before we check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ;
- ▶  $T \models \mathcal{OC}(U)$ .

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

### Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

As before we check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ; (same)
- ▶  $T \models \mathcal{OC}(U)$ . (same)

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

### Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

As before we check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ; (same)
- ▶  $T \models \mathcal{OC}(U)$ . (same)

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

### Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

As before we check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ; (same)
- ▶  $T \models \mathcal{OC}(U)$ . (same)

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

### Inclusion

$$T \subseteq U \Leftrightarrow T \models \mathcal{FC}(U) \wedge T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U)$$

As before we check separately:

- ▶  $T \models \mathcal{FC}(U)$ ;
- ▶  $T \models \mathcal{CC}(U)$ ; (same)
- ▶  $T \models \mathcal{OC}(U)$ . (same)

## Flat Constraints

Without symbol counting on  $T$  we must calculate, for each symbol, the minimum and the maximum number of occurrence.

<i>Operator</i>	$T_1$	$T_2$	<i>Min</i>	<i>Max</i>
$T_1 \cdot T_2$	<b>true</b>	<b>true</b>	$Min_1 + Min_2$	$Max_1 + Max_2$
$T_1 \cdot T_2$	<b>true</b>	<b>false</b>	$Min_1$	$Max_1 + Max_2$
$T_1 \cdot T_2$	<b>false</b>	<b>true</b>	$Min_2$	$Max_1 + Max_2$
$T_1 \cdot T_2$	<b>false</b>	<b>false</b>	$\min(Min_1, Min_2)$	$Max_1 + Max_2$
$T_1 + T_2$			$\min(Min_1, Min_2)$	$\max(Max_1, Max_2)$
$T_1 [m..n]$	<b>true</b>		$Min_1 * m$	$Max_1 * n$
$T_1 [m..n]$	<b>false</b>		$Min_1$	$Max_1 * n$

## Flat Constraints

Without symbol counting on  $T$  we must calculate, for each symbol, the minimum and the maximum number of occurrence.

<i>Operator</i>	$T_1$	$T_2$	<i>Min</i>	<i>Max</i>
$T_1 \cdot T_2$	<b>true</b>	<b>true</b>	$Min_1 + Min_2$	$Max_1 + Max_2$
$T_1 \cdot T_2$	<b>true</b>	<b>false</b>	$Min_1$	$Max_1 + Max_2$
$T_1 \cdot T_2$	<b>false</b>	<b>true</b>	$Min_2$	$Max_1 + Max_2$
$T_1 \cdot T_2$	<b>false</b>	<b>false</b>	$\min(Min_1, Min_2)$	$Max_1 + Max_2$
$T_1 + T_2$			$\min(Min_1, Min_2)$	$\max(Max_1, Max_2)$
$T_1 [m..n]$	<b>true</b>		$Min_1 * m$	$Max_1 * n$
$T_1 [m..n]$	<b>false</b>		$Min_1$	$Max_1 * n$

## Motivations

A quadratic approach is too expensive in some cases of inclusion like these:

- ▶  $T \subseteq T$ ;
- ▶  $T \subseteq T[m..n]$ ;
- ▶  $T_1 \subseteq T_1 + T_2$ ;
- ▶  $T + T \subseteq T$ ;

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

We divide the problem using no-backtracking rules like this:

$$\text{conditions} \Rightarrow T_1 \cdot T_2 \subseteq U_1 \cdot U_2 \Leftrightarrow T_1 \subseteq U_1 \wedge T_2 \subseteq U_2$$

When no *conditions* are valid we use the asymmetric algorithm to compute the set of smaller problem.

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

We divide the problem using no-backtracking rules like this:

$$\text{conditions} \Rightarrow T_1 \cdot T_2 \subseteq U_1 \cdot U_2 \Leftrightarrow T_1 \subseteq U_1 \wedge T_2 \subseteq U_2$$

When no *conditions* are valid we use the asymmetric algorithm to compute the set of smaller problem.

## The Idea

We want to check if  $T \subseteq U$  when only  $U$  satisfy symbol counting and single occurrence constraints.

We divide the problem using no-backtracking rules like this:

$$\text{conditions} \Rightarrow T_1 \cdot T_2 \subseteq U_1 \cdot U_2 \Leftrightarrow T_1 \subseteq U_1 \wedge T_2 \subseteq U_2$$

When no *conditions* are valid we use the asymmetric algorithm to compute the set of smaller problem.

## Example

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq c[2..2] \&d[1..2]$$

$$c[2..2] \subseteq c[2..2]$$

$$d[1..1] \subseteq d[1..2]$$

## Example

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq c[2..2] \&d[1..2]$$

$$c[2..2] \subseteq c[2..2]$$

$$d[1..1] \subseteq d[1..2]$$

## Example

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq c[2..2] \&d[1..2]$$

$$c[2..2] \subseteq c[2..2]$$

$$d[1..1] \subseteq d[1..2]$$

## Example

$$T = c[2..2] \cdot d[1..1]$$

$$U = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \&d[1..2])$$

$$c[2..2] \cdot d[1..1] \subseteq c[2..2] \&d[1..2]$$

$$c[2..2] \subseteq c[2..2]$$

$$d[1..1] \subseteq d[1..2]$$

# Experiments

We want to see if our inclusion algorithms are efficient in practice.  
Our test sets are couples of types  $(T, U)$ .

## Sets

- ▶ random generated couples;
- ▶ random generated types  $(U)$  and random single modify  $(T)$ .

# Experiments

We want to see if our inclusion algorithms are efficient in practice.  
Our test sets are couples of types  $(T, U)$ .

## Sets

- ▶ random generated couples;
- ▶ random generated types ( $U$ ) and random single modify ( $T$ ).

# Experiments

We want to see if our inclusion algorithms are efficient in practice.  
Our test sets are couples of types  $(T, U)$ .

## Sets

- ▶ random generated couples;
- ▶ random generated types  $(U)$  and random single modify  $(T)$ .

## Random set

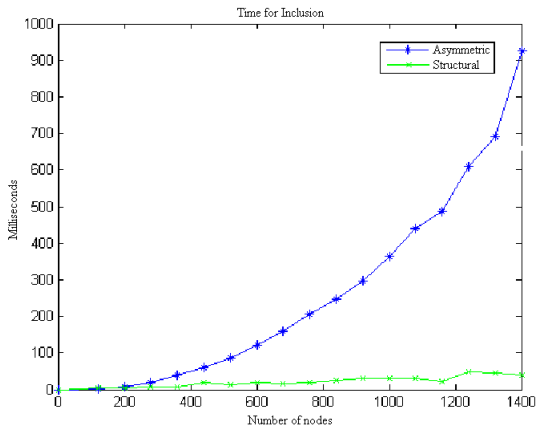


Figure: Algorithms time on random set.

## Random modify set

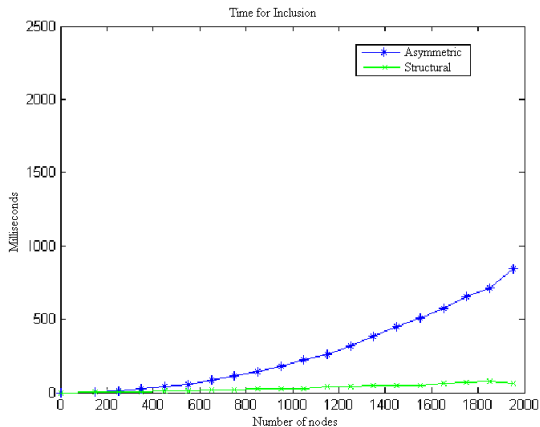


Figure: Algorithms time on random modify set.

## Summary

Introduction

Types and constraints

Types

Constraints

Inclusion

Symmetric algorithm

Asymmetric algorithm

Structural algorithm

Experiments

**Membership**

Membership

XML types

Experiments

Conclusions

References

## The Idea

We want to check if  $w \in T$  when  $T$  satisfy symbol counting and single occurrence constraints.

### Membership

$$w \in T \Leftrightarrow w \models F_T$$

We read each symbol  $a$  of the word  $w$  and we transform the co-occurrence and order constraints; at the end of this transformation we check the absence of any  $A^+$  constraint and the validity of cardinality constraints.

## The Idea

We want to check if  $w \in T$  when  $T$  satisfy symbol counting and single occurrence constraints.

### Membership

$$w \in T \Leftrightarrow w \models F_T$$

We read each symbol  $a$  of the word  $w$  and we transform the co-occurrence and order constraints; at the end of this transformation we check the absence of any  $A^+$  constraint and the validity of cardinality constraints.

## The Idea

We want to check if  $w \in T$  when  $T$  satisfy symbol counting and single occurrence constraints.

### Membership

$$w \in T \Leftrightarrow w \models F_T$$

We read each symbol  $a$  of the word  $w$  and we transform the co-occurrence and order constraints; at the end of this transformation we check the absence of any  $A^+$  constraint and the validity of cardinality constraints.

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a \ a \quad \{a, b\}^+ \Leftrightarrow \{c\}^+$   
 $a \ b \quad \{a\} \prec \{b\}$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a a c$       $\{a, b\}^+ \Leftrightarrow \{c\}^+$   
 $a b a$       $\{a\} \prec \{b\}$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a a c$      $\{c\}^+$   
 $a b a$      $\{a\} \prec \{b\}$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a a c$      $\{c\}^+$   
 $a b a$      $\{a\} \prec \{b\}$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a a c$       **true**

$a b a$        $\{a\} \prec \{b\}$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a a c$       **true**

$a b a$        $\{a\} \prec \{b\}$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a a c$       **true**

$a b a$        $\{a\} \prec \{b\}$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

### Examples

$a a c$       **true**

$a b a$        $\{a\}^-$

## Residuation

### Co-Occurrence constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A^+ \Leftrightarrow B^+$	$A^+ \Leftrightarrow B^+$	$A^+$
Residual	$B^+$	<b>true</b>	<b>true</b>

### Order constraints

Conditions	$a \in A$	$a \in B$	$a \in A$
Constraints	$A \prec B$	$A \prec B$	$A^-$
Residual	$A \prec B$	$A^-$	<b>false</b>

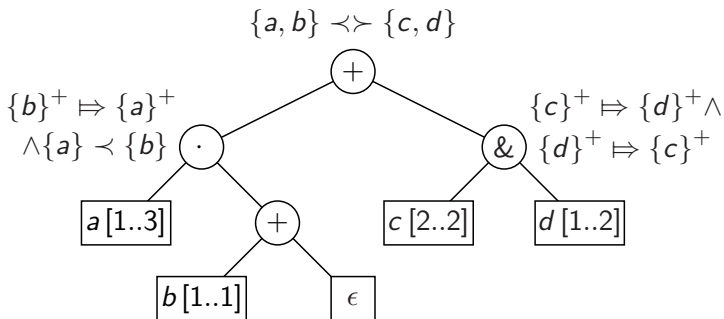
### Examples

$a a c$       **true**

$a b a$       **false**

## Linear algorithm

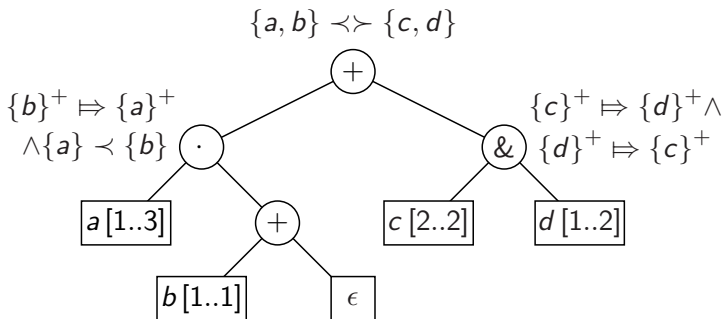
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b \$*

## Linear algorithm

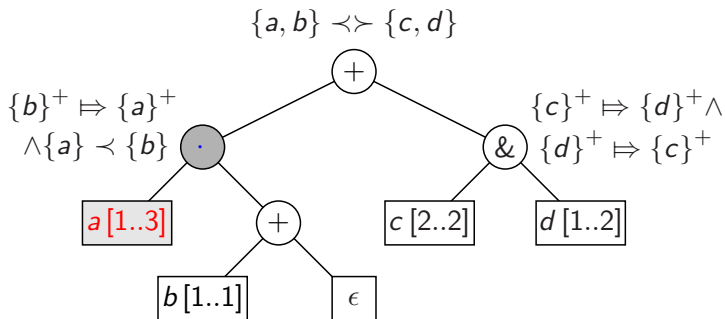
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b \$*

## Linear algorithm

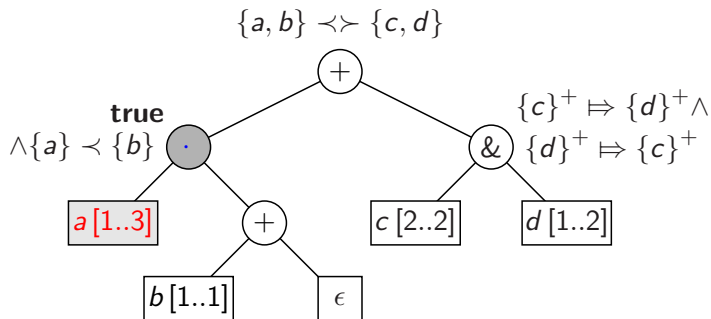
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a* a a b \$

## Linear algorithm

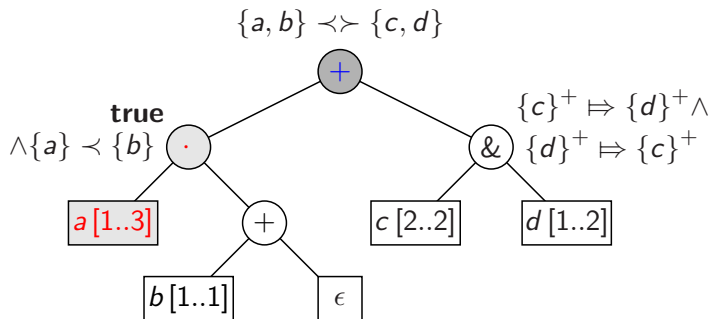
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b \$*

## Linear algorithm

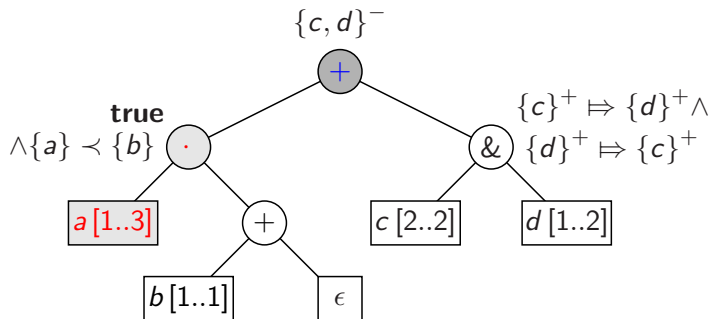
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



$a a a b \$$

## Linear algorithm

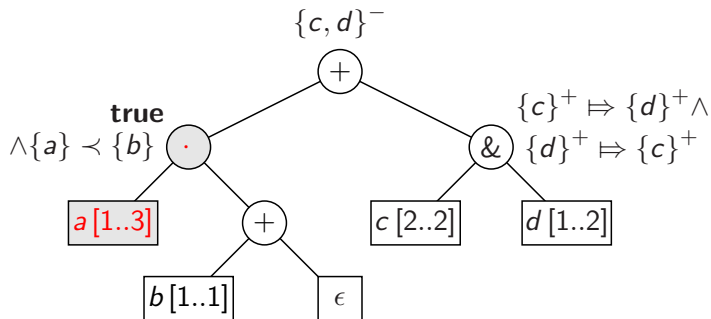
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b \$*

## Linear algorithm

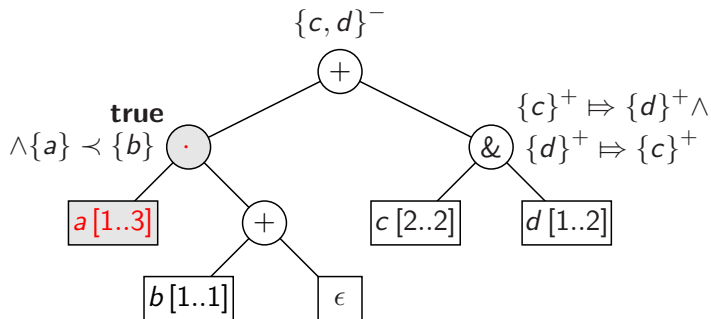
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



$a$   $a$   $a$   $b$   $\$$

## Linear algorithm

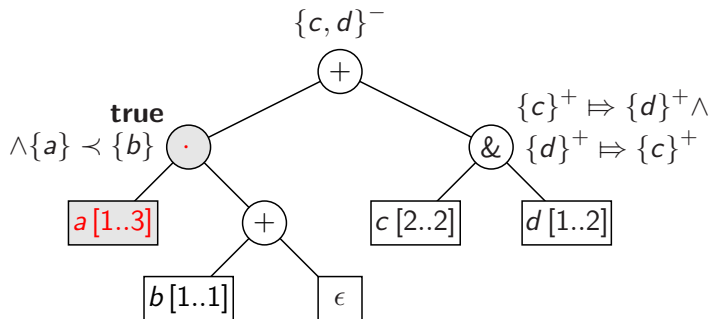
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



$a a a b \epsilon$

## Linear algorithm

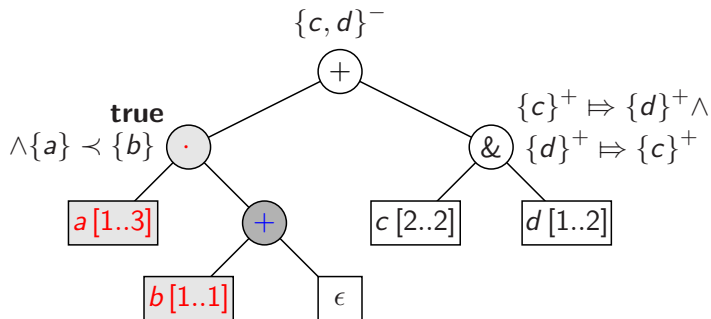
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b* \$

## Linear algorithm

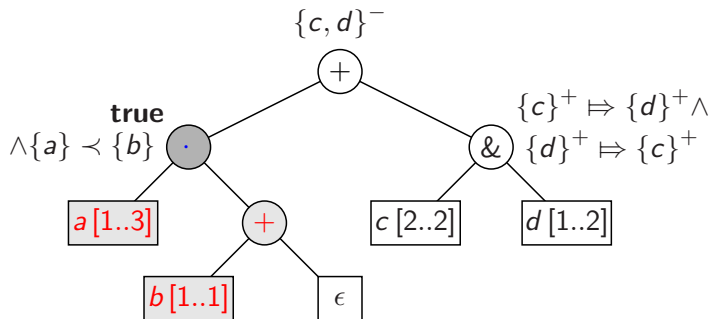
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



$a a a b \$$

## Linear algorithm

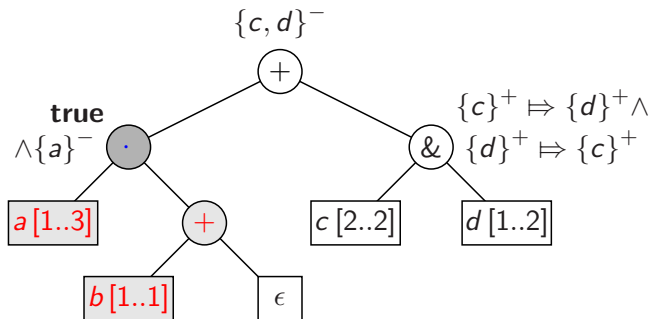
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b* \$

## Linear algorithm

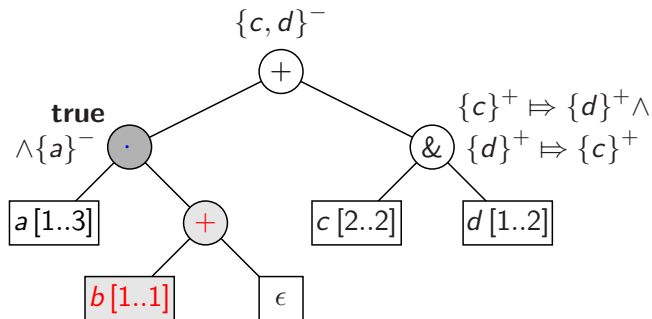
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b* \$

## Linear algorithm

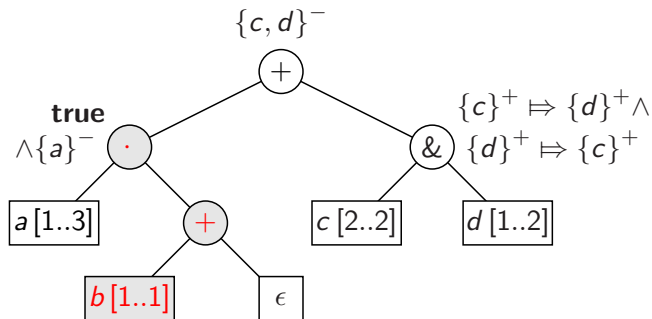
$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b* \$

## Linear algorithm

$$T = (a[1..3] \cdot (b[1..1] + \epsilon)) + (c[2..2] \& d[1..2])$$



*a a a b \$*

## The Idea

We want to check if  $w \in T$  where  $T$  is a XML type and  $w$  is an XML document.

The main idea is to consider the type a system of regular expressions and check membership by the handling 4 events:

- ▶ opening of the document (initialization);
- ▶ opening of an element;
- ▶ closing of an element;
- ▶ closing of the document;

## The Idea

We want to check if  $w \in T$  where  $T$  is a XML type and  $w$  is an XML document.

The main idea is to consider the type a system of regular expressions and check membership by the handling 4 events:

- ▶ opening of the document (initialization);
- ▶ opening of an element;
- ▶ closing of an element;
- ▶ closing of the document;

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

<i>c</i>	$\epsilon$
<i>a</i>	$\epsilon$
<i>T</i>	$\epsilon$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$a$	$\epsilon$
$T$	$\epsilon$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$a$	$\epsilon$
$T$	$\epsilon$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$a$	$\epsilon$
$T$	$\epsilon$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

c	ε
a	ε
T	ε
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$a$	$c$
$T$	$\epsilon$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

c	ε
a	c
T	ε
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

<i>c</i>	<i>ε</i>
<i>a</i>	<i>ε</i>
<i>T</i>	<i>aab</i>
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$a$	$\epsilon$
$T$	$aab$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$d$	$\epsilon$
$a$	$\epsilon$
$T$	$aab$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$d$	$\epsilon$
$a$	$\epsilon$
$T$	$aab$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$a$	$d$
$T$	$aab$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

c	ε
a	d
T	aab
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

<i>c</i>	<i>ε</i>
<i>a</i>	<i>ε</i>
<i>T</i>	<i>aa<b>b</b></i>
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$b$	$\epsilon$
$T$	$aab$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

$c$	$\epsilon$
$b$	$\epsilon$
$T$	$aab$
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

c	ε
a	ε
T	aab
Type	Word

## Lazy algorithm

$$\begin{aligned}
 T &= a[1..2] \& b[1..1] \\
 a &= c[1..1] + d[1..2] \\
 b &= e[1..1] + \epsilon \\
 c &= \epsilon \\
 d &= \epsilon \\
 e &= \epsilon
 \end{aligned}$$

$\langle T \rangle \langle a \rangle \langle c \rangle \langle /c \rangle \langle /a \rangle \langle a \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle \langle b \rangle \langle /b \rangle \langle /T \rangle$

c	ε
a	ε
T	aab
Type	Word

## Eager algorithm

This algorithm have the same behaviour of the Lazy algorithm, the differences are:

- ▶ it save the state of the transformed constraints (instead words);
- ▶ when it open an element it transform the constraints on the top of the stack;
- ▶ when it close an element it check if the constraints are satisfied;

## Eager algorithm

This algorithm have the same behaviour of the Lazy algorithm, the differences are:

- ▶ it save the state of the transformed constraints (instead words);
- ▶ when it open an element it transform the constraints on the top of the stack;
- ▶ when it close an element it check if the constraints are satisfied;

## Eager algorithm

This algorithm have the same behaviour of the Lazy algorithm, the differences are:

- ▶ it save the state of the transformed constraints (instead words);
- ▶ when it open an element it transform the constraints on the top of the stack;
- ▶ when it close an element it check if the constraints are satisfied;

## Eager algorithm

This algorithm have the same behaviour of the Lazy algorithm, the differences are:

- ▶ it save the state of the transformed constraints (instead words);
- ▶ when it open an element it transform the constraints on the top of the stack;
- ▶ when it close an element it check if the constraints are satisfied;

## Experiments

We want to see if our membership algorithms are efficient in practice.

Sets are composed by a type and lot of XML documents.

### Sets

- ▶ XMark;
- ▶ expense-report;
- ▶ DBLP.

## Experiments

We want to see if our membership algorithms are efficient in practice.

Sets are composed by a type and lot of XML documents.

### Sets

- ▶ XMark;
- ▶ expense-report;
- ▶ DBLP.

## Experiments

We want to see if our membership algorithms are efficient in practice.

Sets are composed by a type and lot of XML documents.

### Sets

- ▶ XMark;
- ▶ expense-report;
- ▶ DBLP.

## XMark

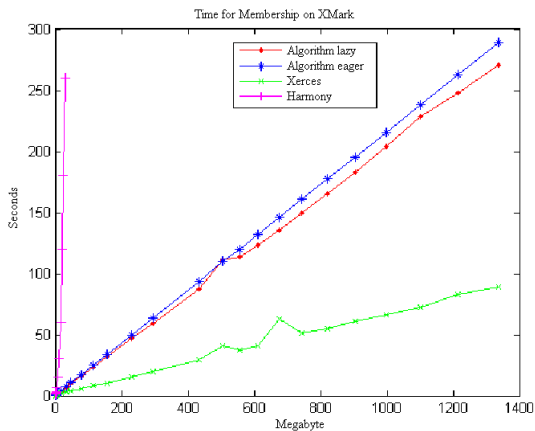


Figure: Algorithms time on XMark.

# Expense-report

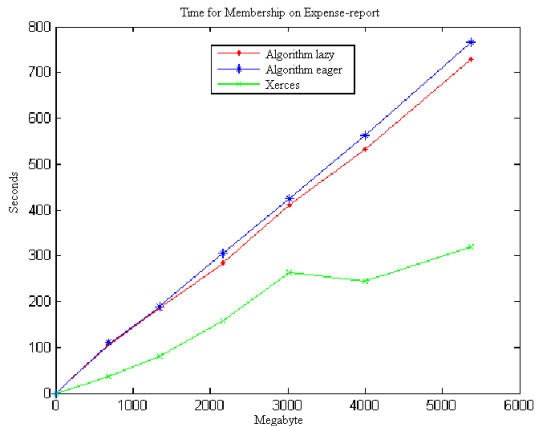


Figure: Algorithms time on expense-report.

## DBLP

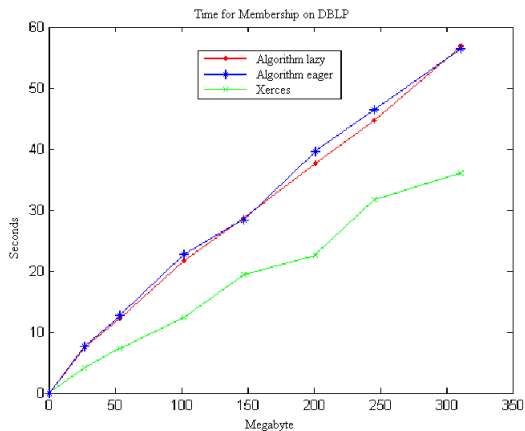


Figure: Algorithms time on DBLP.

## Summary

Introduction

Types and constraints

Types

Constraints

Inclusion

Symmetric algorithm

Asymmetric algorithm

Structural algorithm

Experiments

Membership

Membership

XML types

Experiments

Conclusions

References

## Conclusions

- ▶ We start by a problem expensive in general;
- ▶ We defined a subclass of regular expressions with interleaving which admit a polynomial inclusion and membership;
- ▶ We defined and implemented efficient algorithm.

## Conclusions

- ▶ We start by a problem expensive in general;
- ▶ We defined a subclass of regular expressions with interleaving which admit a polynomial inclusion and membership;
- ▶ We defined and implemented efficient algorithm.

## Conclusions

- ▶ We start by a problem expensive in general;
- ▶ We defined a subclass of regular expressions with interleaving which admit a polynomial inclusion and membership;
- ▶ We defined and implemented efficient algorithm.

## Summary

Introduction

Types and constraints

Types

Constraints

Inclusion

Symmetric algorithm

Asymmetric algorithm

Structural algorithm

Experiments

Membership

Membership

XML types

Experiments

Conclusions

References

## References



Dario Colazzo and Carlo Sartiani.

Efficient subtyping for unordered XML types.

Technical report, Dipartimento di Informatica - Università di Pisa, 2007.



Giorgio Ghelli, Dario Colazzo, and Carlo Sartiani.

Efficient Inclusion for a Class of XML Types with Interleaving and Counting.

In *DBPL*, 2007.



Giorgio Ghelli, Dario Colazzo, and Carlo Sartiani.

Linear Time Membership for a Class of XML Types with Interleaving and Counting.

In *PLAN-X*, 2008.



Wouter Gelade, Wim Martens, and Frank Neven.

Optimizing Schema Languages for XML: Numerical Constraints and Interleaving.  
2007.