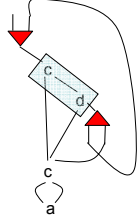


Compressed Trees

(2) Sharing Graphs [Lamping, POPL1990]

→ share identical *connected subgraphs* in a tree / DAG

e.g., on a path



Sharing Graphs

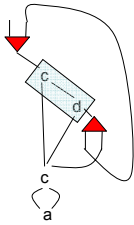
Given a tree t , **minimal** Sharing Graph

- is *not unique*
- to compute one is *NP-complete*
- at most *double-exp.* smaller than t
- equivalent to a minimal *context-free tree grammar* for $\{t\}$

Sharing Graphs

Given a tree t , **minimal** Sharing Graph

- is *not unique*
- to compute one is *NP-complete*
- at most *double-exp.* smaller than t
- equivalent to a minimal *context-free tree grammar* for $\{t\}$



$$S \rightarrow D(D(A))$$

$$D(y) \rightarrow c(A, d(A, y))$$

$$A \rightarrow c(B, B)$$

$$B \rightarrow a$$

Context-free tree grammar

Context-Free Tree Grammars

→ Generalize cf. grammars to trees

$$S \rightarrow a A B c C$$

Context-Free Tree Grammars

→ Generalize cf. grammars to trees

$$S \rightarrow a \overset{e}{A} B c C$$

g

Context-Free Tree Grammars

→ Generalize cf. grammars to trees

$$S \rightarrow a \overset{e}{A} B c C$$

g

$$C(Y_1, Y_2, Y_3) \rightarrow \begin{array}{c} \# \\ \begin{array}{ccc} A & A & A \\ | & | & | \\ B & B & Y_1 \\ | & | & | \\ Y_1 & Y_2 & \end{array} \end{array}$$

Context-Free Tree Grammars

→ Generalize cf. grammars to trees

$S \rightarrow a \overset{e}{A} B \overset{g}{C} C$

$p: C(y_1, y_2, y_3) \rightarrow$

Minimal Sharing Graph → Why not unique?

size 8/9

- (1) Share all c's and share all d's
- (2) Share c—d twice
- (3) Share c—d twice, and share c three times.

Minimal Sharing Graph → Why not unique?

size 8/9

- (1) Share all c's and share all d's
- (2) Share c—d twice
- (3) Share c—d twice, and share c three times.

(1)

Minimal Sharing Graph → Why not unique?

size 8/9

- (1) Share all c's and share all d's
- (2) Share c—d twice
- (3) Share c—d twice, and share c three times.

(1)

(2)

Minimal Sharing Graph → Why not unique?

size 8/9

- (1) Share all c's and share all d's
- (2) Share c—d twice
- (3) Share c—d twice, and share c three times.

(1)

(2)

(3)

Minimal Sharing Graph (= cf tree grammar)

$S \rightarrow$

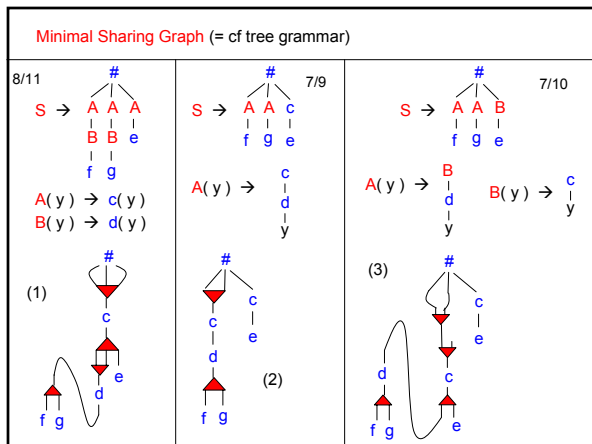
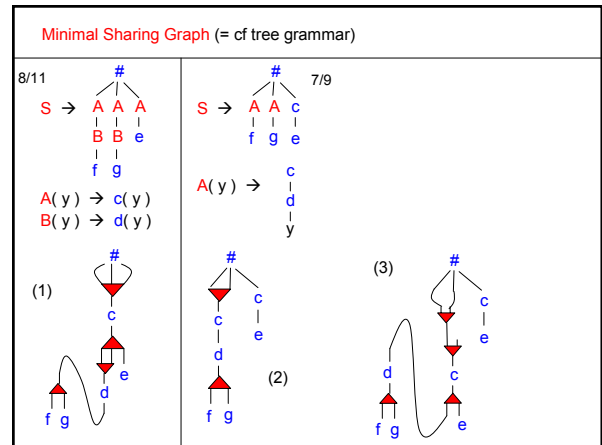
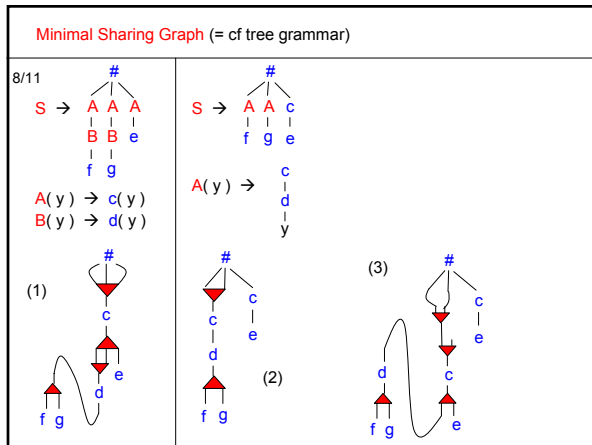
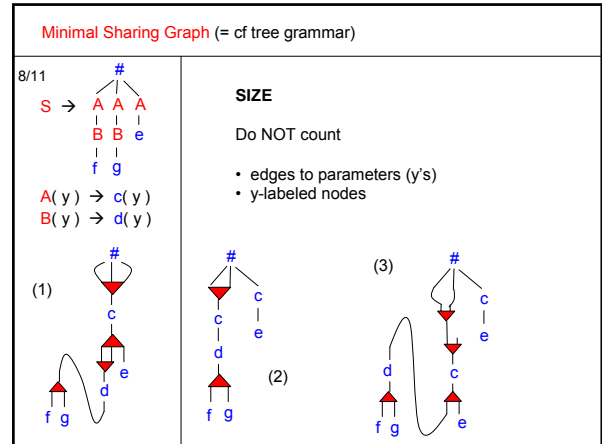
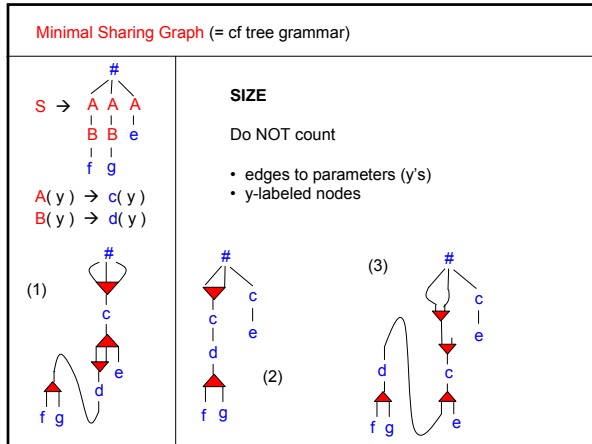
$A(y) \rightarrow c(y)$

$B(y) \rightarrow d(y)$

(1)

(2)

(3)



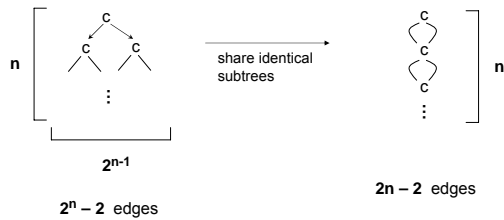
Sharing Graphs

Why **double-exp.** smaller?

Sharing Graphs

Why **double-exp.** smaller?

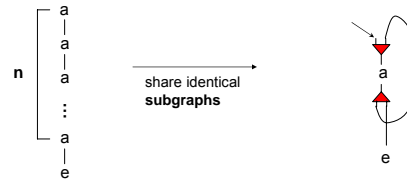
1. DAG can be at most exp. smaller (width)



Sharing Graphs

Why **double-exp.** smaller?

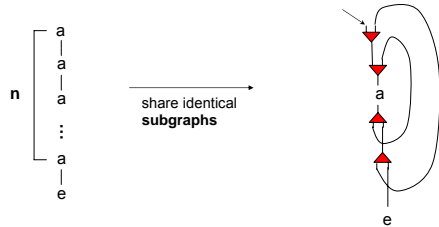
2. Sgraph can be at most exp. smaller in height



Sharing Graphs

Why **double-exp.** smaller?

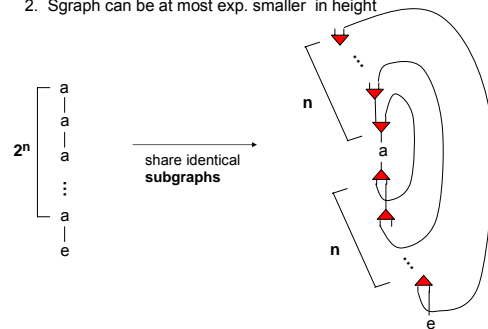
2. Sgraph can be at most exp. smaller in height



Sharing Graphs

Why **double-exp.** smaller?

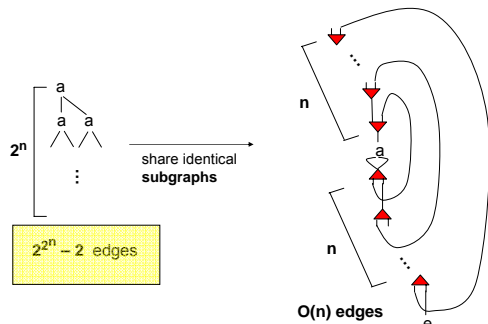
2. Sgraph can be at most exp. smaller in height



Sharing Graphs

Why **double-exp.** smaller?

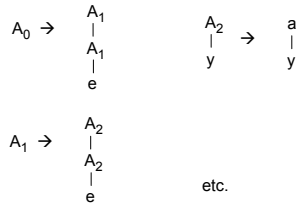
Cf tree grammar??



SL Grammars



SL Grammars



Def. A grammar is **Straight-Line (SL)**, if there is a linear order A_1, \dots, A_n of its nonterminals such that for every production $A_k \rightarrow \text{rhs}$, all nonterminals in rhs have index $j > k$.

Compressed Trees

- stronger compression ↓
- (1) DAG
 - (2) Linear + fixed#param SL grammar
 - (3) fixed#param SL grammar
 - (4) **Straight-Line of tree grammar (SL grammar)** = sharing graph

$$A_1 \rightarrow A_2(A_3, a, A_3)$$

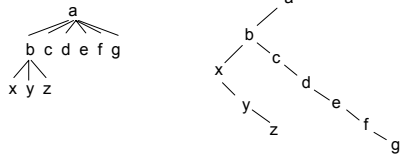
$$A_2(y_1, y_2, y_3) \rightarrow A_5(y_1, y_2, A_6(y_3, y_3))$$

Compressing XML skeletons

→ **DAGs** common XML tree skeletons compress to $\approx 10\%$ of original size [Buneman/Grohe/Koch,VLDB'03]

→ **linear, fixed#param SL grammar** common XML tree skeletons compress to $< 5\%$ of original size [Busatto/Lohrey/Maneth,DBPL'05]

BPLEX = lin. time algorithm to find small linear SL grammar
→ works on *binary encodings* of XML skeletons



BPLEX

BPLEX = linear time algorithm to find small linear SL grammar
"Bottom-up Binary PLEXing"

Input an SL (regular) tree grammar
Output a (smaller) SL of tree grammar

Idea go BU through rhs's, searching for duplicate patterns that do not overlap.

- Only consider patterns of size $< \text{MAX_PSI_ZE}$
- Only consider patterns of rank $< \text{MAX_RANK}$
- Only search the last WIN_SI_ZE nodes/productions

BPLEX

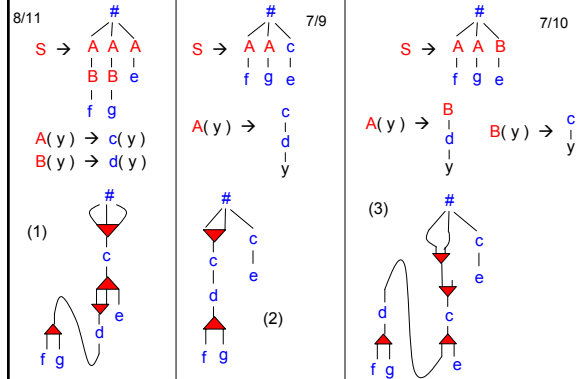
```

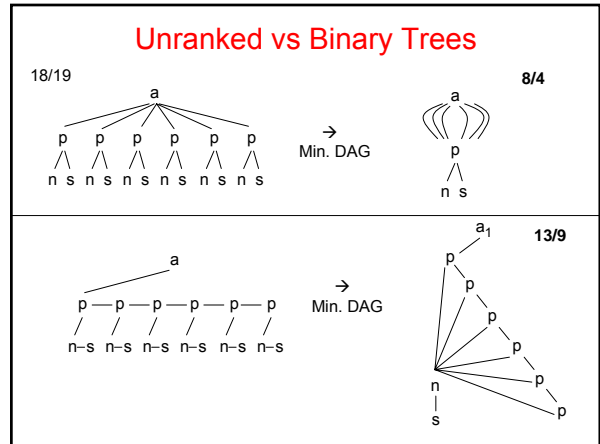
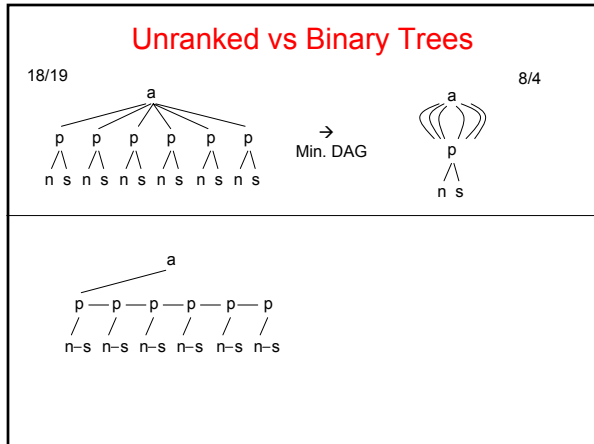
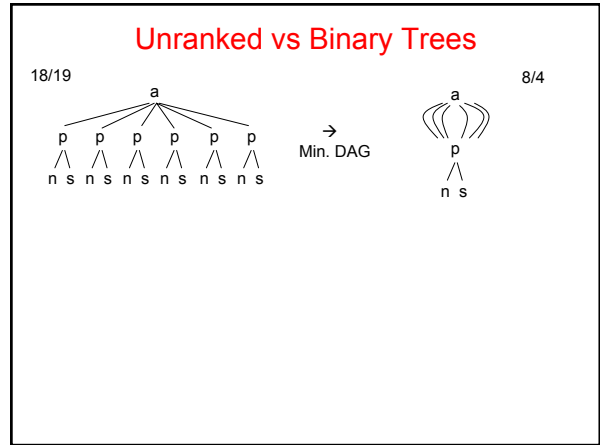
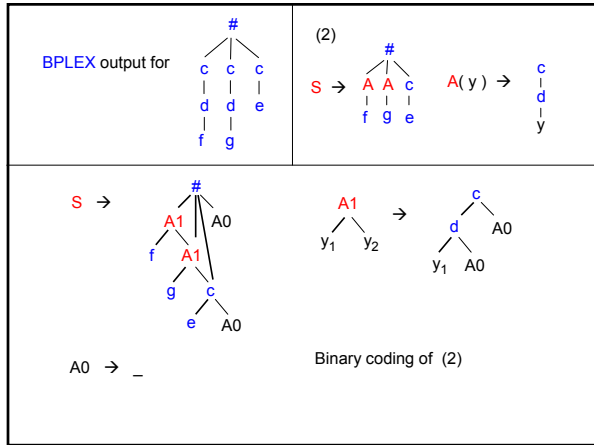
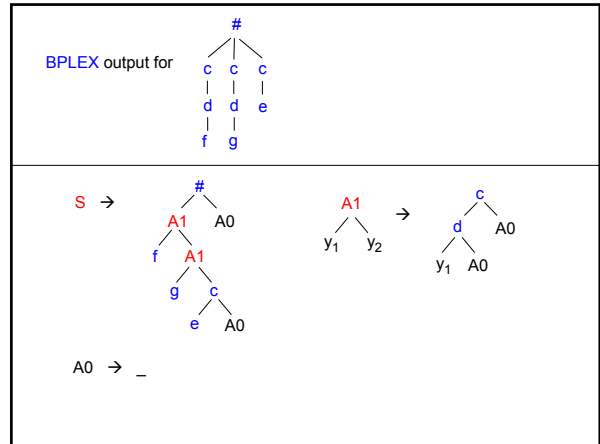
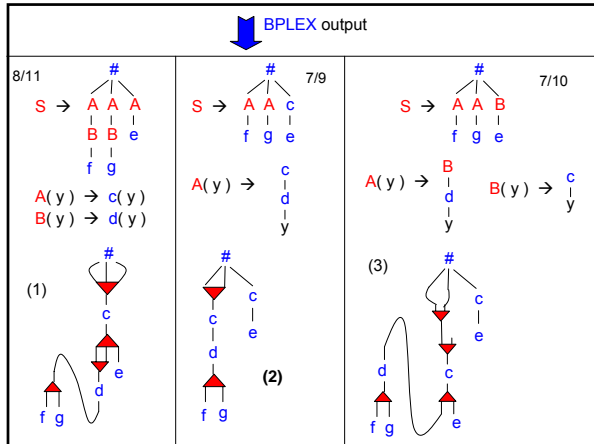
procedure BPLEX(G: grammar, KN: int, KS: int, KR: int): grammar
begin
  A := last symbol in the SL ordering of RG
  z := leftmost leaf of rhsG(A)
  while true do
    repM := RepeatedMatches(G, z, KN)
    newM := NewMatches(G, z, KN, KS, KR)
    if newM != ∅ or repM != ∅ then
      m := max(newM, repM)
      if m in newM then
        G := G[m ← A], with rhsG(A) = pm
      else
        k := rank(pm)
        A := fresh(G, k)
        G := add(G, A(y1, . . . , yk) → pm)
        G := G[m, cm ← A]
      if
        el sel f ∃ exists w in V: z < w then
          z := next(<_G, z)
        el se break
    fi
  od
  return G
end BPLEX
  
```

Replace pattern by NonTerminal

Replace pattern by new NT and add production

Minimal Sharing Graph (= cf tree grammar)





Unranked vs Binary Trees

18/19 3/4

Min. DAG 6

Min. DAG 13/9

Unranked vs Binary Trees

18/19 3/4

Min. DAG 6

Can it be vica versa? (min bin. DAG is smaller)

Unranked vs Binary Trees

18/19 3/4

Min. DAG 6

Can it be vica versa? (min bin. DAG is smaller)

YES!!

→ Has **18 edges**

→ Binary coding has only **12 edges**

x b c b c y b c b c z b c b c

Unranked vs Binary Trees

18/19 3/4

Min. DAG 6

Can it be vica versa? (min bin. DAG is smaller)

YES!!

→ Has **18 edges**

→ Binary coding has only **12 edges**

x b c b c y b c b c z b c b c

DAG compression is sensible to rank/unrankedness!

Compressing XML skeletons

input file	size of tree	min. binary DAG size	min. unranked mDAG size	BPLEX output size
SwissProt (457.4 MB)	10,903,568	1,437,445	1,100,648	311,328
DBLP (103.6 MB)	2,611,931	533,183	222,754	116,902
Treebank (55.9 MB)	2,447,727	1,454,494	1,301,898	519,542
1999statistics (657 KB)	28,306	2,403	726	410
catalog-02 (104M)	2,240,231	52,392	32,267	26,774
catalog-01 (11M)	225,194	6,990	8,503	3,817
dictionary-02 (104M)	2,731,764	681,130	441,322	160,329
dictionary-01 (11M)	277,072	77,554	46,993	20,150
JST_snp.chr1 (36M)	655,946	40,663	25,047	12,856
JST_gene.chr1 (11M)	216,401	14,606	6,658	4,000
NCBI_snp.chr1 (190M)	3,642,225	809,394	15	59
NCBI_gene.chr1 (24M)	360,350	14,356	11,767	7,160

→ BPLEX on unranked trees gives (almost) same as on bin. trees.

NOT sensible to rank/unrankedness!

Algorithms on Compressed Trees

In general:

more compression → more access overhead

Worst case [Busatto/Maneth,FOSSACS'04] for an SL Grammar G

|G| overhead per edge traversal
(for any algorithm that needs read access)

Algorithms on Compressed Trees

For **particular problems**,
is there *less overhead* than the worst case one?

YES!!

Problem (A) Type Validation
(B) Equivalence Test

(A) Type Validation

XML Type Languages

- (1) DTDs (originated from SGML - 1974)
- (2) XML Schema (W3C, recently)
- (3) Relax NG (Oasis, recently)

In terms of (unranked) tree languages

- (1) local
- (2) deterministic top-down
- (3) regular

DTD \subseteq Schemas \subseteq Relax

→ DTD/Schema validation in poly time w.r.t. size of grammar!

Algorithms on Compressed Trees

(A) Type Validation

XML Type described by a **Tree Automaton (TA)**

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $\text{eval}(G) \in T(A)$,
And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

Idea: Run the **TA** on the rhs's of the **SL of tree grammar**.

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $\text{eval}(G) \in T(A)$,
And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

G S1 → S2(S2(a)) A δ(a) = q
S2(y) → S3(S3(y)) δ(g(q, q)) = q'
S3(y) → S4(S4(y)) δ(g(q', q')) = q
S4(y) → g(y, y)

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $\text{eval}(G) \in T(A)$,
And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

G S1 → S2(S2(a)) A δ(a) = q
S2(y) → S3(S3(y)) δ(g(q, q)) = q'
S3(y) → S4(S4(y)) δ(g(q', q')) = q
S4(y) → g(y, y)

Run A on rhs(S4) = g(y, y).
→ function σ_{S4} from {q, q'} to {q, q'}

$\sigma_{S4}(q) = q'$
 $\sigma_{S4}(q') = q$

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $\text{eval}(G) \in T(A)$,
And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

G S1 → S2(S2(a)) A δ(a) = q
S2(y) → S3(S3(y)) δ(g(q, q)) = q'
S3(y) → S4(S4(y)) δ(g(q', q')) = q
S4(y) → g(y, y)

Run A on rhs(S4) = g(y, y).
→ function σ_{S4} from {q, q'} to {q, q'}

$\sigma_{S4}(q) = q'$ $\sigma_{S3}(q) = \sigma_{S4}(\sigma_{S4}(q)) = q$
 $\sigma_{S4}(q') = q$

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $eval(G) \in T(A)$,
 And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

G $S1 \rightarrow S2(S2(a))$ A $\delta(a) = q$
 $S2(y) \rightarrow S3(S3(y))$ $\delta(g(q, q)) = q'$
 $S3(y) \rightarrow S4(S4(y))$ $\delta(g(q', q')) = q$
 $S4(y) \rightarrow g(y, y)$

Run A on $rhs(S4) = g(y, y)$.
 \rightarrow function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

$\sigma_{S4}(q) = q'$ $\sigma_{S3}(q) = \sigma_{S4}(\sigma_{S4}(q)) = q$
 $\sigma_{S4}(q') = q$ $\sigma_{S3}(q') = \sigma_{S4}(\sigma_{S4}(q')) = q'$

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $eval(G) \in T(A)$,
 And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

G $S1 \rightarrow S2(S2(a))$ A $\delta(a) = q$
 $S2(y) \rightarrow S3(S3(y))$ $\delta(g(q, q)) = q'$
 $S3(y) \rightarrow S4(S4(y))$ $\delta(g(q', q')) = q$
 $S4(y) \rightarrow g(y, y)$

Run A on $rhs(S4) = g(y, y)$.
 \rightarrow function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

$\sigma_{S4}(q) = q'$ $\sigma_{S3}(q) = \sigma_{S4}(\sigma_{S4}(q)) = q$ $\sigma_{S2}(q) = q$
 $\sigma_{S4}(q') = q$ $\sigma_{S3}(q') = \sigma_{S4}(\sigma_{S4}(q')) = q'$ $\sigma_{S2}(q') = q'$

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $eval(G) \in T(A)$,
 And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

G $S1 \rightarrow S2(S2(a))$ A $\delta(a) = q$
 $S2(y) \rightarrow S3(S3(y))$ $\delta(g(q, q)) = q'$
 $S3(y) \rightarrow S4(S4(y))$ $\delta(g(q', q')) = q$
 $S4(y) \rightarrow g(y, y)$

Run A on $rhs(S4) = g(y, y)$.
 \rightarrow function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

$\sigma_{S4}(q) = q'$ $\sigma_{S3}(q) = \sigma_{S4}(\sigma_{S4}(q)) = q$ $\sigma_{S2}(q) = q$
 $\sigma_{S4}(q') = q$ $\sigma_{S3}(q') = \sigma_{S4}(\sigma_{S4}(q')) = q'$ $\sigma_{S2}(q') = q'$

$\sigma_{S1} = \sigma_{S2}(\sigma_{S2}(\delta(a))) = q$

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
TA A with n states

can check in **time** $O(n^{k+1} |G| |A|)$ whether $eval(G) \in T(A)$,
 And **time** $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

For each nonterminal N,
 n^k many values of σ_N are computed

\rightarrow At most $n^k |G|$ computation steps (runs of A). ■

Run A on $rhs(S4) = g(y, y)$.
 \rightarrow function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

$\sigma_{S4}(q) = q'$ $\sigma_{S3}(q) = \sigma_{S4}(\sigma_{S4}(q)) = q$ $\sigma_{S2}(q) = q$
 $\sigma_{S4}(q') = q$ $\sigma_{S3}(q') = \sigma_{S4}(\sigma_{S4}(q')) = q'$ $\sigma_{S2}(q') = q'$

$\sigma_{S1} = \sigma_{S2}(\sigma_{S2}(\delta(a))) = q$

Complexity Results [Lohrey/Maneth, CIAA'05]

		det. TDLA	det. BUTA	TA
uncompressed trees [13]	fixed	NC ¹ -complete		
	uniform	L-complete	LOGDCFL, L-hard	LOGCFL- complete
dags	fixed	NL-complete	P-complete	
	uniform			
lin. SL + fixed number para.	fixed	P-complete		
	uniform			
SL + fixed number para.	fixed	P-complete		PSPACE- complete
	uniform			
unrestricted SL	fixed	P-complete	PSPACE-complete	
	uniform			

Complexity Results [Frick/Grohe/Koch, LICS'03] = [1] [Lohrey/Maneth, CIAA'05] = [2]

		det. TDLA	det. BUTA	TA	Core XPath
uncompressed trees [13]	fixed	NC ¹ -complete			
	uniform	L-complete	LOGDCFL, L-hard	LOGCFL- complete	
dags	fixed	NL-complete	P-complete		PSPACE- Complete[1]
	uniform				
lin. SL + fixed number para.	fixed	P-complete			PSPACE- Complete[2]
	uniform				
SL + fixed number para.	fixed	P-complete		PSPACE- complete	
	uniform				
unrestricted SL	fixed	P-complete	PSPACE-complete		
	uniform				

Theorem [Busatto/Lohrey/Maneth,DBPL'05] Testing equivalence of two SL of tree grammars G_1, G_2 can be done in PSPACE, and in polynomial time if G_1 and G_2 are linear.

→ Change G_1, G_2 such that

- (1) each parameter y_i appears exactly once in every rhs
- (2) order of params in every rhs is y_1, y_2, \dots, y_k

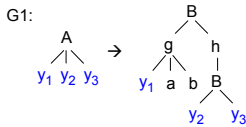
→ Construct cf (string) grammars H_1, H_2 generating depth-first left-to-right traversals of $\text{tree}(G_1), \text{tree}(G_2)$

For A of G_1 with rank $k > 0$,
 H_1 has nonterminals $A_{0,1}, A_{1,2}, \dots, A_{k-1,k}, A_{k,0}$

Theorem [Busatto/Lohrey/Maneth,DBPL'05] Testing equivalence of two SL of tree grammars G_1, G_2 can be done in PSPACE, and in polynomial time if G_1 and G_2 are linear.

→ Construct cf (string) grammars H_1, H_2 generating depth-first left-to-right traversals of $\text{tree}(G_1), \text{tree}(G_2)$

For A of G_1 with rank $k > 0$,
 H_1 has nonterminals $A_{0,1}, A_{1,2}, \dots, A_{k-1,k}, A_{k,0}$

G1:  H_1 : $A_{1,2} \rightarrow g_{1,2} a g_{2,3} b g_{3,0} B_{1,2} h_{0,1} B_{0,1}$

Theorem [Busatto/Lohrey/Maneth,DBPL'05] Testing equivalence of two SL of tree grammars G_1, G_2 can be done in PSPACE, and in polynomial time if G_1 and G_2 are linear.

→ Construct cf (string) grammars H_1, H_2 generating depth-first left-to-right traversals of $\text{tree}(G_1), \text{tree}(G_2)$

For A of G_1 with rank $k > 0$,
 H_1 has nonterminals $A_{0,1}, A_{1,2}, \dots, A_{k-1,k}, A_{k,0}$

→ G_1 equiv. G_2 iff H_1 equiv H_2
 (and, size of H_1, H_2 is poly in size of G_1, G_2)

By [Plandowski,ESA'94] equivalence of H_1, H_2 can be Decided in poly time wrt sizes of H_1, H_2 . ■

Conclusions

SL cf. tree grammars can represent XML documents more space efficiently than DAGs!

- (1) XML type validation and
- (2) (core) Xpath evaluation

have same complexity bounds on lin. SL grammars as on DAGs!

Open

- How big are efficiency gains in practice? (full queries on full XML docus)
- Succinct Representation of SL grammars?!
- XPath on non-linear SL grammars. In PSPACE?
- Equivalence of non-linear SL grammars. In PTIME?

