

Grammar-Based Tree Compression

Sebastian Maneth
NICTA & UNSW

Joint work with: Giorgio Busatto (Universität Oldenburg)
and Markus Lohrey (Universität Stuttgart)

Dagstuhl Seminar on
Structure-Based Compression of Complex Massive Data
26. June 2008

Outline

- Compressed Trees: DAGs & SL Grammars
- Generating SL Grammars with BPLEX
- Unranked vs. Binary Trees
- Algorithms on SL Grammars
 - Type Validation
 - Equivalence Test

What we did:

- Find a **small, pointer-based tree representation**, and
- an **algorithm (BPLEX) generating it** from a given tree.
- We tested **BPLEX** on XML tree structures of common (big) XML documents. (→ **performs well!**)

What we did:

- Find a **small, pointer-based tree representation**, and
 - an **algorithm (BPLEX) generating it** from a given tree.
 - We tested **BPLEX** on XML tree structures of common (big) XML documents. (→ **performs well!**)
-

What we have NOT done:

- Test **access times** of our structures (wrt, e.g., TD / DOM access)
- Design / implement **updates** on our structure.
- Check impact of our method wrt storing **full XML documents**.

Technical (headache-) Issues, e.g.,

- Whitespace
 - Processing instructions
 - Namespaces
- Etc.

NOTE state-of-the art native XML databases use *pointer-less succinct tree representations*!!

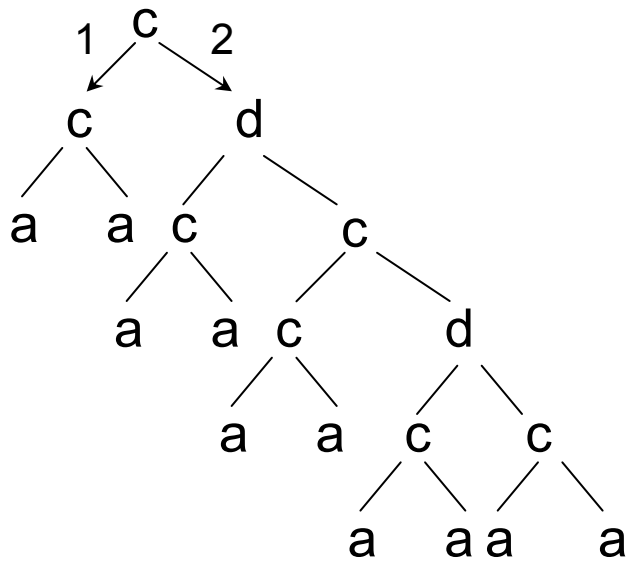
Please see:

[Lam, Shui, Fisher, Wong “Querying and Maintaining Succinct XML Data”, TechRep UNSW-CSE-TR-0424, 2004.]

→ Check impact of our method wrt *storing full XML documents*.

Compressed Trees

Our Trees node-labeled, ranked, ordered, rooted



DAGs

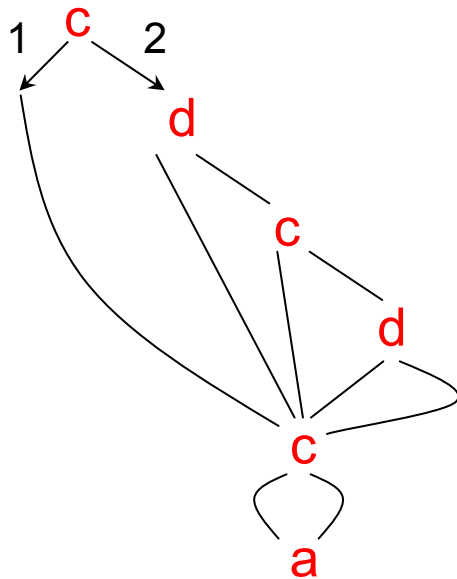
Given a tree t , its **minimal DAG** is

- *unique*
- can be computed in *linear time*
- at most *exponentially* smaller than t
- equivalent to the minimal *regular tree grammar* for $\{t\}$
(= minimal tree automaton for $\{t\}$)

DAGs

Given a tree t , its **minimal DAG** is

- *unique*
- can be computed in *linear time*
- at most *exponentially* smaller than t
- equivalent to the minimal *regular tree grammar* for $\{t\}$
(= minimal tree automaton for $\{t\}$)



$S \rightarrow c(A, d(A, c(A, d(A, A))))$

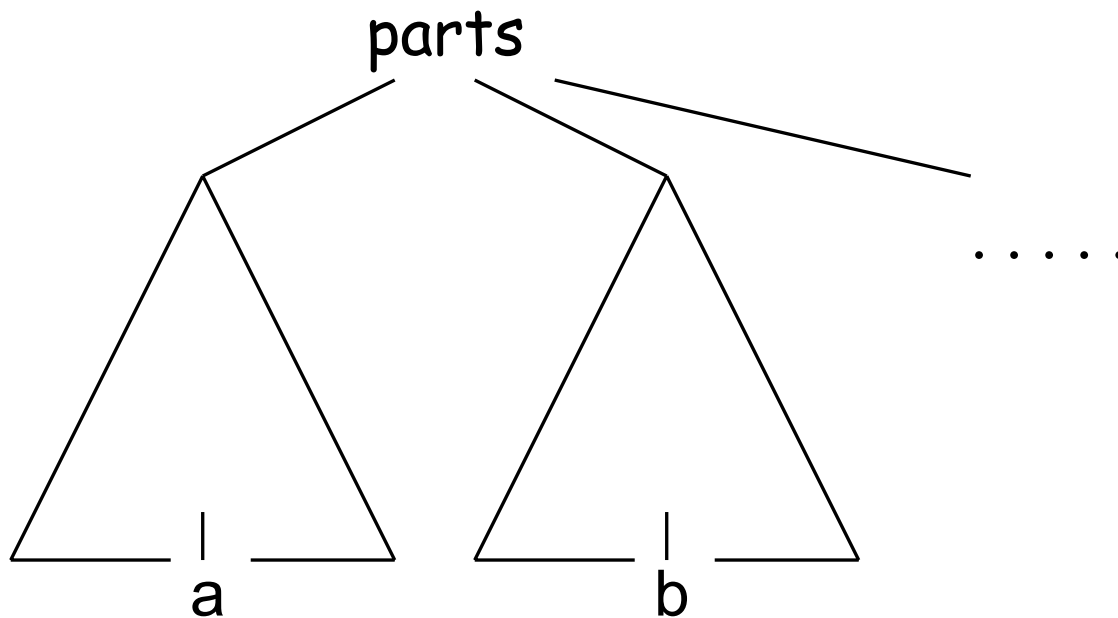
$A \rightarrow c(B, B)$

$B \rightarrow a$

regular tree grammar

Compressed Trees

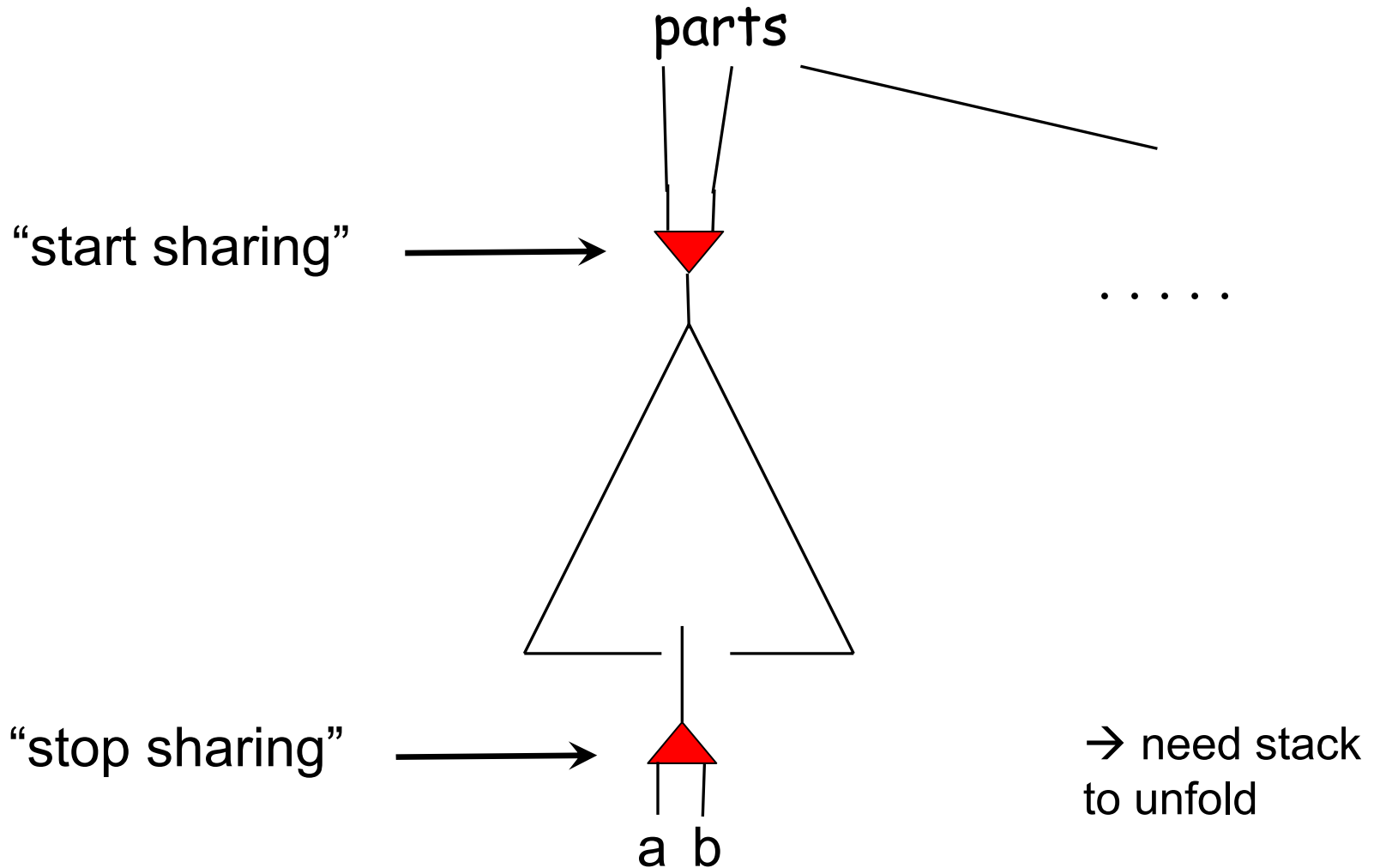
(2) Sharing Graphs [Lamping, POPL1990]



Can **NOT** be shared in a DAG, different in one leaf.

Compressed Trees

(2) Sharing Graphs [Lamping, POPL1990]

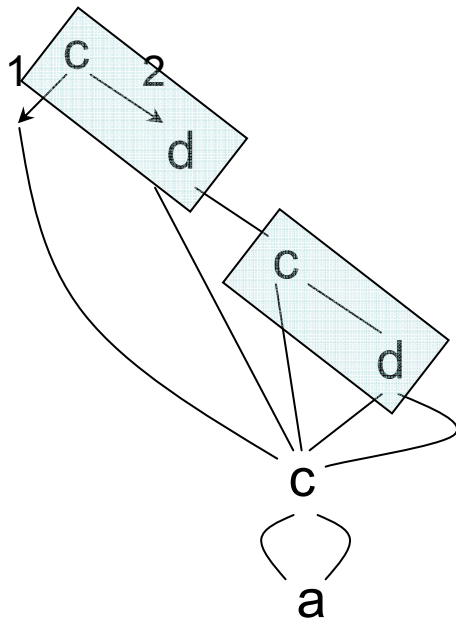


Compressed Trees

(2) **Sharing Graphs** [Lamping, POPL1990]

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

e.g., on a path

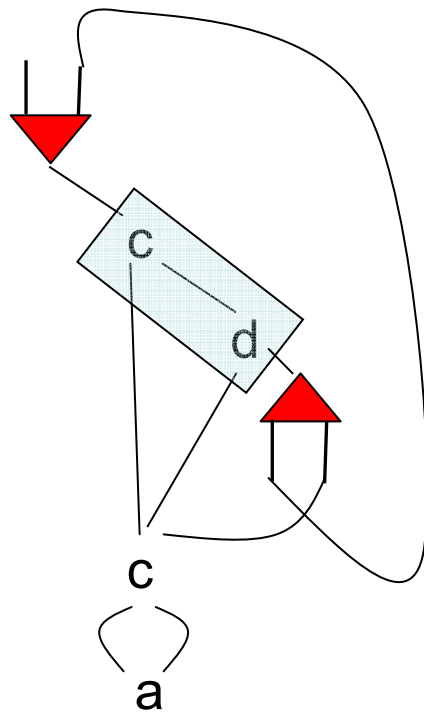


Compressed Trees

(2) **Sharing Graphs** [Lamping, POPL1990]

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

e.g., on a path



Sharing Graphs

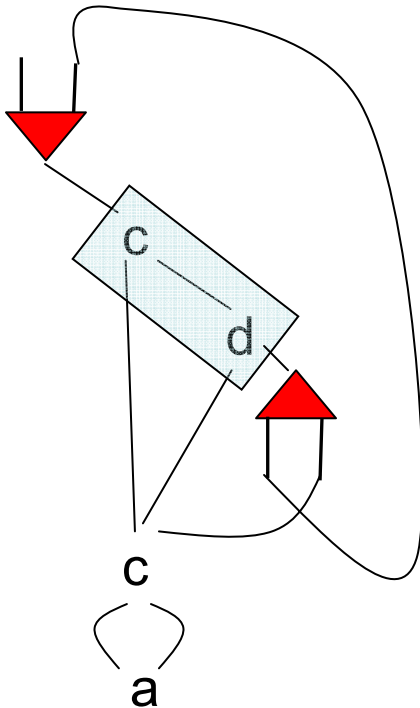
Given a tree t , **minimal** Sharing Graph

- 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

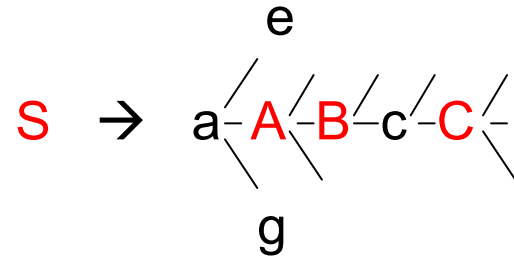
Context-Free Tree Grammars

→ Generalize cf. grammars to trees

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

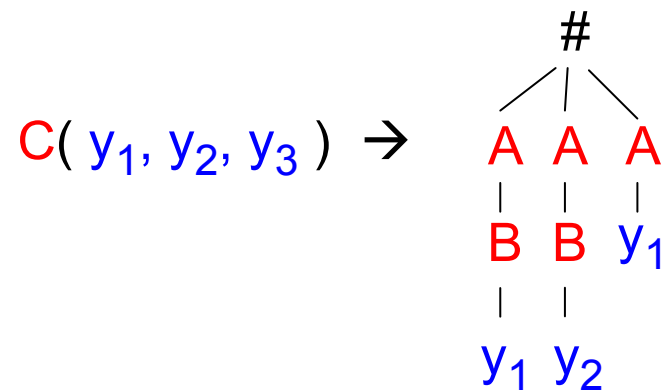
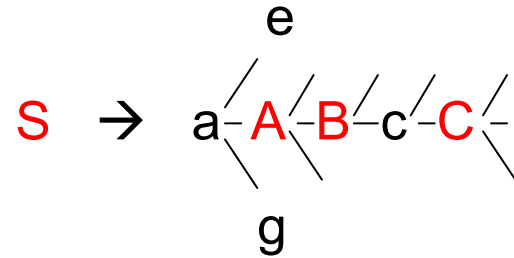
Context-Free Tree Grammars

→ Generalize cf. grammars to trees



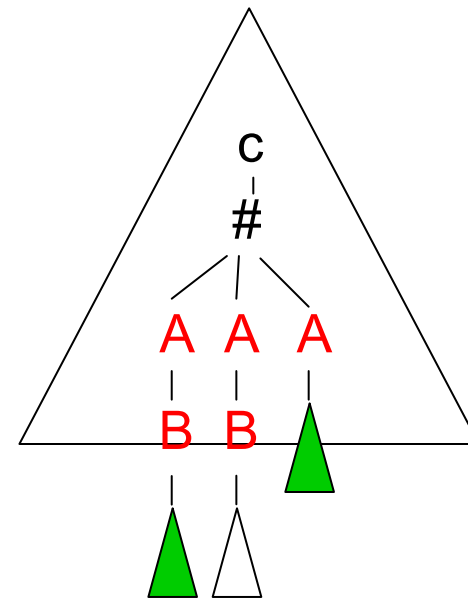
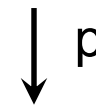
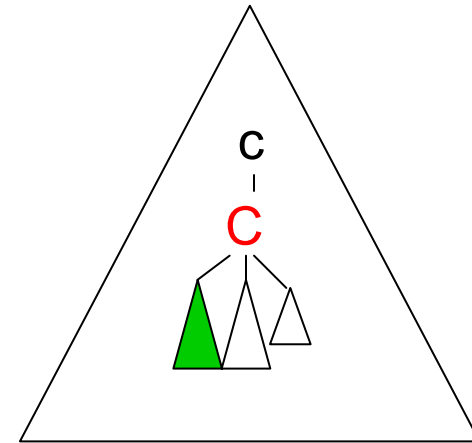
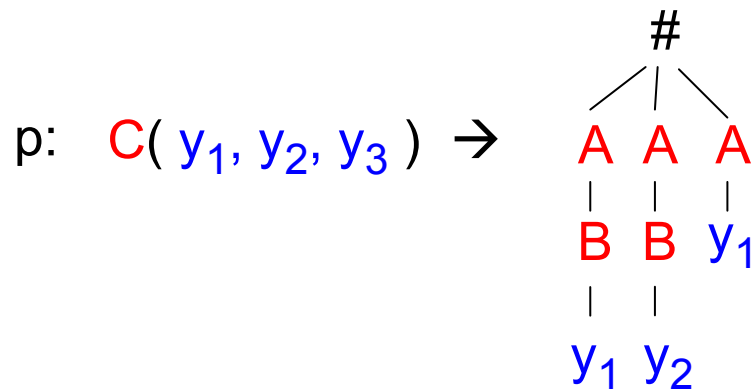
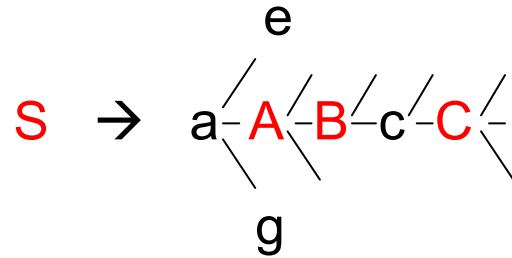
Context-Free Tree Grammars

→ Generalize cf. grammars to trees



Context-Free Tree Grammars

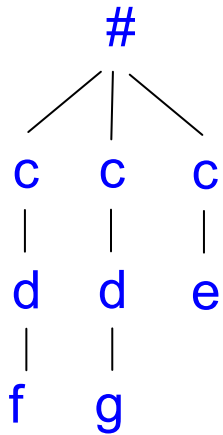
→ Generalize cf. grammars to trees



(→ surface lang's are the *indexed lang's* of [Aho,1967])

Minimal Sharing Graph

→ Why not unique?



size 8/9

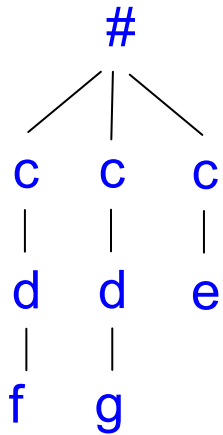
(1) Share all **c**'s and share all **d**'s

(2) Share **c—d** twice

(3) Share **c—d** twice, and
share **c** three times.

Minimal Sharing Graph

→ Why not unique?



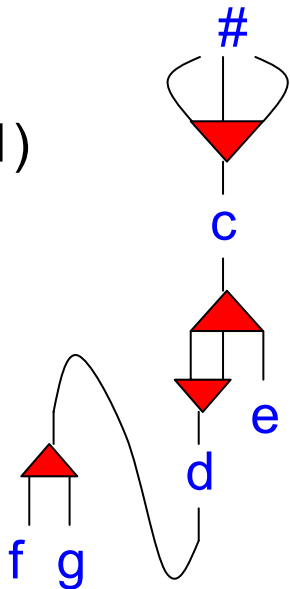
size 8/9

(1) Share all **c**'s and share all **d**'s

(2) Share **c—d** twice

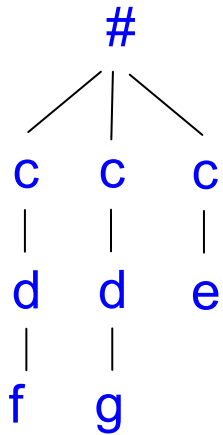
(3) Share **c—d** twice, and share **c** three times.

(1)



Minimal Sharing Graph

→ Why not unique?

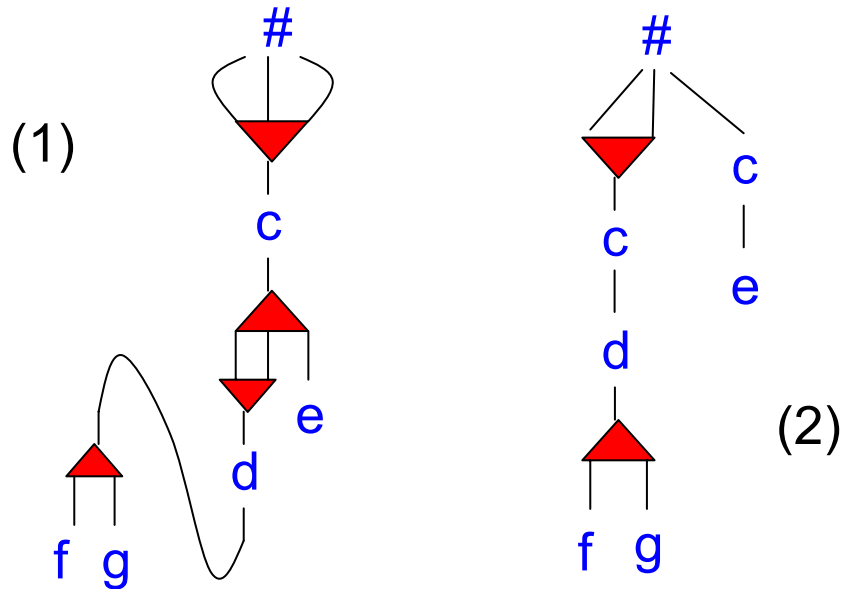


size 8/9

(1) Share all c's and share all d's

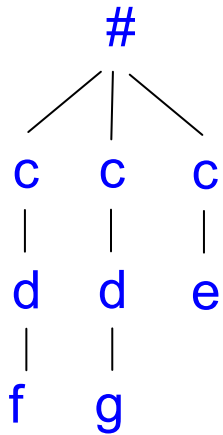
(2) Share c—d twice

(3) Share c—d twice, and share c three times.



Minimal Sharing Graph

→ Why not unique?

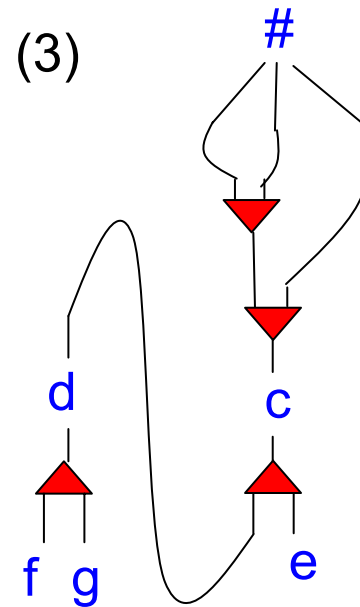
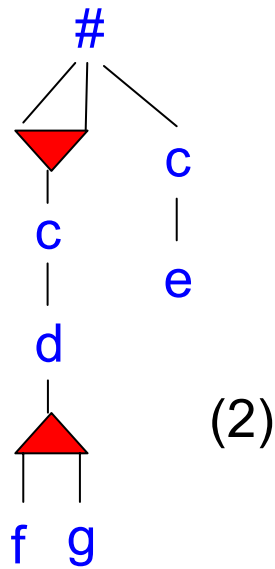
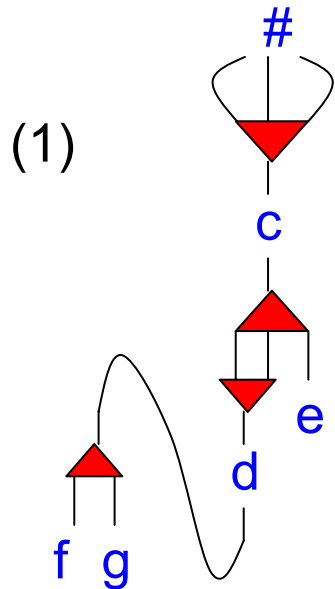


size 8/9

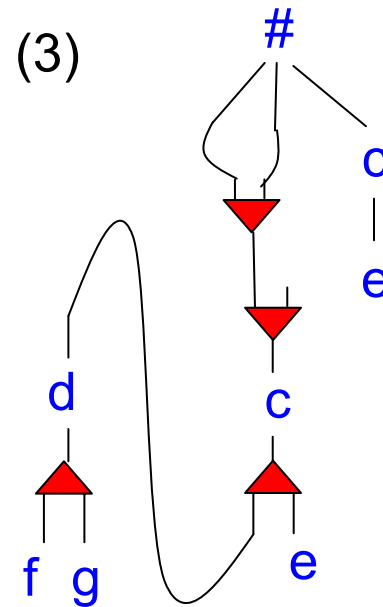
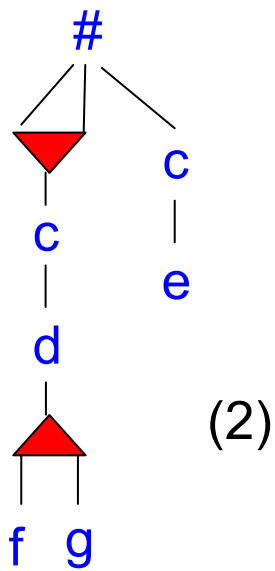
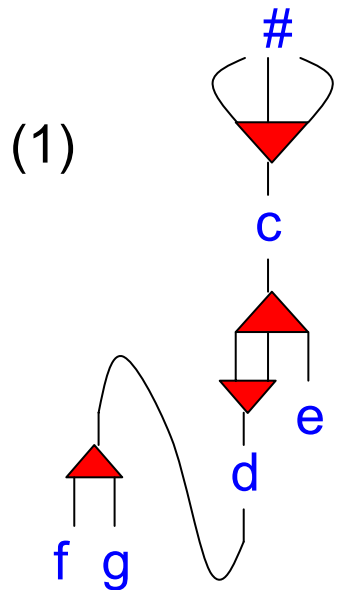
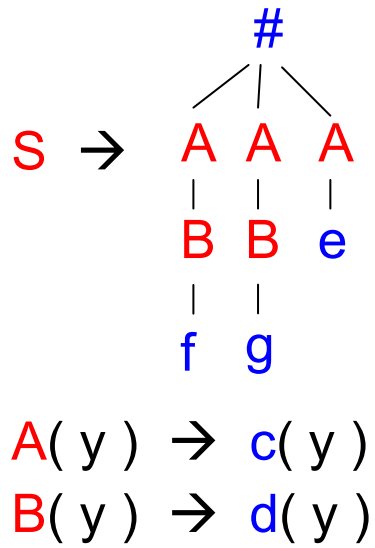
(1) Share all **c**'s and share all **d**'s

(2) Share **c—d** twice

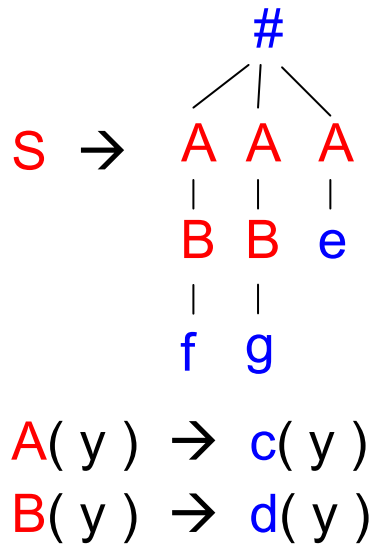
(3) Share **c—d** twice, and share **c** three times.



Minimal Sharing Graph (= cf tree grammar)



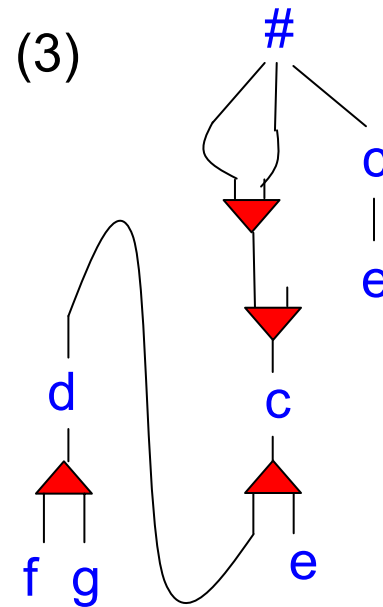
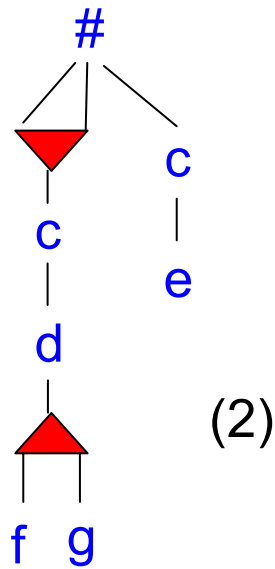
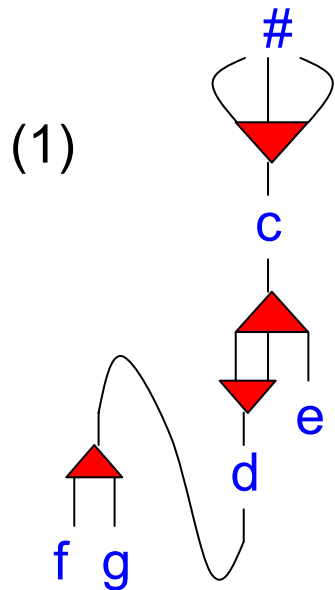
Minimal Sharing Graph (= cf tree grammar)



SIZE

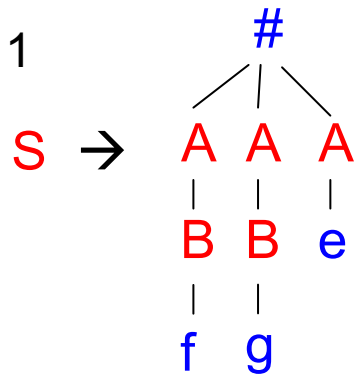
Do NOT count

- edges to parameters (y's)
- y-labeled nodes



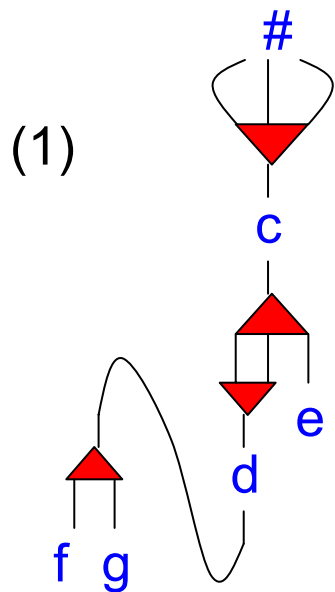
Minimal Sharing Graph (= cf tree grammar)

8/11



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

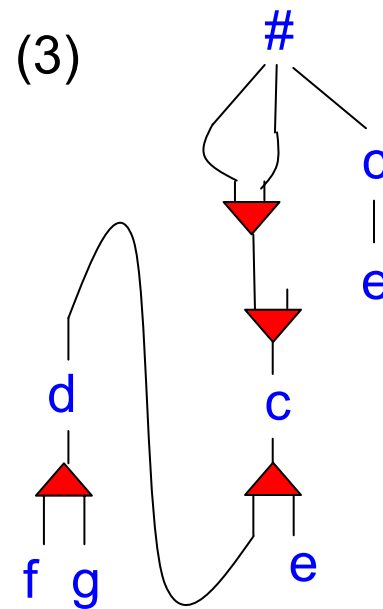
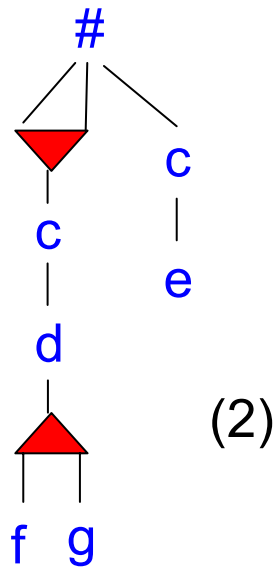
$B(y) \rightarrow d(y)$



SIZE

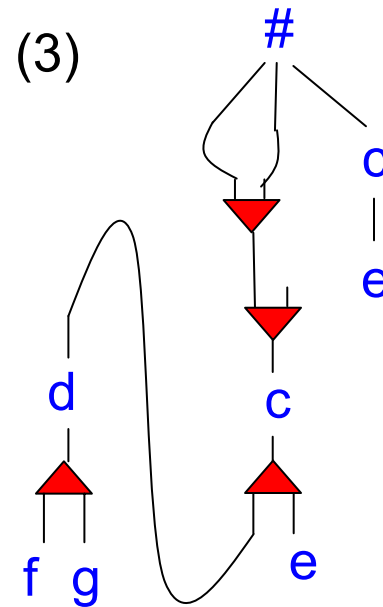
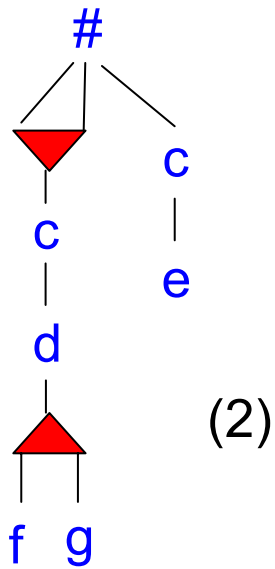
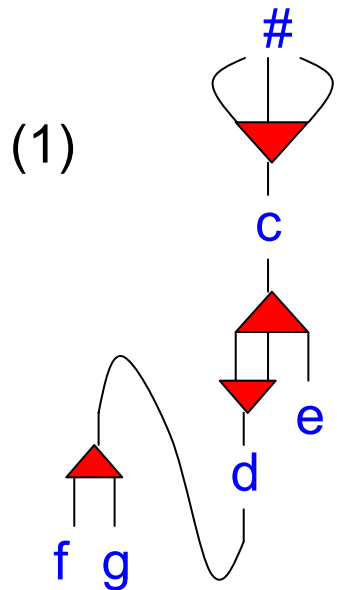
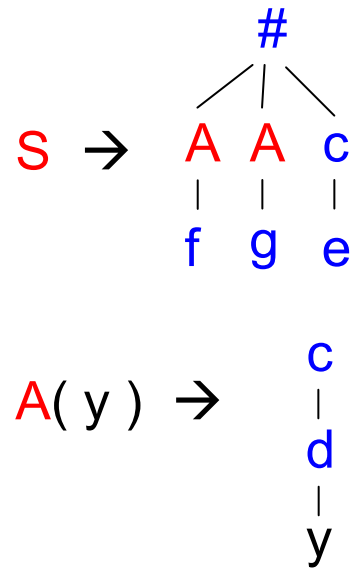
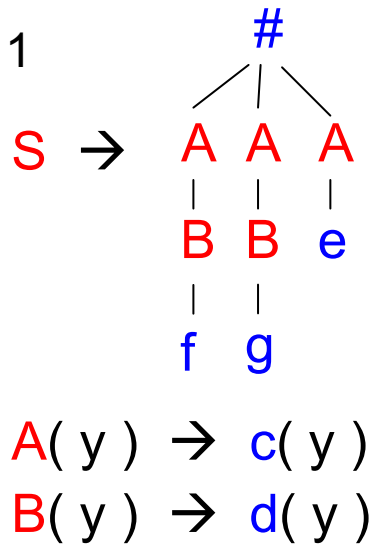
Do NOT count

- edges to parameters (y's)
- y-labeled nodes



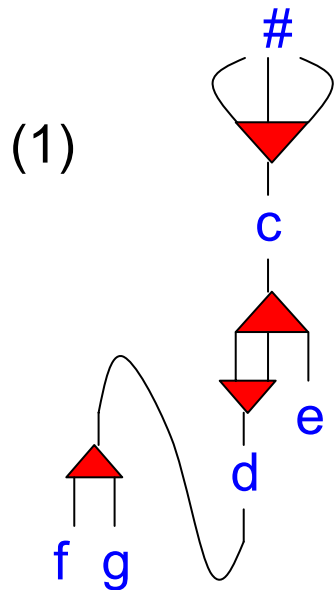
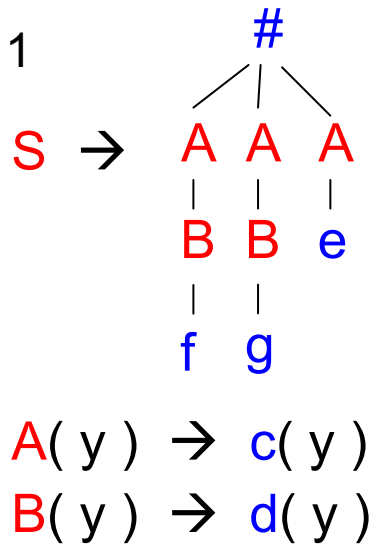
Minimal Sharing Graph (= cf tree grammar)

8/11

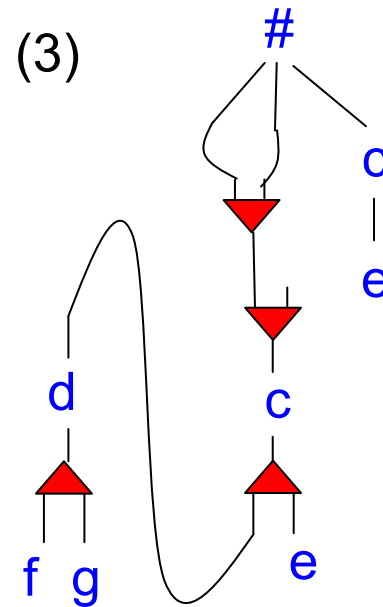
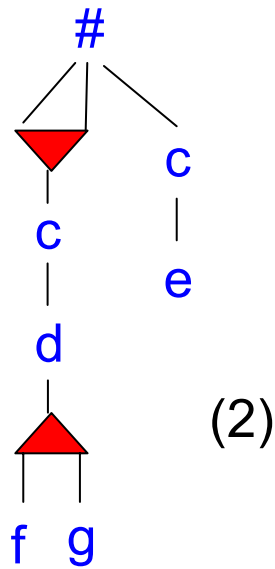
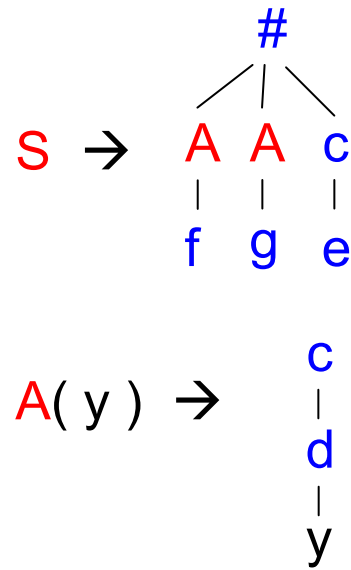


Minimal Sharing Graph (= cf tree grammar)

8/11

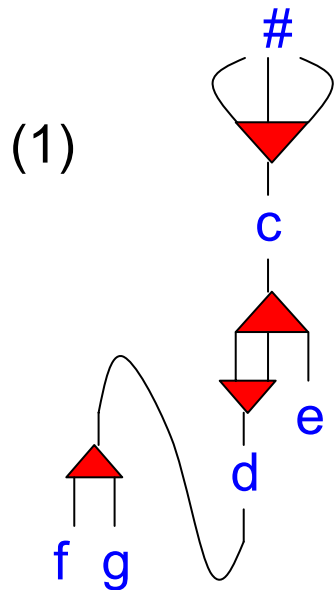
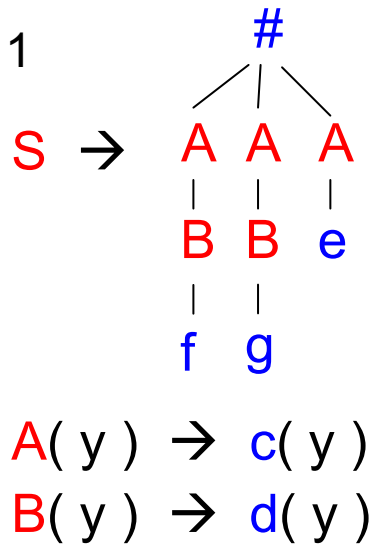


7/9

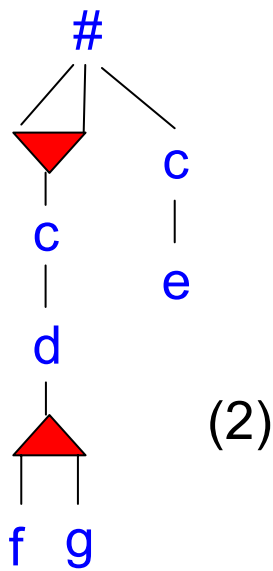
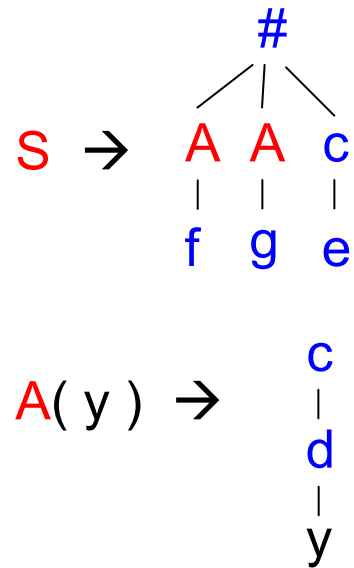


Minimal Sharing Graph (= cf tree grammar)

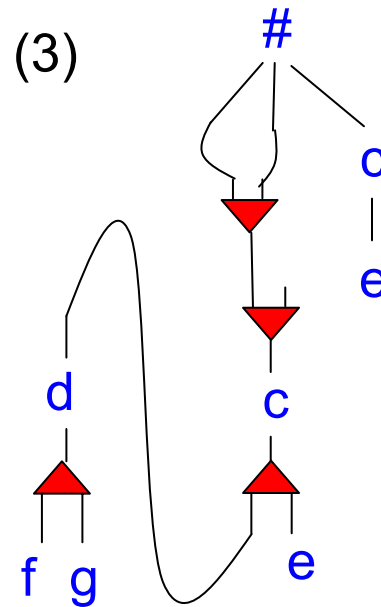
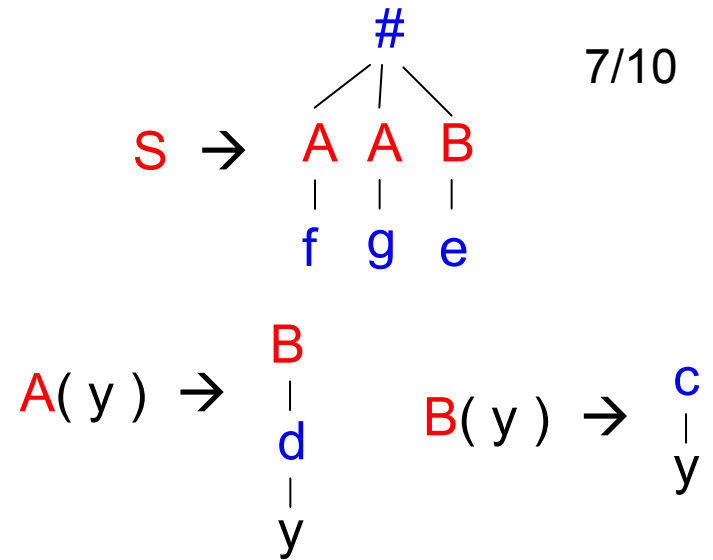
8/11



7/9



7/10



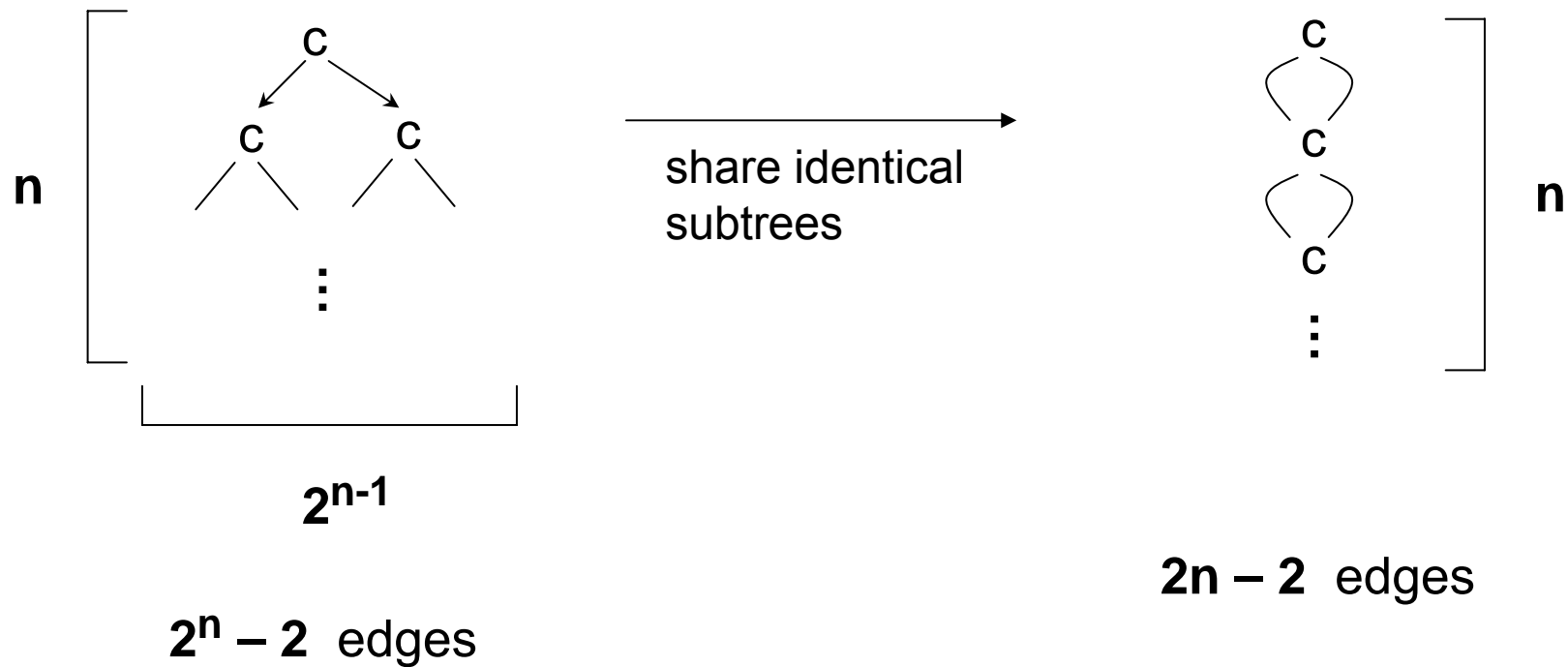
Sharing Graphs

Why **double-exp.** smaller?

Sharing Graphs

Why **double-exp.** smaller?

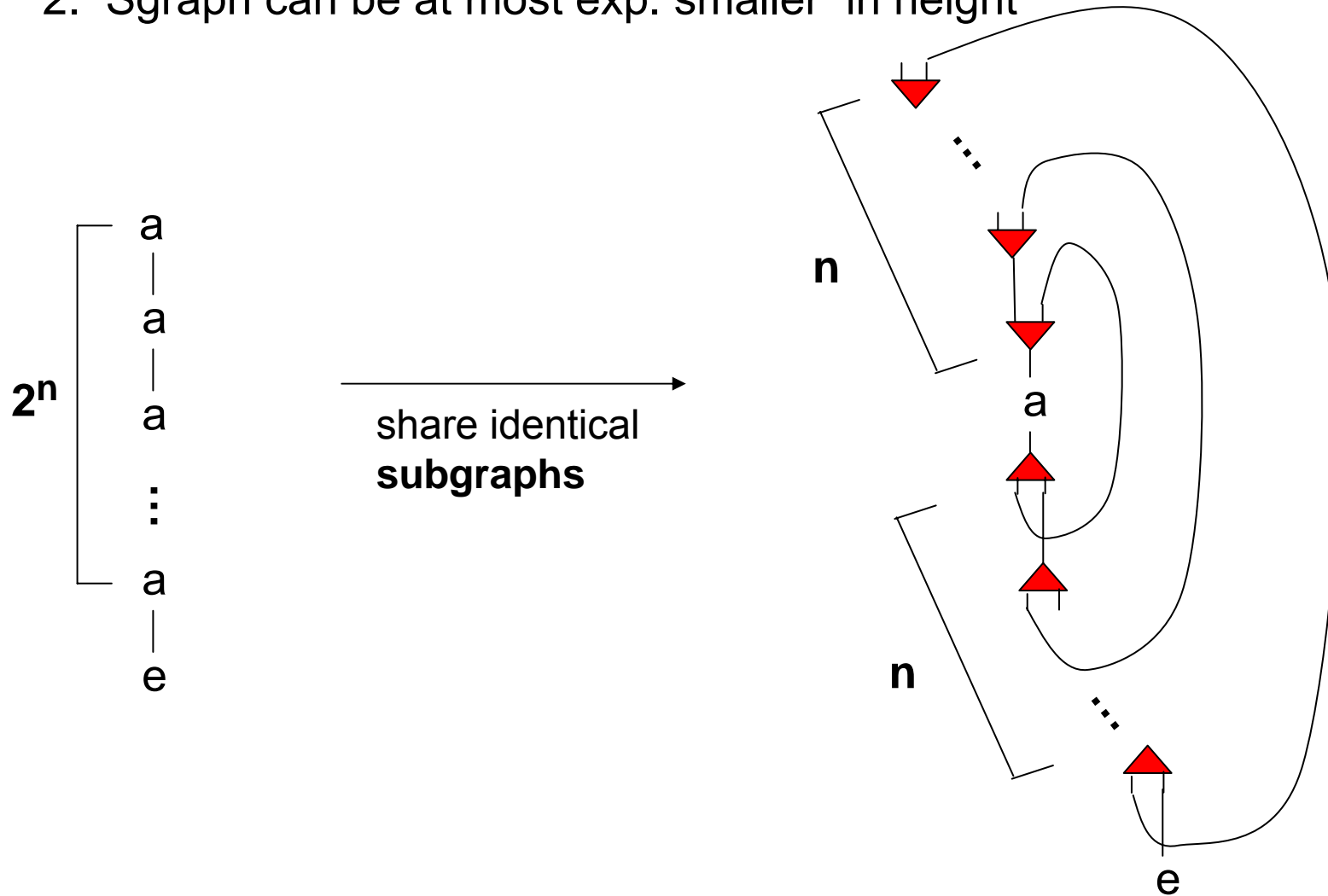
1. DAG can be at most exp. smaller (width)



Sharing Graphs

Why **double-exp.** smaller?

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

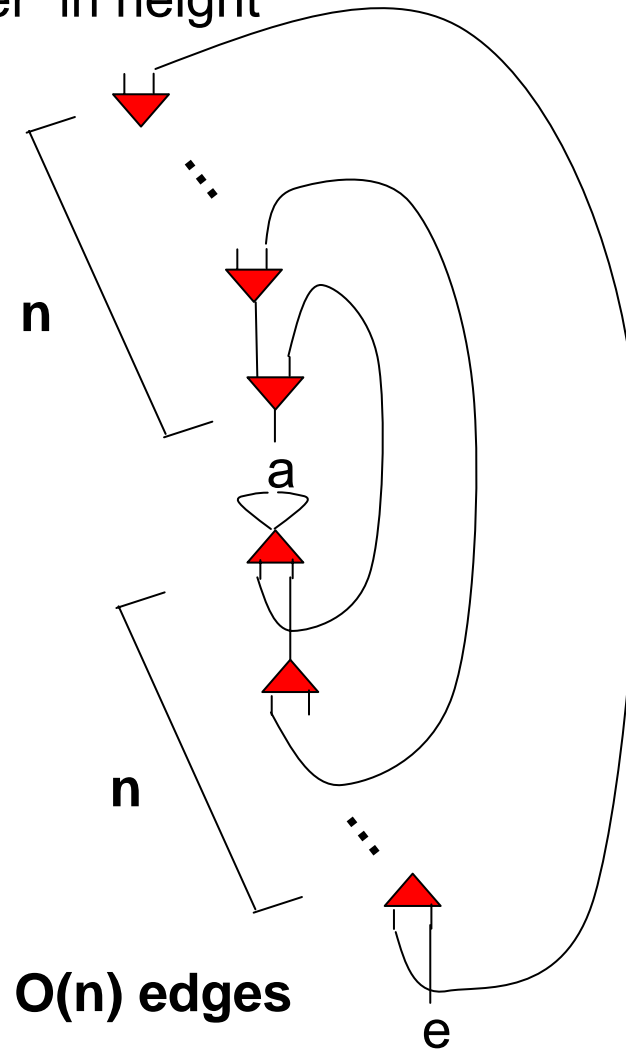
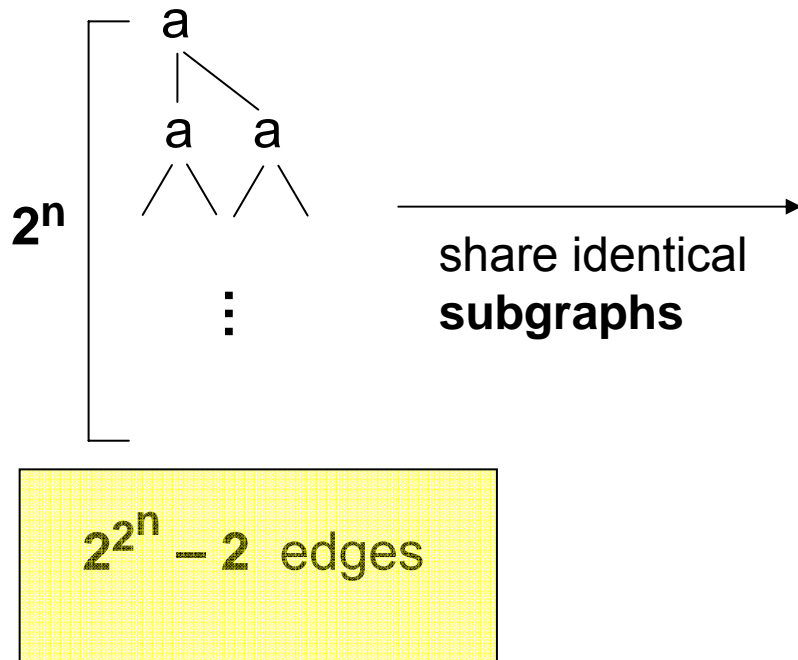


Sharing Graphs

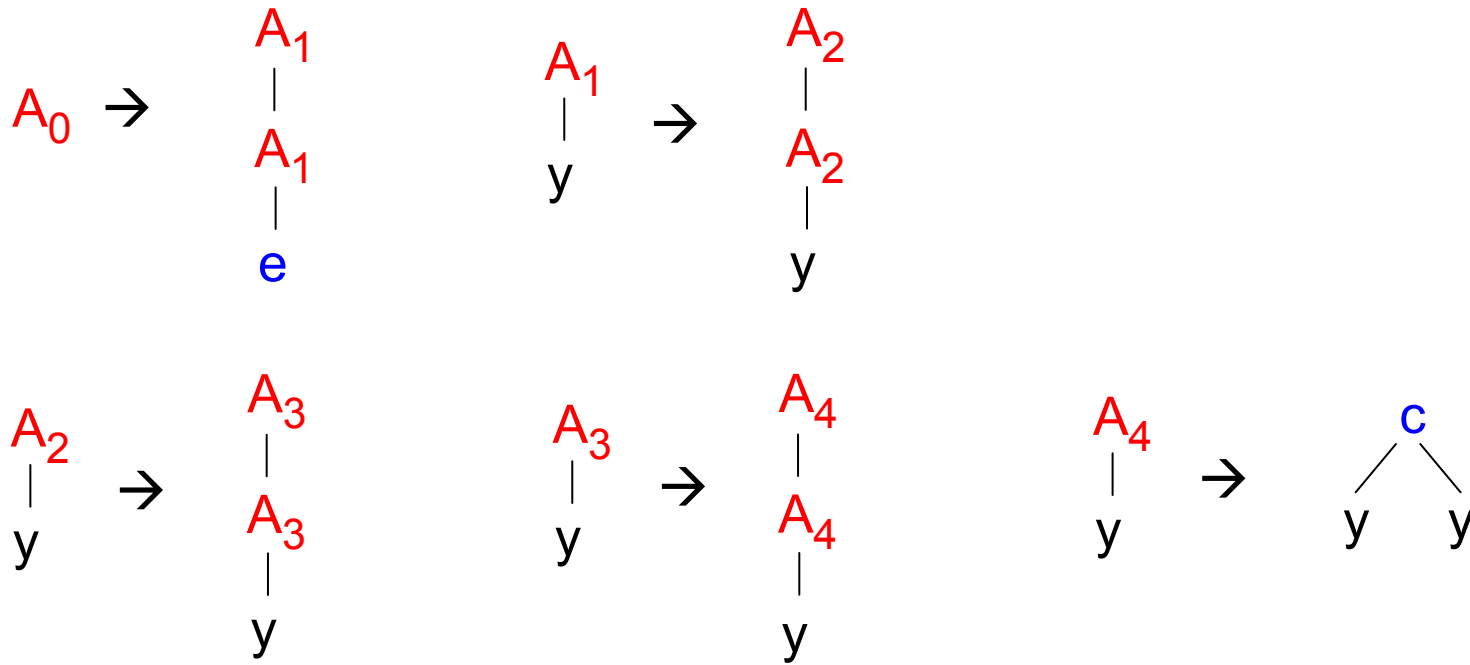
Why **double-exp.** smaller?

Cf tree grammar??

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



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 $A_k \rightarrow rhs$, 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)

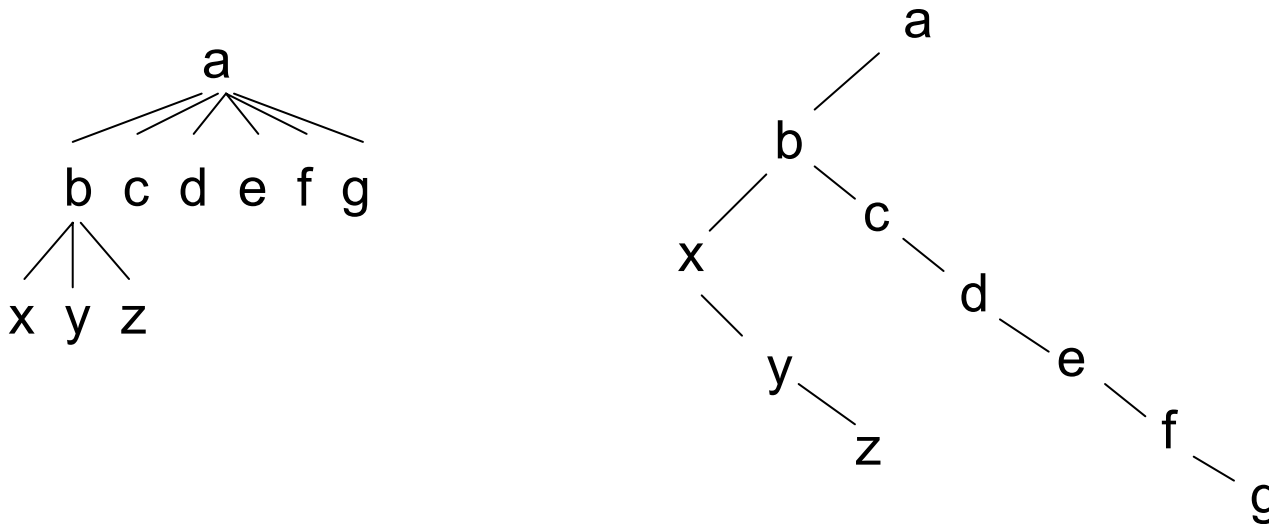
$$A_1 \rightarrow A_2(A_3, a , A_3)$$

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

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** [this paper]

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 $< \text{MAX_PSI_SIZE}$

→ Only consider patterns of rank $< \text{MAX_RANK}$

→ Only search the last WIN_SIZE nodes/productions

BPLEX

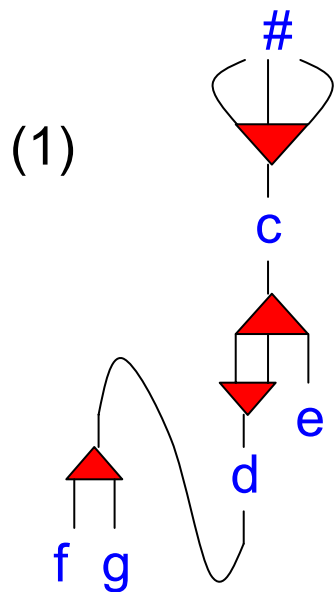
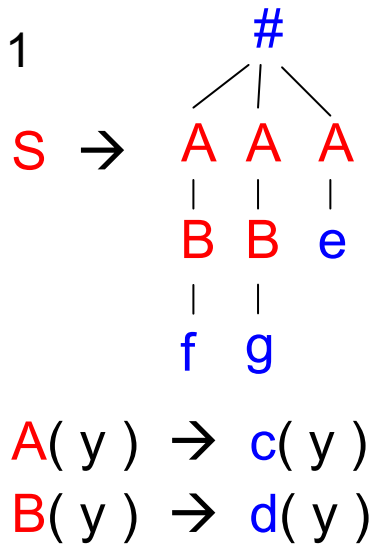
```
procedure BPLEX(G: grammar, KN: int, KS: int, KR: int): grammar
begin
  A := last symbol in the SL ordering of RG
  z := leftmost leaf of rhsG(A)
  while true do
    repM := RepeatedMatches(G, z, KN)
    newM := NewMatches(G, z, KN, KS, KR)
    if newM  $\neq$   $\emptyset$  or repM  $\neq$   $\emptyset$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

Replace pattern by
NonTerminal

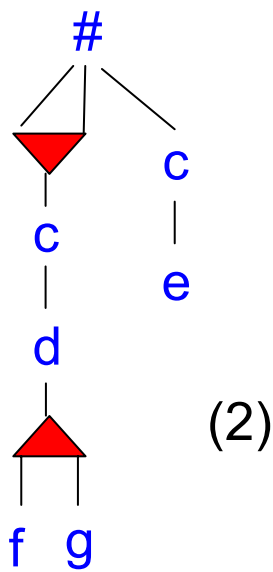
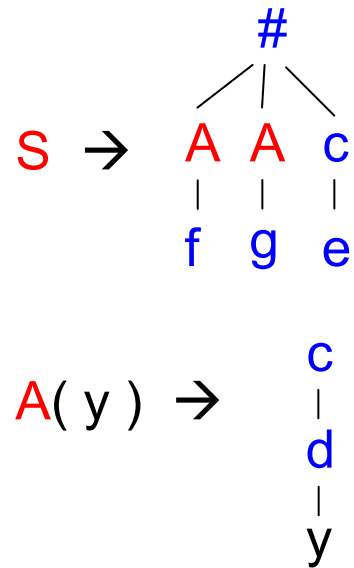
Replace pattern by new
NT and add production

Minimal Sharing Graph (= cf tree grammar)

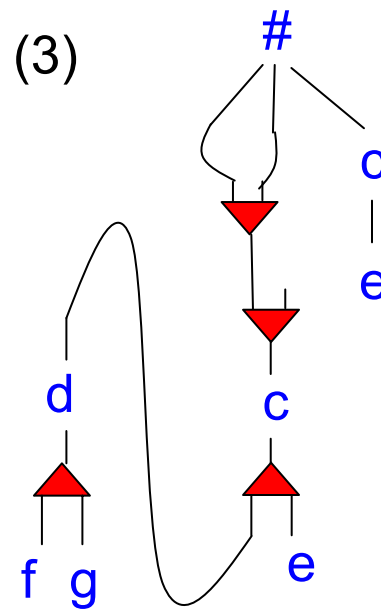
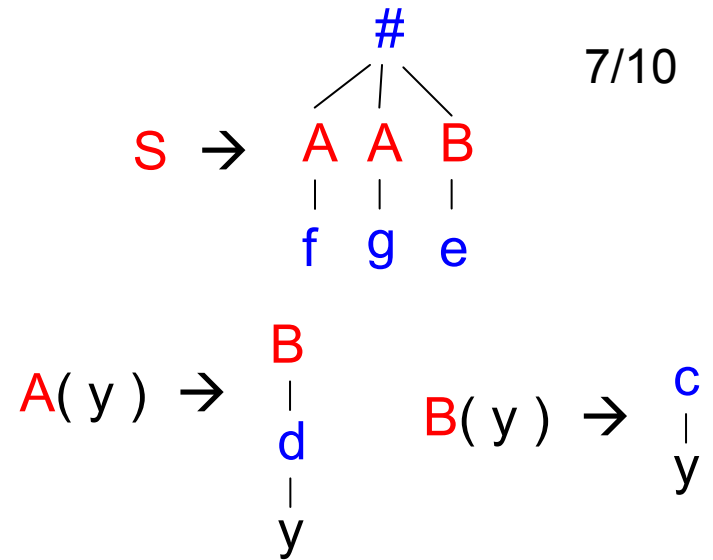
8/11



7/9

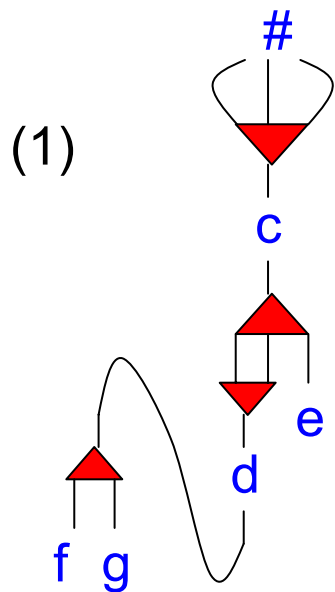
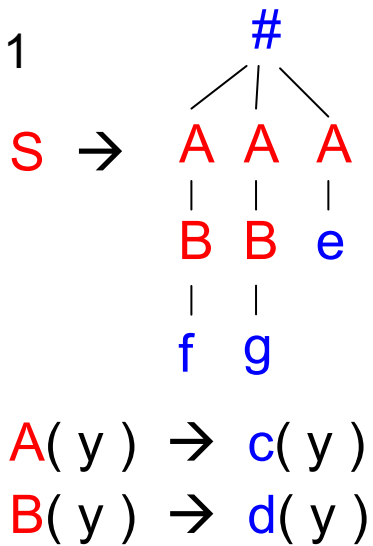


7/10

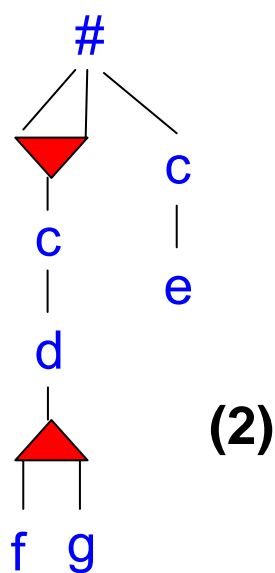
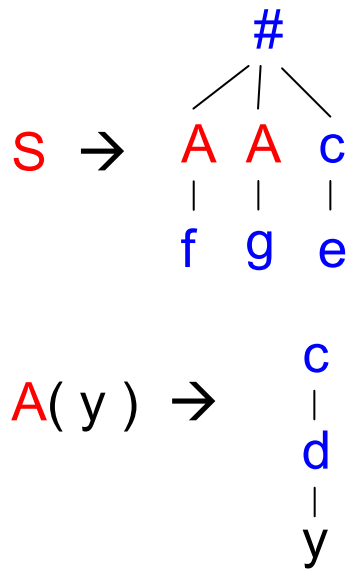



BPLEX output

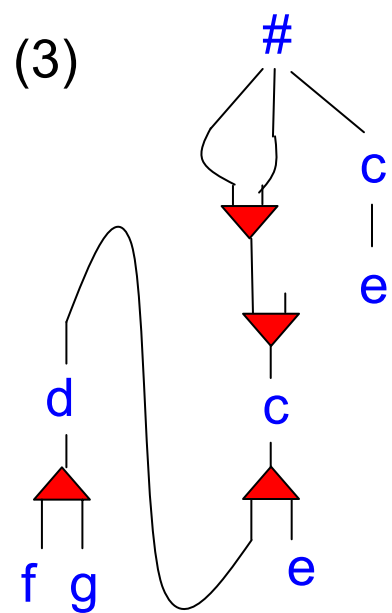
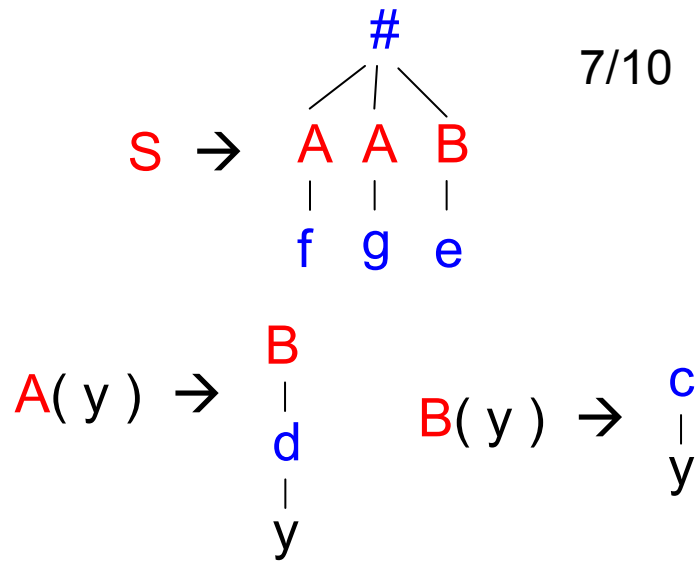
8/11



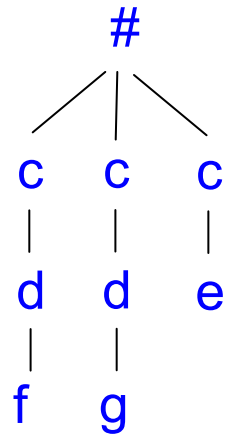
7/9



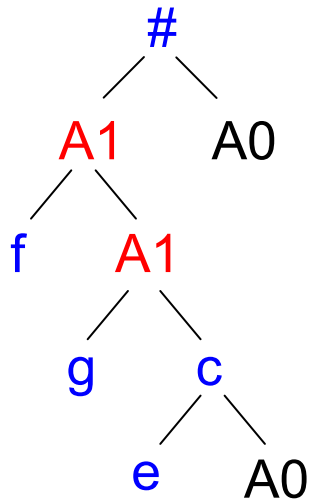
7/10



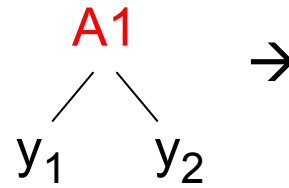
BPLEX output for



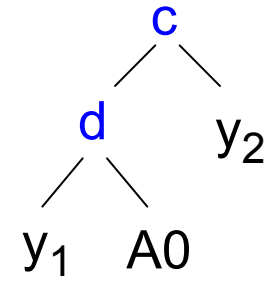
S →



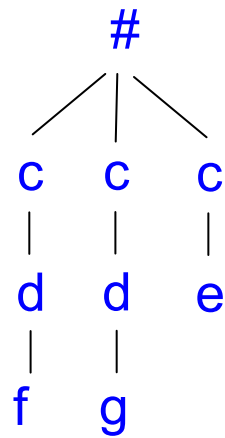
A0 → _



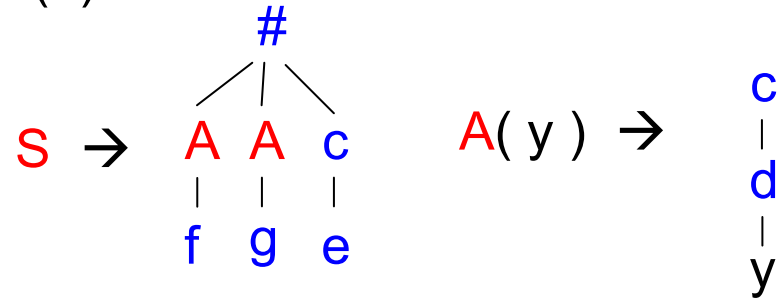
→



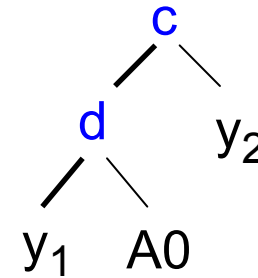
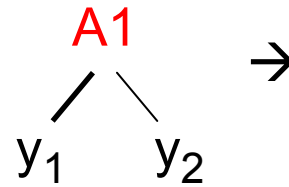
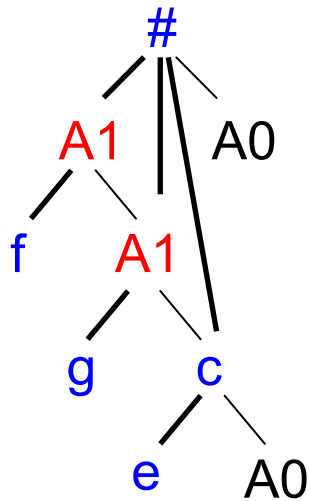
BPLEX output for



(2)



S →

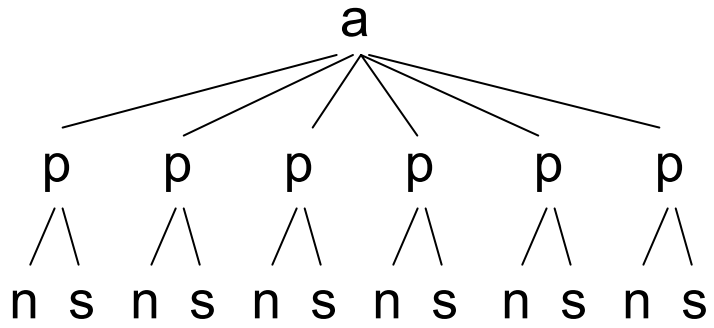


A0 → _

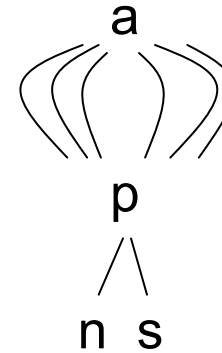
Binary coding of (2)

Unranked vs Binary Trees

18/19



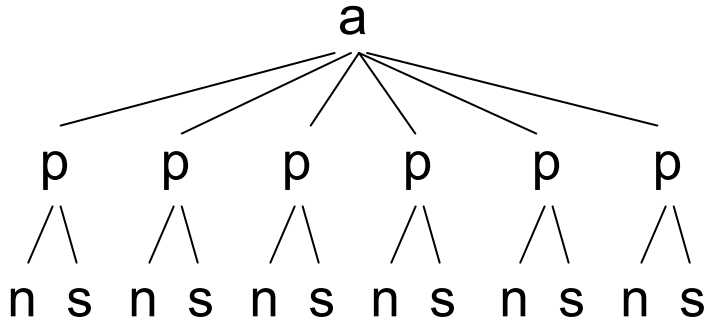
→
Min. DAG



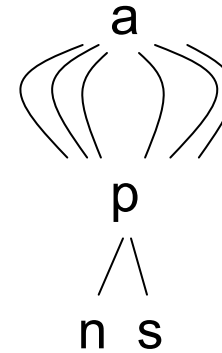
8/4

Unranked vs Binary Trees

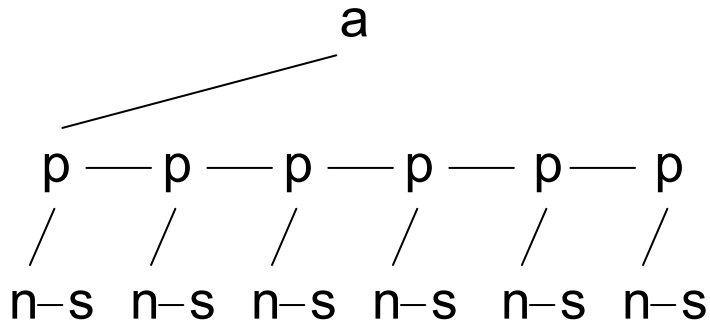
18/19



→
Min. DAG

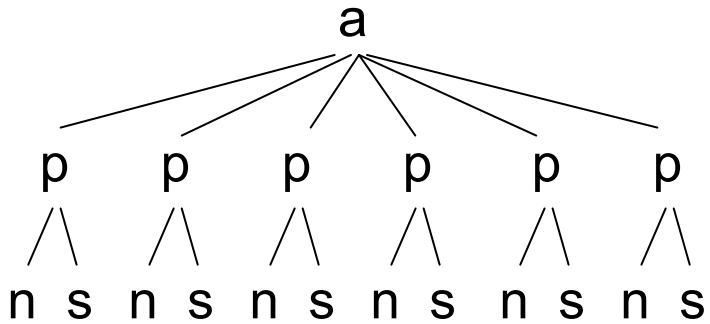


8/4

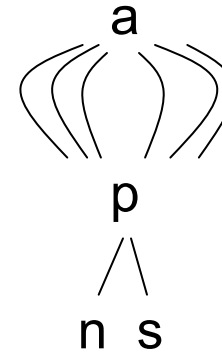


Unranked vs Binary Trees

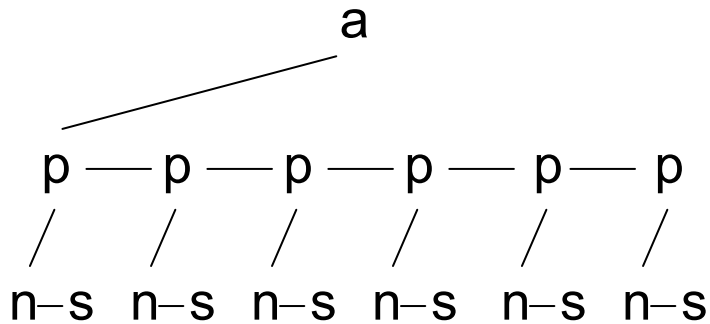
18/19



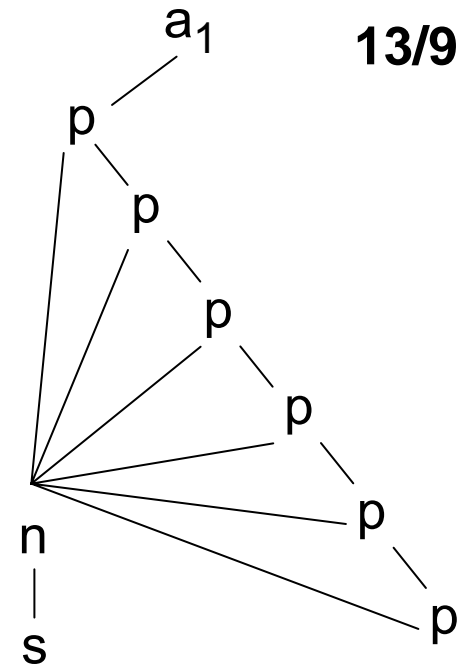
→
Min. DAG



8/4



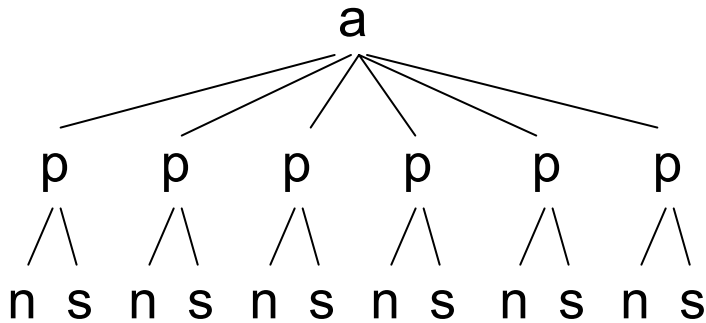
→
Min. DAG



13/9

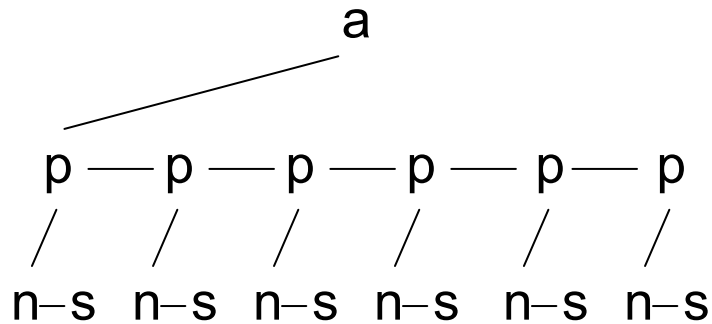
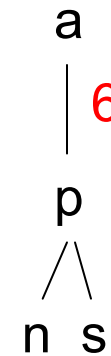
Unranked vs Binary Trees

18/19



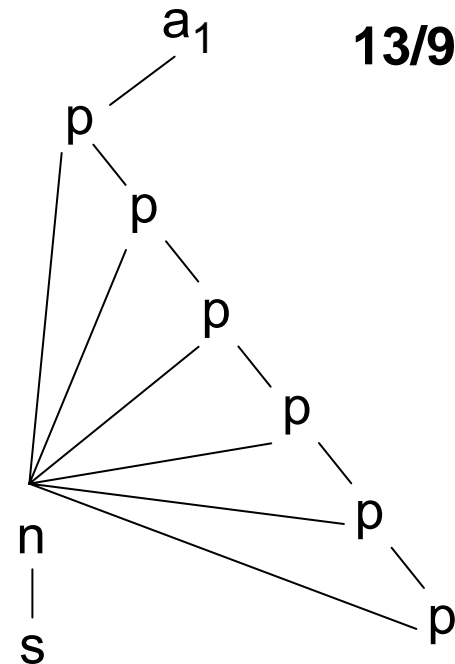
→
Min. DAG

3/4



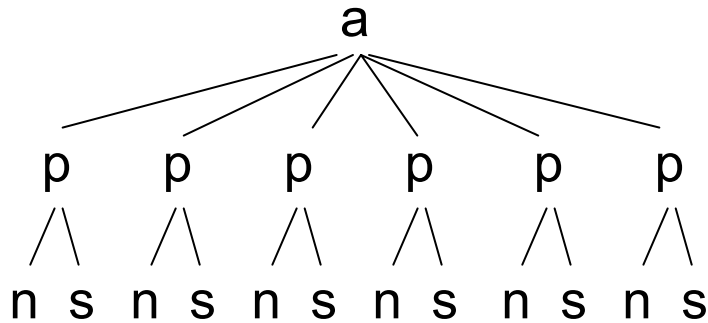
→
Min. DAG

13/9



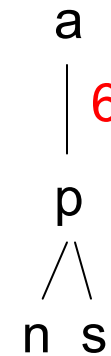
Unranked vs Binary Trees

18/19



→
Min. DAG

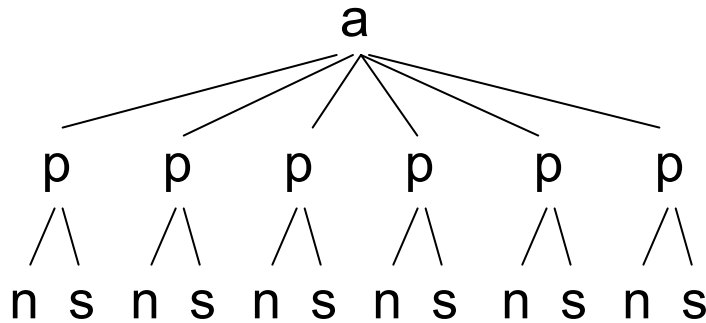
3/4



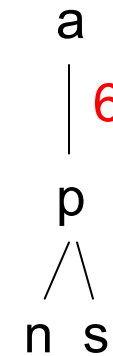
Can it be vica versa? (min bin. DAG is smaller)

Unranked vs Binary Trees

18/19



→
Min. DAG



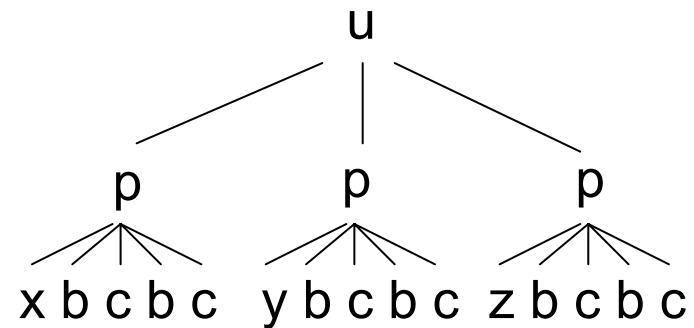
3/4

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

YES!!

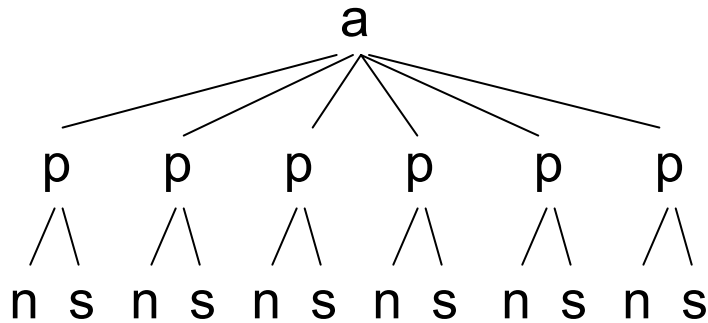
→ Has **18 edges**

→ DAG of bin.coding only **12 edges**

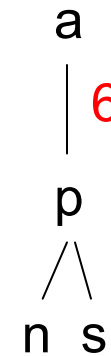


Unranked vs Binary Trees

18/19



→
Min. DAG



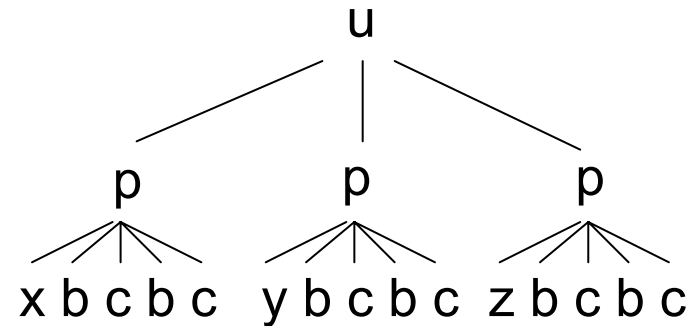
3/4

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

YES!!

→ Has **18 edges**

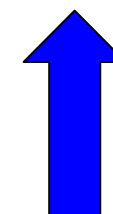
→ DAG of bin.coding only **12 edges**



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	13.2%	1,100,648	10.1%	311,328	2.9%
DBLP (103.6 MB)	2,611,931	533,183	20.4%	222,754	8.5%	115,902	4.4%
Trebank (55.8 MB)	2,447,727	1,454,494	59.4%	1,301,688	53.2%	519,542	21.2%
1998statistics (657 KB)	28,306	2,403	8.5%	726	2.6%	410	1.4%
catalog-02 (104M)	2,240,231	52,392	2.3%	32,267	1.4%	26,774	1.2%
catalog-01 (11M)	225,194	6,990	3.1%	8,503	2.8%	3,817	1.7%
dictionary-02 (104M)	2,731,764	681,130	24.9%	441,322	16.2%	160,329	5.9%
dictionary-01 (11M)	277,072	77,554	28.0%	46,993	17.0%	20,150	7.3%
JST_snp.chr1 (36M)	655,946	40,663	6.2%	25,047	2.3%	12,858	1.8%
JST_gene.chr1 (11M)	216,401	14,606	6.7%	5,658	2.6%	4,000	1.8%
NCBI_snp.chr1 (190M)	3,642,225	809,394	22.2%	15	<0.1%	59	<0.1%
NCBI_gene.chr1 (24M)	360,350	14,356	4.0%	11,767	3.3%	7,160	2.0%

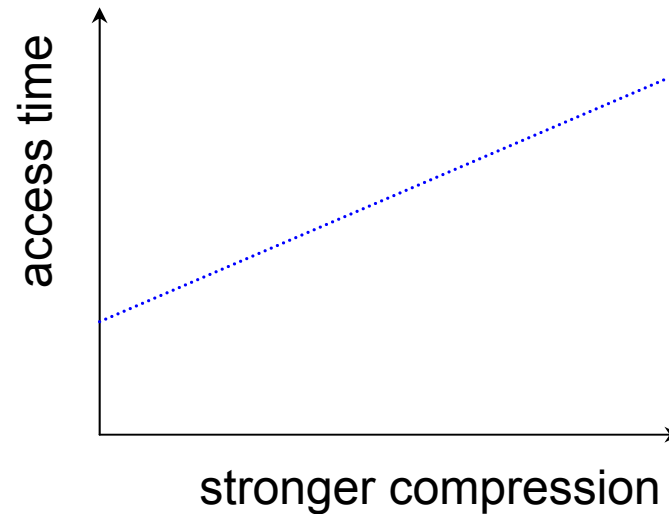


→ BPLEX on unranked trees gives (almost) same as on bin. trees.

BPLEX compression insensible 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 \rightarrow S2(S2(a))
S2(y) \rightarrow S3(S3(y))
S3(y) \rightarrow S4(S4(y))
S4(y) \rightarrow g(y , y)

A $\delta(a) = q$
 $\delta(g(q , q)) = q'$
 $\delta(g(q' , q')) = q$

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 $S1 \rightarrow S2(S2(a))$ **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 $\text{rhs}(S4) = g(y, y)$.

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

Run **A** on $\text{rhs}(S4) = g(y, y)$.

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

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

Run **A** on $\text{rhs}(S4) = g(y, y)$.

→ function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

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

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

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

Run **A** on $\text{rhs}(S4) = g(y, y)$.

→ function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

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

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

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

Run **A** on $\text{rhs}(S4) = g(y, y)$.

→ function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

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

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

Algorithms on Compressed Trees

Theorem Given *linear SL of tree grammar G with k parameters*
T(A) 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(!).

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

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

Run A on $\text{rhs}(S4) = g(y, y)$.

→ function σ_{S4} from $\{q, q'\}$ to $\{q, q'\}$

$$\begin{array}{lll} \sigma_{S4}(q) = q' & \sigma_{S3}(q) = \sigma_{S4}(\sigma_{S4}(q)) = q & \sigma_{S2}(q) = q \\ \sigma_{S4}(q') = q & \sigma_{S3}(q') = \sigma_{S4}(\sigma_{S4}(q')) = q' & \sigma_{S2}(q') = q' \end{array}$$

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

Complexity Results [Lohrey/Maneth, CAA'05]

		det. TDTA	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. TDTA	det. BUTA	TA	Core XPath
uncompressed trees [13]	fixed	NC ¹ -complete			PSPACE-Complete[1]
	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				

Theorem *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_j 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 **H1, H2** generating *depth-first left-to-right traversals* of $\text{tree}(G_1), \text{tree}(G_2)$

For A of G_1 with rank $k > 0$,

H1 has nonterminals $A_{0,1}, A_{1,2}, \dots, A_{k-1,k}, A_{k,0}$

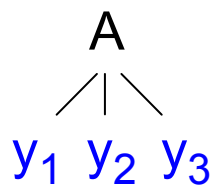
Theorem Testing equivalence of two SL of tree grammars $G1, G2$ can be done in PSPACE, *and in polynomial time if $G1$ and $G2$ are linear.*

→ Construct cf (string) grammars **H1, H2** generating *depth-first left-to-right traversals* of $\text{tree}(G1), \text{tree}(G2)$

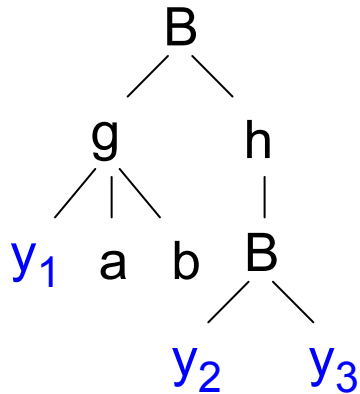
For A of $G1$ with rank $k > 0$,

H1 has nonterminals $A_{0,1}, A_{1,2}, \dots, A_{k-1,k}, A_{k,0}$

G1:



→



H1:

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

Theorem *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

Claim **SL cf. tree grammars** can represent XML tree structures *more space efficiently* than DAGs!

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

have same complexity bounds on linear SL grammars as on DAGs!

Open

- How big are efficiency gains in practice?
(full queries on full XML docus)
- Succinct Representation of SL grammars?!
- Equivalence of non-linear SL grammars. In PTIME?
- Better approx. algos than BPLEX!? → a la [Rytter](#)

→ The END ←

