

Tree Automata and XPath on Compressed Trees

Markus Lohrey
University Stuttgart, Germany

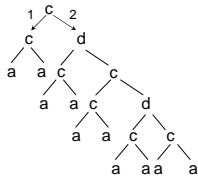
Sebastian Maneth
EPFL, Switzerland

Outline

- Compressed Trees: DAGs & SL Grammars
- Tree Automata (TA)
- Membership Problems
- XML Type Checking
- Conclusions

Compressed Trees

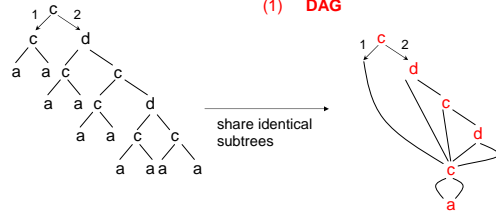
Our Trees node-labeled, ranked, ordered, rooted



Compressed Trees

Our Trees node-labeled, ranked, ordered, rooted

(1) DAG



18/19 edges/nodes

10/6 edges/nodes

DAGs

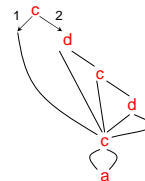
Given a tree t , its **minimal DAG** is

- *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 $\{t\}$)



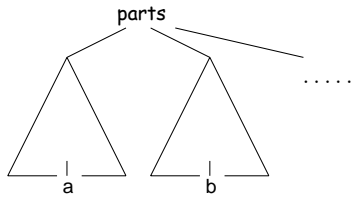
$S \rightarrow c(A, d(A, c(A, d(A, A))))$

$A \rightarrow c(B, B)$

$B \rightarrow a$

Compressed Trees

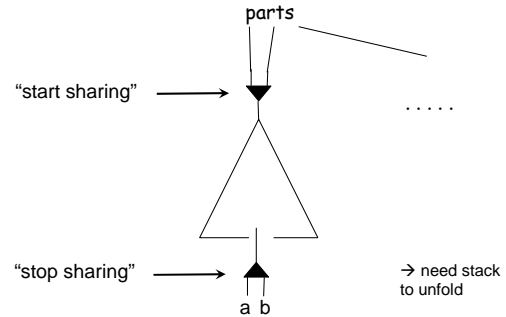
(2) **Sharing Graphs** [Lamping, POPL1990]



Can NOT be shared, different in one node.

Compressed Trees

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

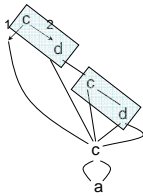


Compressed Trees

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

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

e.g., on a path

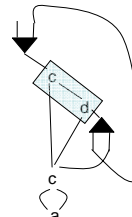


Compressed Trees

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

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

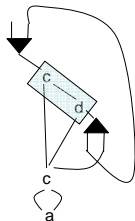
e.g., on a path



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 $\{t\}$



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

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

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

$$B \rightarrow a$$

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)

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

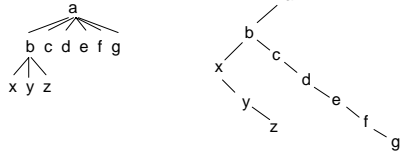
$$A_2(y_1, y_2, y_3) \rightarrow A_5(y_1, y_2, A_6(y_3, y_3))$$

Compressing XML skeletons

→ **DAGs** common XML tree skeletons compress to
 ≈ **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/Maneth04]

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



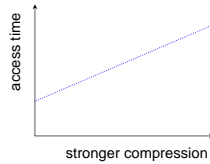
Compressing XML skeletons

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	115,902
Treebank (55.8 MB)	2,447,727	1,454,494	1,301,688	519,542
1998statistics (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	48,993	20,150
JST_snp.chr1 (36M)	655,946	40,663	25,047	12,858
JST_gene.chr1 (11M)	216,401	14,606	5,659	4,000
NCBI_snp.chr1 (190M)	3,642,225	809,394	15	59
NCBI_gene.chr1 (24M)	380,350	14,356	11,767	7,160



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!! [this paper]

- Problem (A) run deterministic TD tree automaton
- (B) run deterministic BU tree automaton
- (C) run non-deterministic BU tree automaton
- (D) evaluate Core Xpath expression

Tree Automata

(A) Deterministic Top-Down

$$\delta(q_0, g(x_1, x_2)) = (q_1, q_2)$$

(B) Deterministic Bottom-Up

$$\delta(g(q_1, q_2)) = q$$

Note

Det. BU = Nondet. BU = Nondet TD ≠ Det. TD

{g(a,b), g(b,a)} is *not* det. TD

Results

		det. TDTA	det. BU/A	TA
uncompressed trees [13]	fixed	NC ¹ -complete		
	uniform	L-complete	LOGCFL, 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 (2)		PSPACE-complete
	uniform			
unrestricted SL	fixed	P-complete (1)	PSPACE-complete	
	uniform			

(1) The uniform membership problem for deterministic TDTAs and SL of tree grammars is in P.

Idea from cf tree grammar G generate small cf grammar C for the paths in G

S → A(a, b)
 A(x1, x2) → B(B(x1, x2), B(x1, x2))
 B(x1, x2) → g(x1, x2)

S → A1 a | A2 b
 A1 → B1 B1 | B2 B1
 A2 → B1 B2 | B2 B2
 Bi → gi for i ∈ {1, 2}

(1) The uniform membership problem for deterministic TDTAs and SL of tree grammars is in P.

Idea from cf tree grammar G generate small cf grammar C for the paths in G
 from det. TDTA A generate DFA B for paths of L(A)

$$L(C) - L(B) = \emptyset \quad \text{iff} \quad L(G) - L(A) = \emptyset$$

→ C constructable in poly time in |G|
 → B constructable in poly time in |A|

(2) 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 $\text{eval}(G) \in T(A)$.
 Time $O(n^k |G| |A|)$ if A deterministic and G nonlinear(!).

Idea run automaton A on the rhs's of 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)

XML 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 ⊆ Schemas ⊆ Relax

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

XPath Evaluation

→ As for DAGs, it is PSPACE-complete to evaluate a (core) XPath expression on a linear SL grammar G.

Important a node of $\text{eval}(G)$ can be represented in poly space wrt |G|

Conclusions

Linear SL grammars can represent XML documents more space efficiently than DAGs!

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

have same complexity bounds on SL grammars as on DAGs!

Open Problems

- How big are efficiency gains in practice?
- XPath on non-linear SL grammars. In PSPACE?