Enumeration for MSO-Queries on Compressed Trees
Part 2

Markus Lohrey and Markus Schmid

Universität Siegen and Humboldt-Universität zu Berlin

DLT 2024

The Main Result

Lohrey, Schmid 2024

Fix a query $\Phi(X)$. One can enumerate $\mathsf{select}(\Phi(X), \mathsf{val}(\mathcal{G}))$ for a given FSLP $\mathcal{G}$ in linear preprocessing time and output-linear delay.

The Main Result

Lohrey, Schmid 2024

Fix a query $\Phi(X)$. One can enumerate $\mathsf{select}(\Phi(X), \mathsf{val}(\mathcal{G}))$ for a given FSLP $\mathcal{G}$ in linear preprocessing time and output-linear delay.

Proof roadmap:

Step 1: Reduction to a slightly simpler problem about tree automata and DAG-compressed binary trees.

Step 2: Extension of a known enumeration algorithm for tree automata on binary trees to the case of DAG-compressed binary trees,...

Step 3: ...which boils down to solving a problem of enumerating paths in a DAG.

Step 1 – Tree Automata as Query Mechanisms

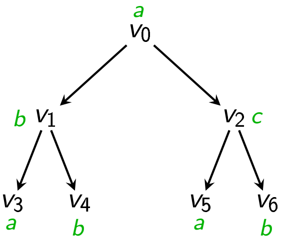
Let $T = (V, E)$ be a tree and let $S \subseteq V$.

mark(T, S): mark all nodes from S in T.
(formally, label a of node v is replaced by either $(a, 0)$ or $(a, 1)$.)

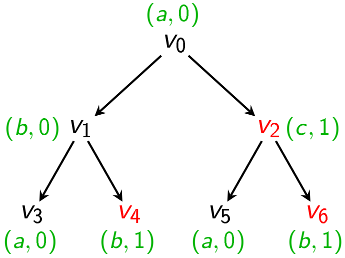
Step 1 – Tree Automata as Query Mechanisms

Let $T = (V, E)$ be a tree and let $S \subseteq V$.

mark(T, S): mark all nodes from S in T.
(formally, label a of node v is replaced by either $(a, 0)$ or $(a, 1)$.)



Tree T



mark$(T, \{v_2, v_4, v_6\})$

Step 1 – Tree Automata as Query Mechanisms

A tree automaton $\mathcal{A}$ is **node selecting** if it accepts marked trees, i. e., trees over the alphabet $\Sigma \times \{0, 1\}$.

Step 1 – Tree Automata as Query Mechanisms

A tree automaton $\mathcal{A}$ is **node selecting** if it accepts marked trees, i. e., trees over the alphabet $\Sigma \times \{0, 1\}$.

For a node selecting tree automaton $\mathcal{A}$, we define:

$$\text{select}(\mathcal{A}, T) := \{S \subseteq V \mid \text{mark}(T, S) \in L(\mathcal{A})\}$$

Step 1 – Tree Automata as Query Mechanisms

A tree automaton $\mathcal{A}$ is **node selecting** if it accepts marked trees, i. e., trees over the alphabet $\Sigma \times \{0,1\}$.

For a node selecting tree automaton $\mathcal{A}$, we define:

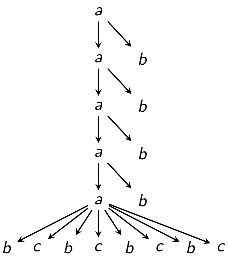
$$\text{select}(\mathcal{A}, T) := \{S \subseteq V \mid \text{mark}(T, S) \in L(\mathcal{A})\}$$

Carme, Niehren, Tommasi, 2004

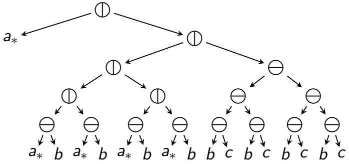
From a given MSO-formula $\Phi(X)$ one can construct a node selecting nondeterministic stepwise tree automaton (nSTA) $\mathcal{A}_\Phi$ such that for every forest F:

$$\text{select}(\mathcal{A}_\Phi, F) = \text{select}(\Phi(X), F)$$

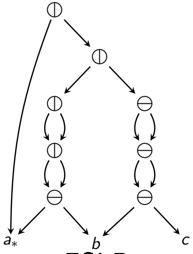
Step 1 – dBUTAs Over DAG-Foldings of Binary Trees



Forest

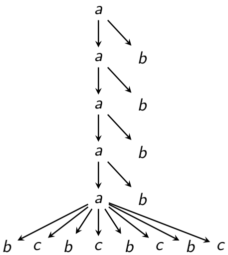


Forest algebra expression

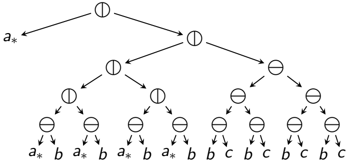


FSLP

Step 1 – dBUTAs Over DAG-Foldings of Binary Trees



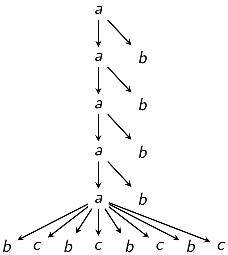
Forest



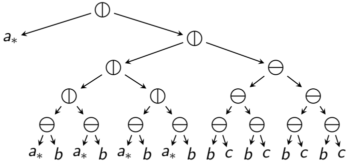
Forest algebra expression

FSLP

Step 1 – dBUTAs Over DAG-Foldings of Binary Trees



Forest



Forest algebra expression

FSLP

Kleest-Meißner, Marasus, Niewerth, 2022

From an nSTA $\mathcal{A}$ working on forests one can construct a deterministic bottom-up tree automaton (dBUTA) $\mathcal{B}$ working on forest algebra expressions with $L(\mathcal{B}) = \{E : \mathrm{val}(E) \in L(\mathcal{A})\}$.

Step 1 – dBUTAs Over DAG-Foldings of Binary Trees

We have reduced our problem to the following task:

After linear preprocessing time,
enumerate select($\mathcal{B}, F$) with output-linear delay,

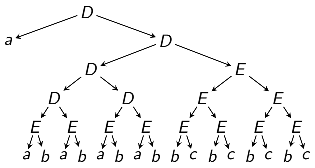
Step 1 – dBUTAs Over DAG-Foldings of Binary Trees

We have reduced our problem to the following task:

After linear preprocessing time,
enumerate select($\mathcal{B}, F$) with output-linear delay,

where $\mathcal{B}$ is a fixed leaf selecting dBUTA and...

... F is a binary tree...



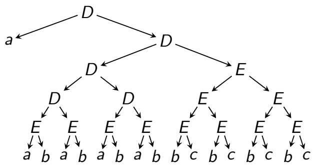
Step 1 – dBUTAs Over DAG-Foldings of Binary Trees

We have reduced our problem to the following task:

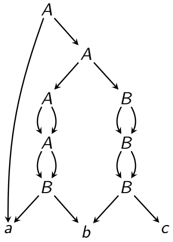
After linear preprocessing time,
enumerate select($\mathcal{B}$, F) with output-linear delay,

where $\mathcal{B}$ is a fixed leaf selecting dBUTA and...

...F is a binary tree...



...but given as its DAG folding!



Bagan's Algorithm

For explicit trees, the problem can be solved by Bagan's algorithm:

Theorem — Bagan 2006

For a fixed leaf-selecting dBUTA $\mathcal{B}$ and a binary node-labelled tree T, after a preprocessing in time $O(|T|)$, we can enumerate $\text{select}(\mathcal{B}, T)$ with output linear delay.

Bagan's Algorithm

For explicit trees, the problem can be solved by Bagan's algorithm:

Theorem — Bagan 2006

For a fixed leaf-selecting dBUTA $\mathcal{B}$ and a binary node-labelled tree T, after a preprocessing in time $O(|T|)$, we can enumerate $\text{select}(\mathcal{B}, T)$ with output linear delay.

$\rightsquigarrow$ Step 2 – Extend Bagan's algorithm to the DAG-compressed setting

Step 2 – Deterministic Bottom-Up Tree Automata

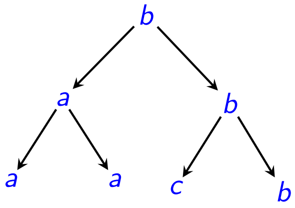
Leaf-rules: $a \to q$ for label a and state q

Branching-rules: $(r, p, a) \to q$ for label a and states r, p, q

Step 2 – Deterministic Bottom-Up Tree Automata

Leaf-rules: $a \to q$ for label a and state q

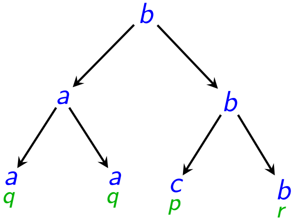
Branching-rules: $(r, p, a) \to q$ for label a and states r, p, q



Step 2 – Deterministic Bottom-Up Tree Automata

Leaf-rules: $a \rightarrow q$ for label a and state q

Branching-rules: $(r, p, a) \rightarrow q$ for label a and states r, p, q

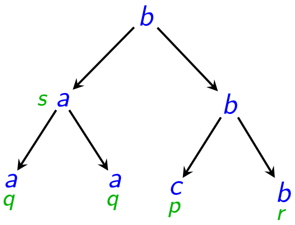


$a \rightarrow q$
$b \rightarrow r$
$c \rightarrow p$

Step 2 – Deterministic Bottom-Up Tree Automata

Leaf-rules: $a \to q$ for label a and state q

Branching-rules: $(r, p, a) \to q$ for label a and states r, p, q



$a \to q$

$b \to r$

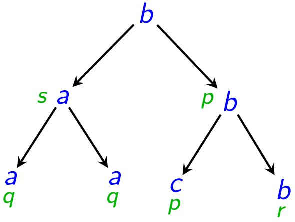
$c \to p$

$(q, q, a) \to s$

Step 2 – Deterministic Bottom-Up Tree Automata

Leaf-rules: $a \rightarrow q$ for label a and state q

Branching-rules: $(r, p, a) \rightarrow q$ for label a and states r, p, q



$a \rightarrow q$

$b \rightarrow r$

$c \rightarrow p$

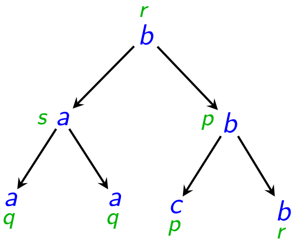
$(q, q, a) \rightarrow s$

$(p, r, c) \rightarrow p$

Step 2 – Deterministic Bottom-Up Tree Automata

Leaf-rules: $a \to q$ for label a and state q

Branching-rules: $(r, p, a) \to q$ for label a and states r, p, q



$a \to q$

$b \to r$

$c \to p$

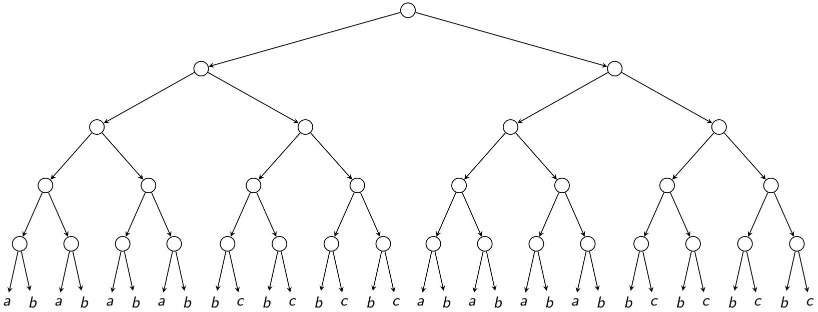
$(q, q, a) \to s$

$(p, r, c) \to p$

$(s, p, b) \to r$

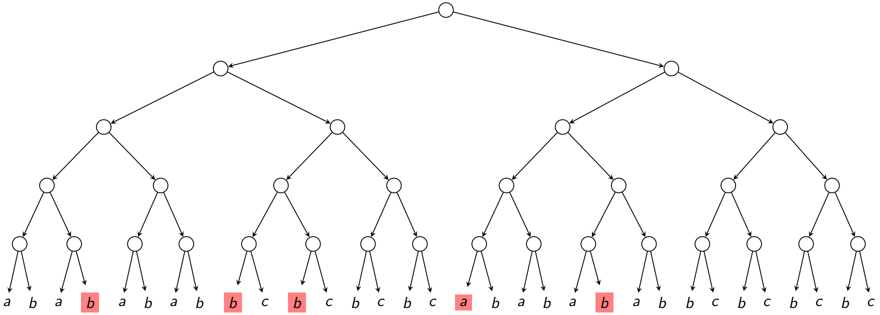
Step 2 – Bagan's Algorithm

Leaf-labelled tree T:



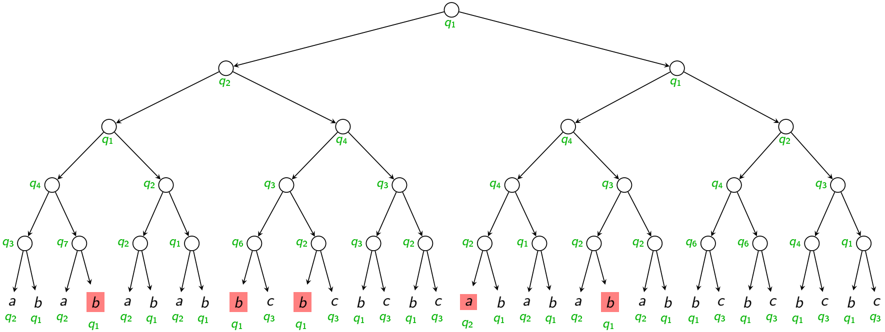
Step 2 – Bagan's Algorithm

Marked tree mark(T, S) for leaf-set S:



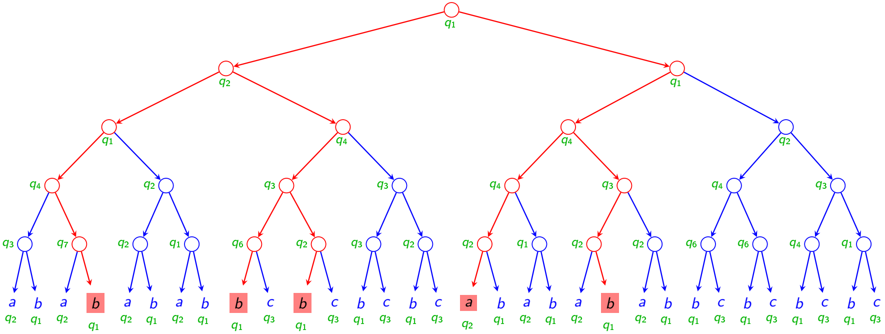
Step 2 – Bagan's Algorithm

Run on mark(T, S):

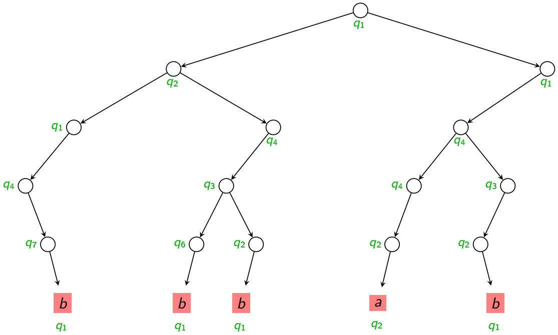


Step 2 – Bagan's Algorithm

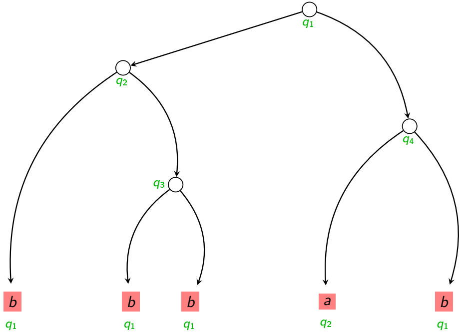
Run on mark(T, S):



Step 2 – Bagan's Algorithm

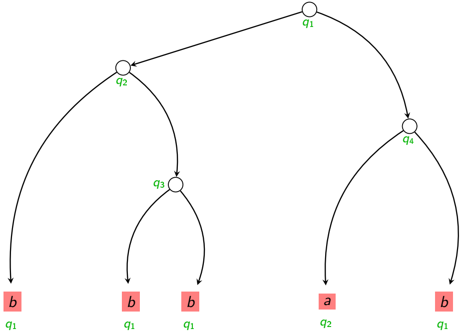


Step 2 – Bagan's Algorithm



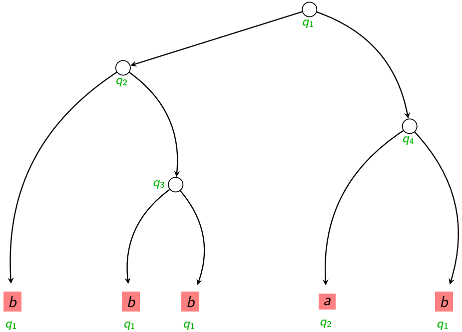
Step 2 – Bagan's Algorithm

Witness tree for leaf-set S:



Step 2 – Bagan's Algorithm

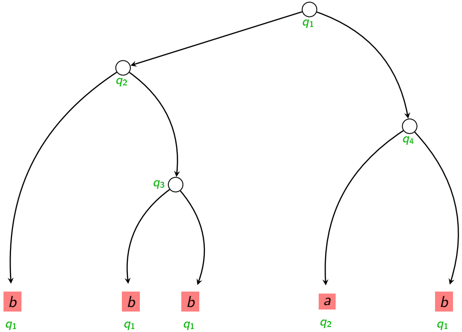
Witness tree for leaf-set S:



Main idea: Enumerate all witness trees.

Step 2 – Bagan's Algorithm

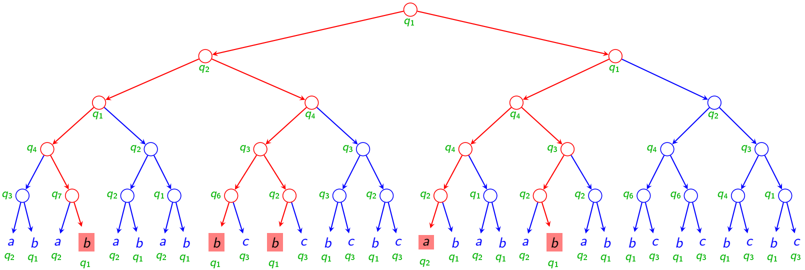
Witness tree for leaf-set S:



Main idea: Enumerate all witness trees.

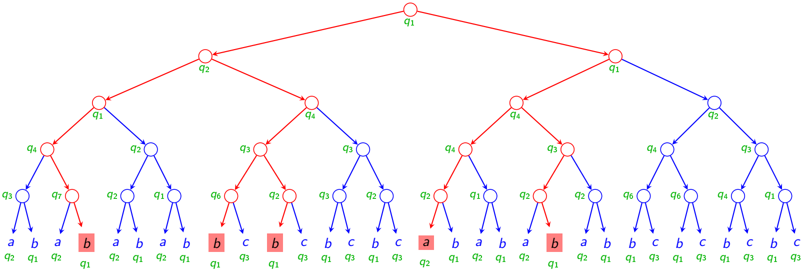
But how to do that?

Step 2 – Bagan's Algorithm



A **configuration** $(v, q) \in V \times Q$ is. . .

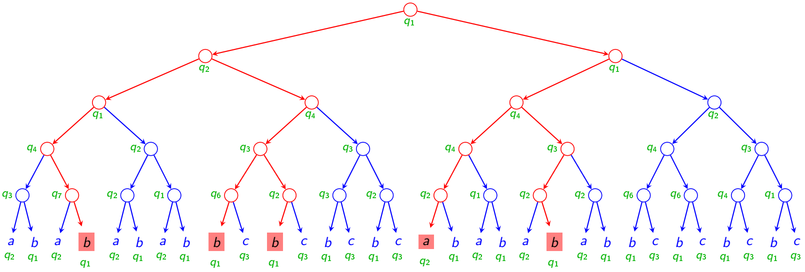
Step 2 – Bagan's Algorithm



A **configuration** $(v, q) \in V \times Q$ is...

...**active** wrt mark(T, S) if it is red in the run on mark(T, S).

Step 2 – Bagan's Algorithm

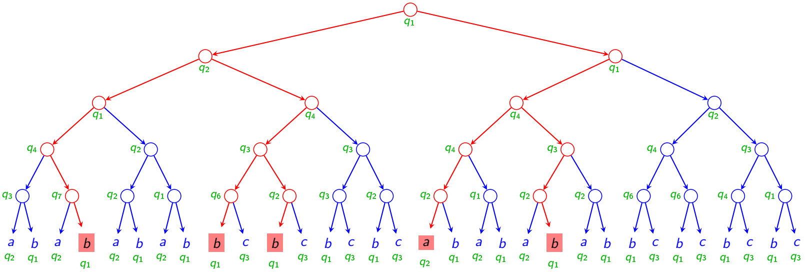


A **configuration** $(v, q) \in V \times Q$ is...

...**active** wrt mark(T, S) if it is red in the run on mark(T, S).

...**useful** wrt mark(T, S) if it is a red leaf-configuration or a red configuration with two red children in the run on mark(T, S).

Step 2 – Bagan's Algorithm



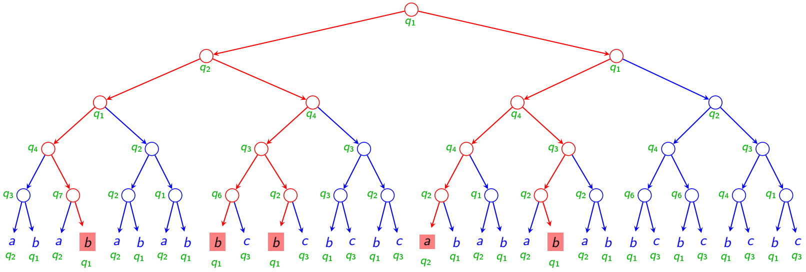
A **configuration** $(v, q) \in V \times Q$ is...

...**active** wrt mark(T, S) if it is red in the run on mark(T, S).

...**useful** wrt mark(T, S) if it is a red leaf-configuration or a red configuration with two red children in the run on mark(T, S).

...**active/useful** (in general) if it is **active/useful** wrt some mark(T, S).

Step 2 – Bagan's Algorithm



A **configuration** $(v, q) \in V \times Q$ is...

...**active** wrt mark(T, S) if it is red in the run on mark(T, S).

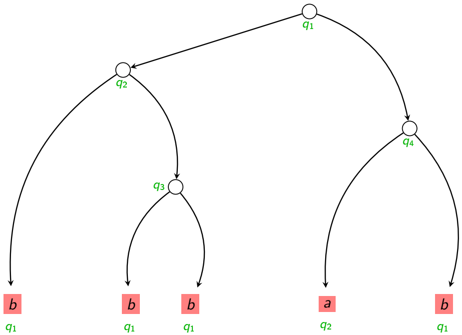
...**useful** wrt mark(T, S) if it is a red leaf-configuration or a red configuration with two red children in the run on mark(T, S).

...**active/useful** (in general) if it is **active/useful** wrt some mark(T, S).

...**nullable** if it is blue in the run on some mark(T, S).

Step 2 – Bagan's Algorithm

Top-down construction of witness trees by appending **useful** configurations:



Step 2 – Bagan's Algorithm

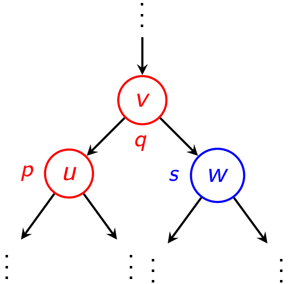
Compute a binary relation $\vdash$ on **active** configurations:

$(v, q) \vdash (u, p) \iff \exists$ **nullable** (w, s) and some mark(T, S) with a run with

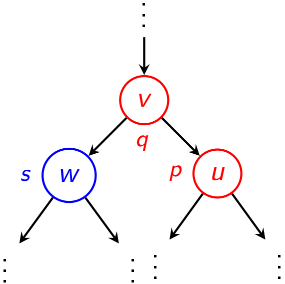
Step 2 – Bagan's Algorithm

Compute a binary relation $\vdash$ on **active** configurations:

$(v, q) \vdash (u, p) \iff \exists$ **nullable** (w, s) and some mark(T, S) with a run with



OR



Step 2 – Bagan's Algorithm

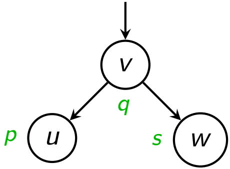
Shortcut forest: ("active configurations", $\vdash$).

Step 2 – Bagan's Algorithm

Shortcut forest: ("active configurations", $\vdash$).

Use shortcut forest for witness tree construction:

Assume (v, q) is a useful configuration of a witness tree and

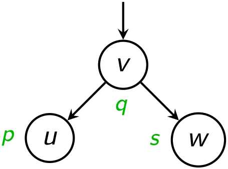


Step 2 – Bagan's Algorithm

Shortcut forest: (“active configurations”, $\vdash$).

Use shortcut forest for witness tree construction:

Assume (v, q) is a useful configuration of a witness tree and



Let (u', p') be useful and $(u, p) \vdash^* (u', p')$.

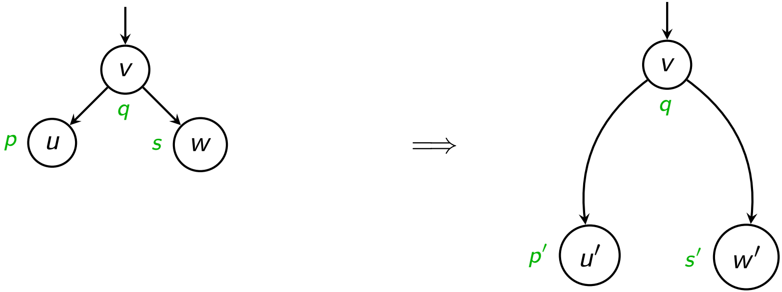
Let (w', s') be useful and $(w, s) \vdash^* (w', s')$

Step 2 – Bagan's Algorithm

Shortcut forest: (“active configurations”, $\vdash$).

Use shortcut forest for witness tree construction:

Assume (v, q) is a useful configuration of a witness tree and



Let (u', p') be useful and $(u, p) \vdash^* (u', p')$.

Let (w', s') be useful and $(w, s) \vdash^* (w', s')$

Step 2 – Bagan's Algorithm for DAG-Compressed Trees

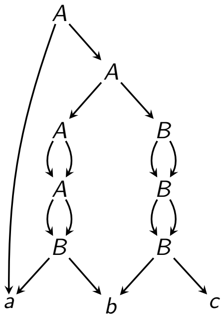
Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Question: How are tree leaves represented in DAG?

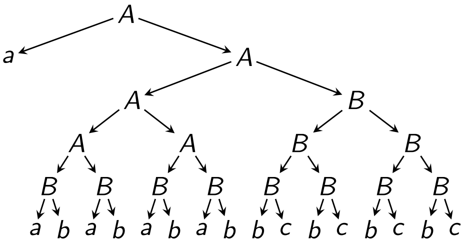
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Question: How are tree leaves represented in DAG?



DAG-folding of the tree.

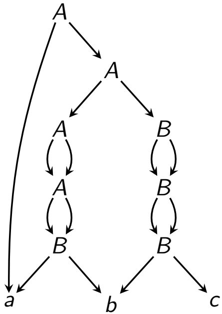


Binary tree.

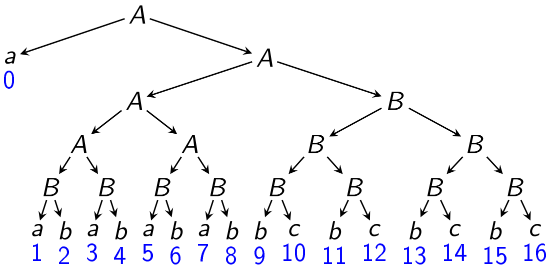
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Question: How are tree leaves represented in DAG?



DAG-folding of the tree.

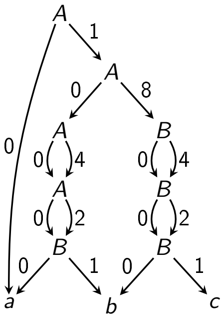


Binary tree.

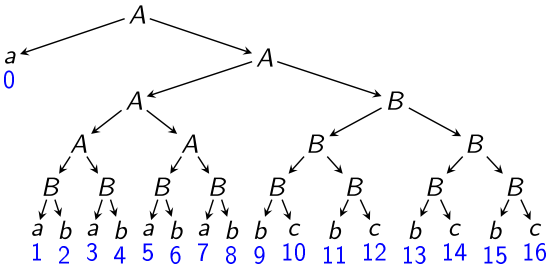
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Question: How are tree leaves represented in DAG?



DAG-folding of the tree.

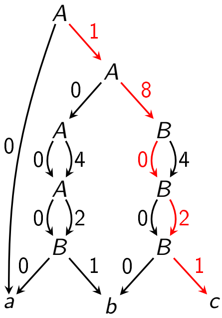


Binary tree.

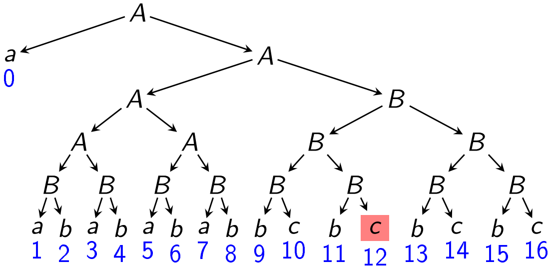
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Question: How are tree leaves represented in DAG?



DAG-folding of the tree.

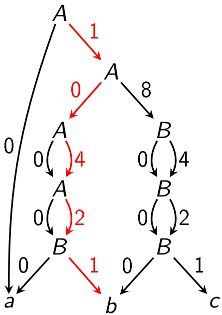


Binary tree.

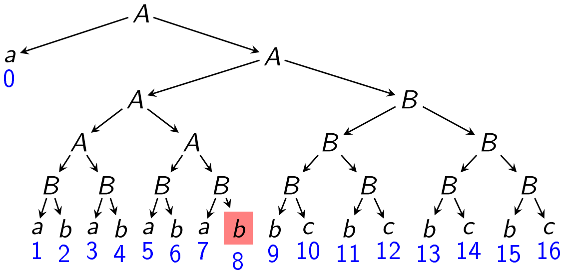
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Question: How are tree leaves represented in DAG?



DAG-folding of the tree.

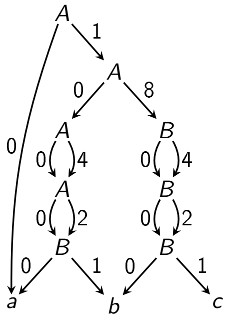


Binary tree.

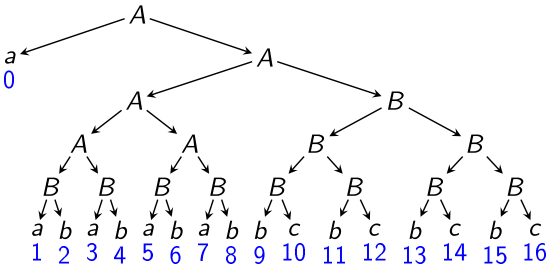
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Question: How are tree leaves represented in DAG?



DAG-folding of the tree.

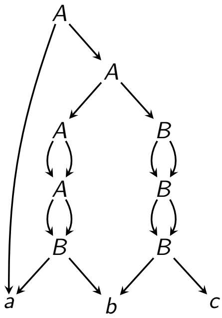


Binary tree.

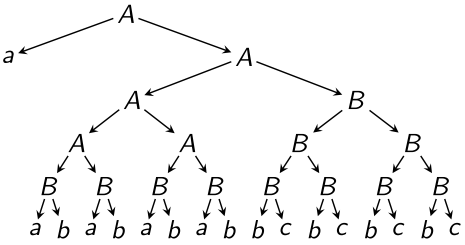
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Observation: We can run dBUTA directly on DAG.



DAG-folding of the tree.

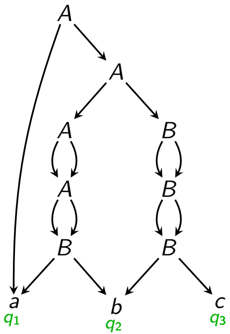


Binary tree.

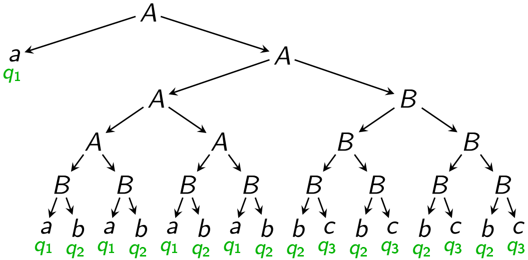
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Observation: We can run dBUTA directly on DAG.



DAG-folding of the tree.

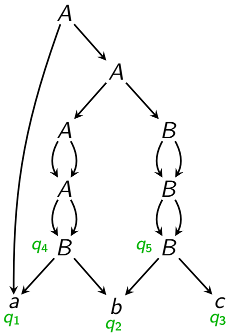


Binary tree.

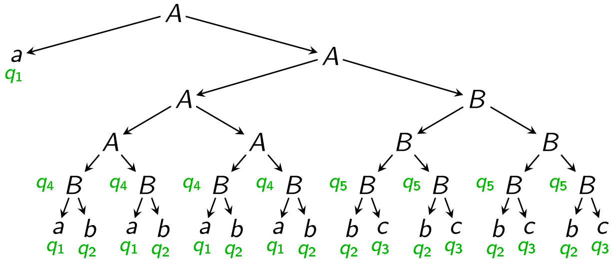
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Observation: We can run dBUTA directly on DAG.



DAG-folding of the tree.

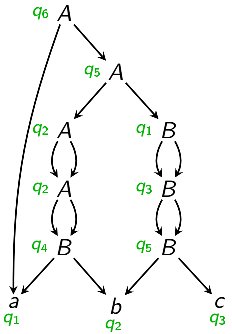


Binary tree.

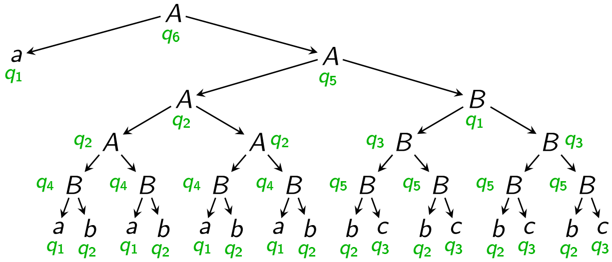
Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Goal: Run Bagan's algorithm on the DAG-folding of the tree.

Observation: We can run dBUTA directly on DAG.



DAG-folding of the tree.



Binary tree.

Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Problem: We cannot afford to compute the full shortcut forest

Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

Problem: For a given (u, p), we cannot afford to explicitly compute all useful (u', p') with $(u, p) \vdash^* (u', p')$.

Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

Problem: For a given (u, p), we cannot afford to explicitly compute all useful (u', p') with $(u, p) \vdash^* (u', p')$.

Solution: For a given (u, p), we can efficiently **enumerate** all useful (u', p') with $(u, p) \vdash^* (u', p')$.

Step 2 – Bagan's Algorithm for DAG-Compressed Trees

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

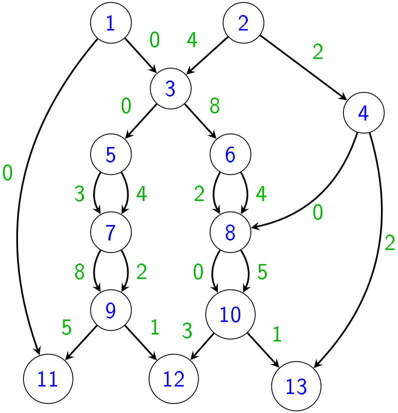
Problem: For a given (u, p), we cannot afford to explicitly compute all useful (u', p') with $(u, p) \vdash^* (u', p')$.

Solution: For a given (u, p), we can efficiently **enumerate** all useful (u', p') with $(u, p) \vdash^* (u', p')$.

This boils down to the following path enumeration problem in DAGs.

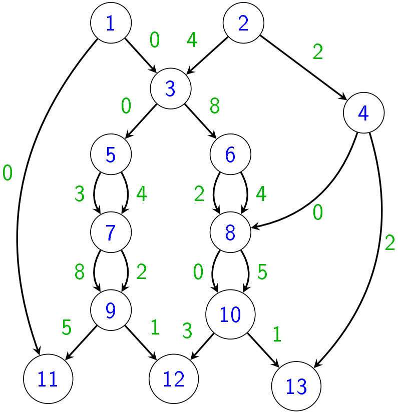
Step 3 – Path Enumeration in DAGs

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.



Step 3 – Path Enumeration in DAGs

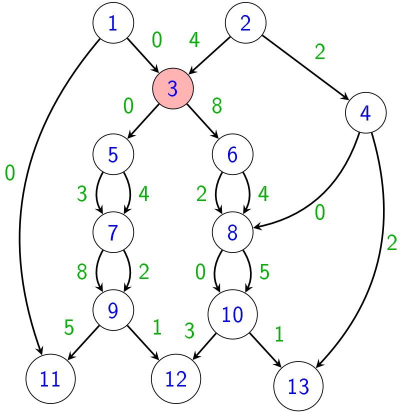
Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.



Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths π from s to some leaf u represented by $(u, \gamma(\pi))$.

Step 3 – Path Enumeration in DAGs

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.

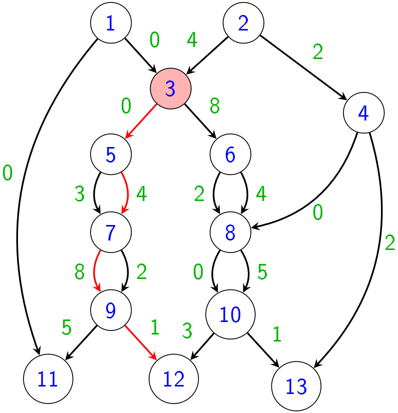


Enumeration for start node 3:

Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths π from s to some leaf u represented by $(u, \gamma(\pi))$.

Step 3 – Path Enumeration in DAGs

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.

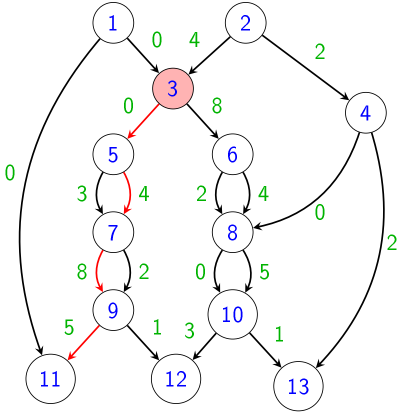


Enumeration for start node 3:
$(12, 13)$

Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths π from s to some leaf u represented by $(u, \gamma(\pi))$.

Step 3 – Path Enumeration in DAGs

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.

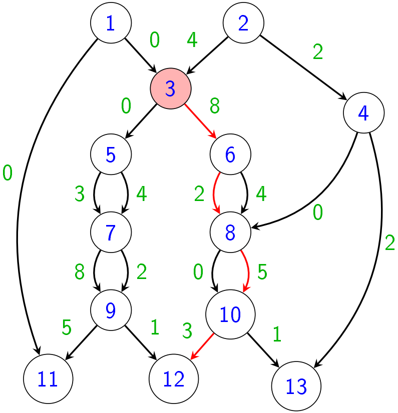


Enumeration for start node 3:
$(12, 13), (11, 17)$

Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths π from s to some leaf u represented by $(u, \gamma(\pi))$.

Step 3 – Path Enumeration in DAGs

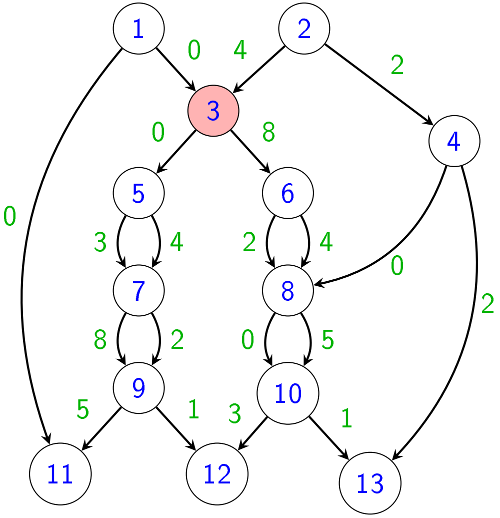
Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.



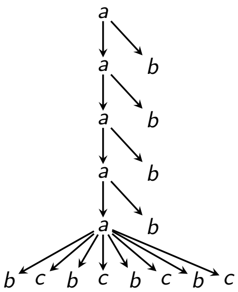
Enumeration for start node 3:
$(12, 13), (11, 17), (12, 18), \ldots$

Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths π from s to some leaf u represented by $(u, \gamma(\pi))$.

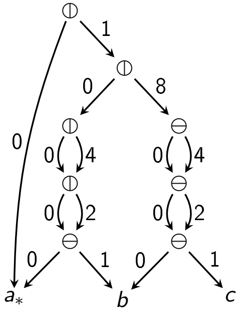
Step 3 – Path Enumeration in DAGs



Additional Aspects – Representation of Nodes

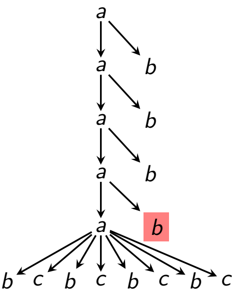


Forest

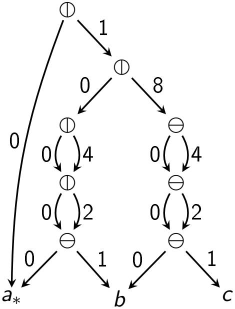


FSLP

Additional Aspects – Representation of Nodes

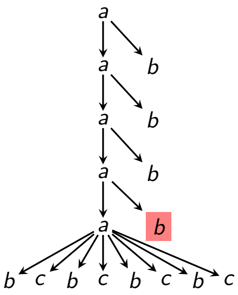


Forest

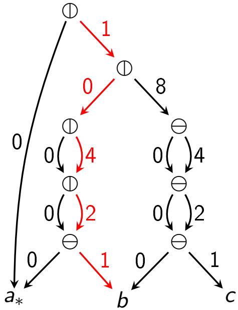


FSLP

Additional Aspects – Representation of Nodes

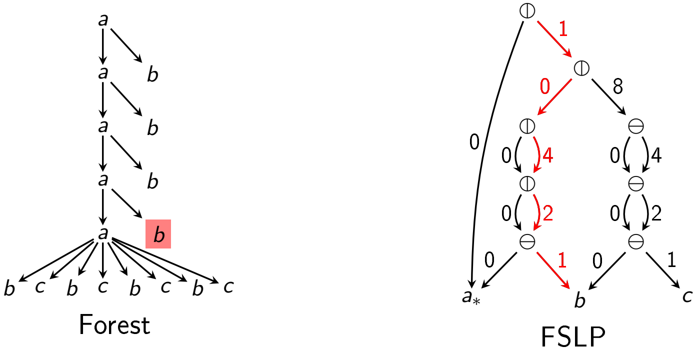


Forest



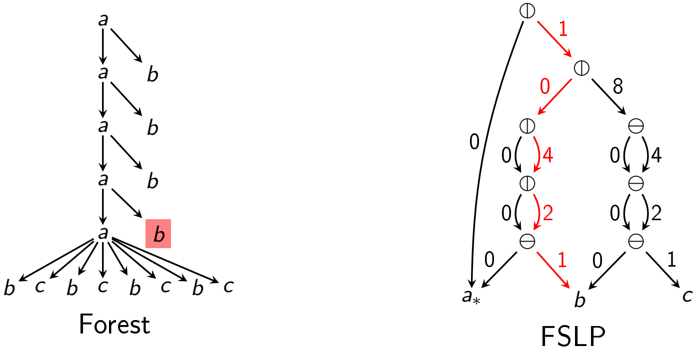
FSLP

Additional Aspects – Representation of Nodes



$\rightsquigarrow$ representation of nodes depends on structure of FSLPs!

Additional Aspects – Representation of Nodes



$\rightsquigarrow$ representation of nodes depends on structure of FSLPs!

representation by preorder numbers is also possible (by using edge weights from a complicated monoid).

Additional Aspects – Relabelling Updates

Relabelling updates:

Let F be a forest, v a node of F and x some label.

Additional Aspects – Relabelling Updates

Relabelling updates:

Let F be a forest, v a node of F and x some label.

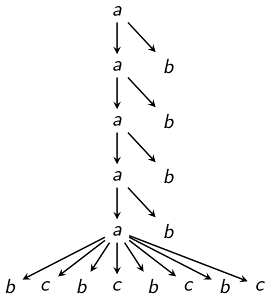
relabel(F, v, x): The forest F with node v relabelled to x.

Additional Aspects – Relabelling Updates

Relabelling updates:

Let F be a forest, v a node of F and x some label.

relabel(F, v, x): The forest F with node v relabelled to x.



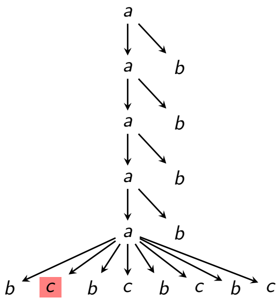
Forest

Additional Aspects – Relabelling Updates

Relabelling updates:

Let F be a forest, v a node of F and x some label.

relabel(F, v, x): The forest F with node v relabelled to x.



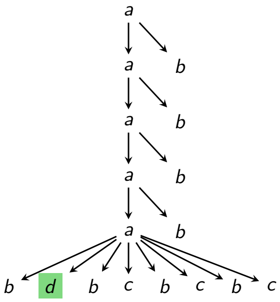
Forest

Additional Aspects – Relabelling Updates

Relabelling updates:

Let F be a forest, v a node of F and x some label.

relabel(F, v, x): The forest F with node v relabelled to x.



Forest

Additional Aspects – Relabelling Updates

Maintaining relabelling updates in the FSLP-compressed setting:

Carry out the linear preprocessing wrt. FSLP $\mathcal{G}$.

Enumerate the query result w.r.t. $F := \mathsf{val}(\mathcal{G})$ with output linear delay.

Additional Aspects – Relabelling Updates

Maintaining relabelling updates in the FSLP-compressed setting:

Carry out the linear preprocessing wrt. FSLP $\mathcal{G}$.

Enumerate the query result w.r.t. $F := \mathsf{val}(\mathcal{G})$ with output linear delay.

Update data $F' := \mathsf{relabel}(F, v, x)$.

Additional Aspects – Relabelling Updates

Maintaining relabelling updates in the FSLP-compressed setting:

Carry out the linear preprocessing wrt. FSLP $\mathcal{G}$.

Enumerate the query result w.r.t. $F := \mathsf{val}(\mathcal{G})$ with output linear delay.

Update data $F' := \mathsf{relabel}(F, v, x)$.

Enumerate the query result w.r.t. F' with output-linear delay.

Additional Aspects – Relabelling Updates

Maintaining relabelling updates in the FSLP-compressed setting:

Carry out the linear preprocessing wrt. FSLP $\mathcal{G}$.

Enumerate the query result w.r.t. $F := \mathrm{val}(\mathcal{G})$ with output linear delay.

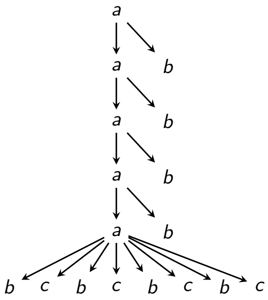
Update data $F' := \mathrm{relabel}(F, v, x)$.

Enumerate the query result w.r.t. F' with output-linear delay.

Theorem

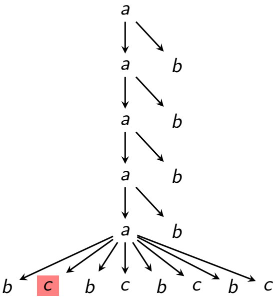
We can maintain relabelling updates in the FSLP-compressed setting in time $\mathrm{O}(\log(|F|))$, where the relabelled node is given by its preorder number w.r.t. F.

Additional Aspects – Relabelling Updates



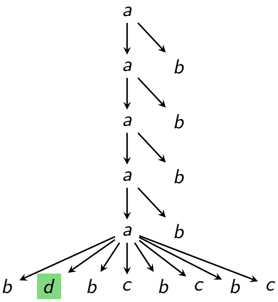
Forest

Additional Aspects – Relabelling Updates



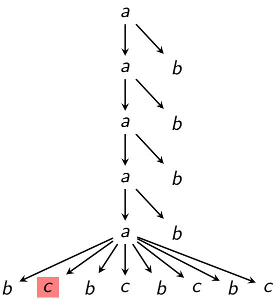
Forest

Additional Aspects – Relabelling Updates

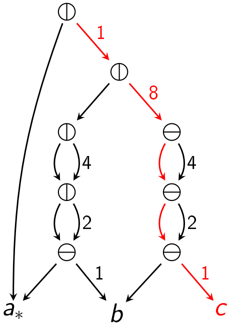


Forest

Additional Aspects – Relabelling Updates

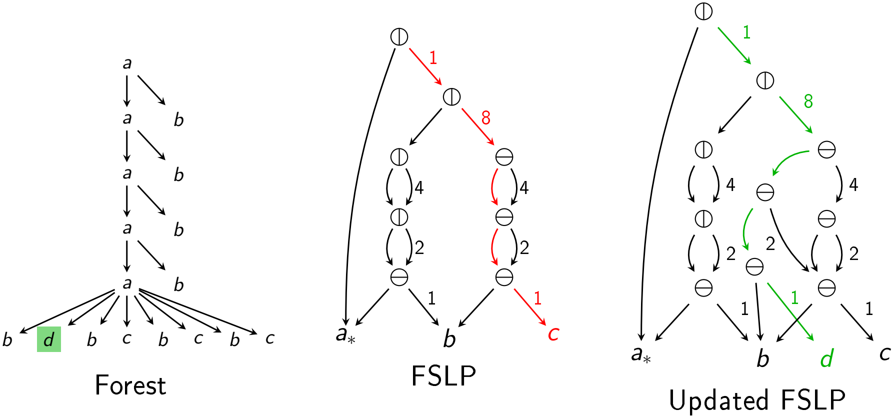


Forest

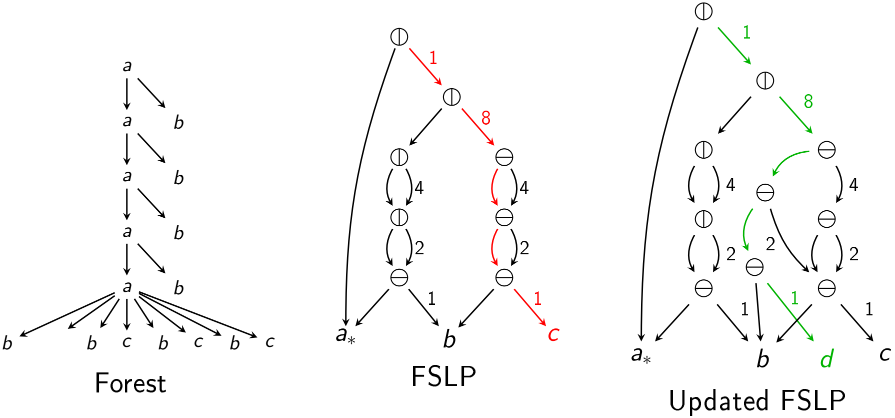


FSLP

Additional Aspects – Relabelling Updates

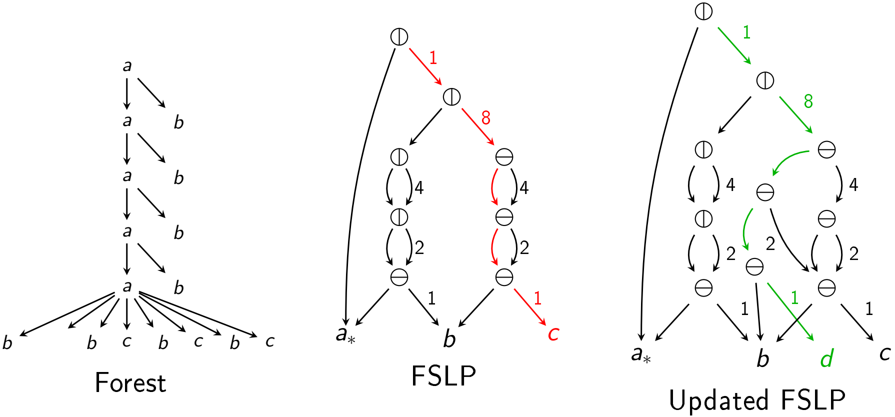


Additional Aspects – Relabelling Updates



$\rightsquigarrow$ Running time: Height of FSLP.

Additional Aspects – Relabelling Updates



$\rightsquigarrow$ Running time: Height of FSLP.

Height can be bounded by the FSLP balancing theorem:

Theorem (Ganardi, Jez, Lohrey 2021)

FSLPs can be balanced in linear time.

Thank you very much for your attention.