Enumerating regular languages with bounded delay

Antoine Amarilli, Mikaël Monet; STACS'23

July 27, 2023

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input $\rightarrow$ YES/NO

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input $\rightarrow$ YES/NO

Measure: running time

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input → YES/NO

Measure: running time

Computation problem

Input → { ■ ▼ , ■ ■ ▼ , ■ ▼ ▼ }

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input $\rightarrow$ YES/NO

Measure: running time

Computation problem

Input $\rightarrow$ $\{ \blacksquare \blacktriangledown, \blacksquare \blacksquare \blacktriangledown, \blacksquare \blacktriangledown \blacktriangledown \}$

Measure: running time

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input $\rightarrow$ YES/NO | Measure: running time

Computation problem

Input $\rightarrow$  | Measure: running time

Enumeration problem

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input → YES/NO Measure: running time

Computation problem

Input →  Measure: running time

Enumeration problem

Input →

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input $\rightarrow$ YES/NO — Measure: running time

Computation problem

Input $\rightarrow$ $\{\ \blacksquare\ \blacktriangledown\ ,\ \blacksquare\ \blacksquare\ \blacktriangledown\ ,\ \blacksquare\ \blacktriangledown\ \blacktriangledown\ \}$ — Measure: running time

Enumeration problem

Input $\rightarrow$ $\{\ \blacksquare\ \blacktriangledown\ ,$

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input $\rightarrow$ YES/NO

Measure: running time

Computation problem

Input $\rightarrow$ $\{ \blacksquare \blacktriangledown, \blacksquare \blacksquare \blacktriangledown, \blacksquare \blacktriangledown \blacktriangledown \}$

Measure: running time

Enumeration problem

Input $\rightarrow$ $\{ \blacksquare \blacktriangledown,$
$\blacksquare \blacksquare \blacktriangledown,$

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input → YES/NO — Measure: running time

Computation problem

Input → { ■ ▼ , ■ ■ ▼ , ■ ▼ ▼ } — Measure: running time

Enumeration problem

Input → { ■ ▼ ,
■ ■ ▼ ,
■ ▼ ▼ }

Context: Enumeration algorithms

Enumeration algorithms: framework for computation problems producing many results

Decision problem

Input $\rightarrow$ YES/NO — Measure: running time

Computation problem

Input $\rightarrow$ $\{\ \blacksquare\ \blacktriangledown\ ,\ \blacksquare\ \blacksquare\ \blacktriangledown\ ,\ \blacksquare\ \blacktriangledown\ \blacktriangledown\ \}$ — Measure: running time

Enumeration problem

Input $\rightarrow$ $\{\ \blacksquare\ \blacktriangledown\ ,$
$\blacksquare\ \blacksquare\ \blacktriangledown\ ,$
$\blacksquare\ \blacktriangledown\ \blacktriangledown\ \}$

Measure: max delay between two consecutive results

Holy grail: Constant-delay enumeration

Holy grail: Constant-delay enumeration

On acyclic conjunctive queries and constant delay enumeration

Guillaume Bagan [*] Arnaud Durand [†] Etienne Grandjean [‡]

Holy grail: Constant-delay enumeration

On acyclic conjunctive queries and constant delay enumeration

Guillau

Enumeration of MSO Queries on Strings with Constant Delay and Logarithmic Updates

Matthias Niewerth
University of Bayreuth

Luc Segoufin
INRIA and ENS Ulm

Holy grail: Constant-delay enumeration

On acyclic conjunctive queries and constant delay enumeration

Guilla

Enumeration of MSO Queries on Strings with Constant Delay and Logarithmic Updates

M
U

Constant delay enumeration for FO queries over databases with local bounded expansion

Luc Segoufin
INRIA and ENS Ulm
Paris

Alexandre Vigny
Université Paris Diderot Paris 7
Paris

Holy grail: Constant-delay enumeration

On acyclic c

Guillau

Enu

Constant Delay and Logarithmic Updates

M

U

A glimpse on constant delay enumeration

Luc Segoufin

INRIA and ENS Cachan

Constant delay enumeration for FO queries over databases with local bounded expansion

Luc Segoufin
INRIA and ENS Ulm
Paris

Alexandre Vigny
Université Paris Diderot Paris 7
Paris

Holy grail: Constant-delay enumeration

On acyclic c

A glimpse on constant delay enumeration

Luc Segoufin

INRIA and ENS Cachan

Guillau

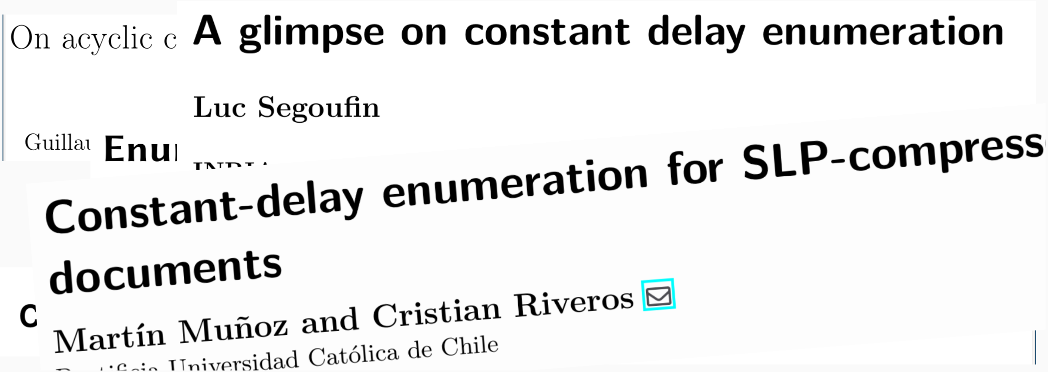
Enu

Constant Delay and Logarithmic Updates

N

Constant delay enumeration for FO queries over databases with local bounded expansion

Constant Delay Enumeration for Conjunctive Queries — a Tutorial


On acyclic c

Guillau

Enu

C

A glimpse on constant delay enumeration

Luc Segoufin

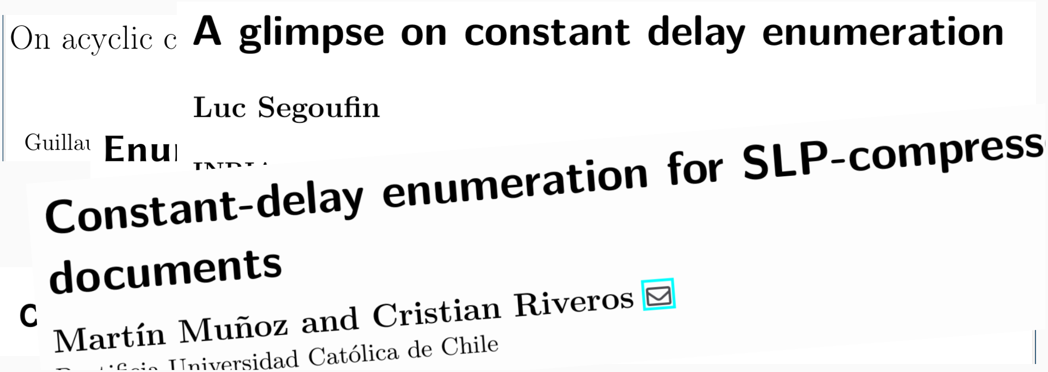
Constant-delay enumeration for SLP-compress

documents

Martín Muñoz and Cristian Riveros

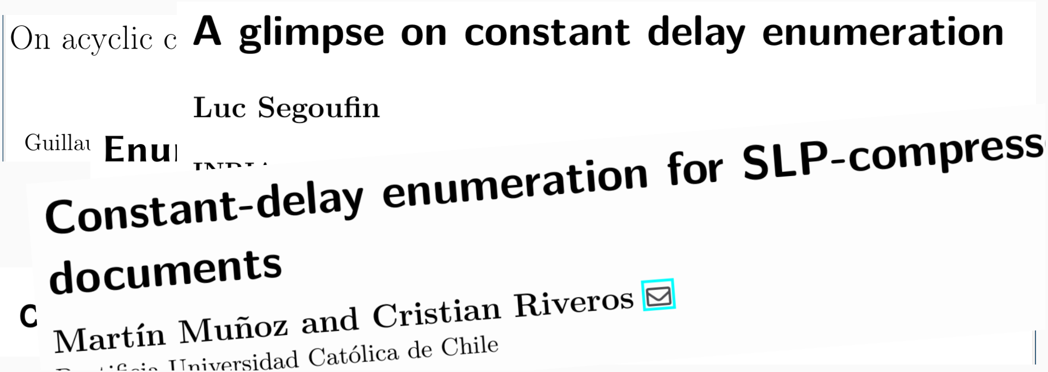
Pontificia Universidad Católica de Chile

Holy grail: Constant-delay enumeration



Problem: assumes that the results to enumerate have constant size

Holy grail: Constant-delay enumeration



Problem: assumes that the results to enumerate have constant size

Ambitious goal

How can we enumerate results of unbounded size in constant delay?

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

____	____	____	____	____	$\cdots$	Results:
1	2	3	4	5		

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

____ ____ ____ ____ ____ . . . Results:

1 2 3 4 5

Put(1, );

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, );

Solution: Cheat!

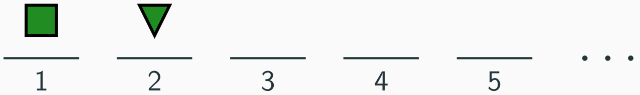
Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼);

Solution: Cheat!

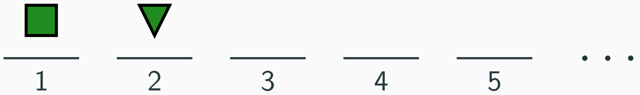
Do not write each result from scratch, but by editing the previous result!



Results:

Put(1, ■); Put(2, ▼);

Do not write each result from scratch, but by editing the previous result!

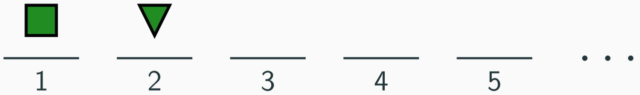


Results:

Put(1, ■); Put(2, ▼); Output();

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!



Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Put(3, ▼);

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Put(3, ▼);

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Put(3, ▼); Output();

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Put(3, ▼); Output();

■ ▼ ▼

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Put(3, ▼); Output();

■ ▼ ▼

Put(2, ■);

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

■	■	▼			
1	2	3	4	5	. . .

Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Put(3, ▼); Output();

■ ▼ ▼

Put(2, ■);

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼); Output();

■ ▼

Put(3, ▼); Output();

■ ▼ ▼

Put(2, ■); Output();

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

Results:

Put(1, ■); Put(2, ▼); Output(); ■ ▼

Put(3, ▼); Output(); ■ ▼ ▼

Put(2, ■); Output(); ■ ■ ▼

Solution: Cheat!

Do not write each result from scratch, but by editing the previous result!

■	■	▼			
1	2	3	4	5	. . .

Results:

Put(1, ■); Put(2, ▼); Output(); — ■ ▼

Put(3, ▼); Output(); — ■ ▼ ▼

Put(2, ■); Output(); — ■ ■ ▼

Remark: Solutions need to be ordered with small distance between consecutive solutions

Enumeration for automata

Problem: enumerate the words accepted by a word automaton

Enumeration for automata

Problem: enumerate the words accepted by a word automaton

- Input: Deterministic finite automaton A on alphabet Σ
- Output: The words of its language $\mathrm{L}(A) \subseteq \Sigma^*$

Enumeration for automata

Problem: enumerate the words accepted by a word automaton

- Input: Deterministic finite automaton A on alphabet Σ
- Output: The words of its language $\mathrm{L}(A) \subseteq \Sigma^*$

We want to produce each word by editing the previous word.

Enumeration for automata

Problem: enumerate the words accepted by a word automaton

- Input: Deterministic finite automaton A on alphabet Σ
- Output: The words of its language $\mathrm{L}(A) \subseteq \Sigma^*$

We want to produce each word by editing the previous word. Questions:

Enumeration for automata

Problem: enumerate the words accepted by a word automaton

- Input: Deterministic finite automaton A on alphabet Σ
- Output: The words of its language $\mathrm{L}(A) \subseteq \Sigma^*$

We want to produce each word by editing the previous word. Questions:

- Can we find a distance bound $C \in \mathbb{N}$ and order $\mathrm{L}(A) = \{w_1, w_2, \ldots\}$ such that $d(w_i, w_{i+1}) \leq C$ for all $i \geq 1$?
  - Here, d is the Levenshtein distance

Enumeration for automata

Problem: enumerate the words accepted by a word automaton

- Input: Deterministic finite automaton A on alphabet Σ
- Output: The words of its language $\mathrm{L}(A) \subseteq \Sigma^*$

We want to produce each word by editing the previous word. Questions:

- Can we find a distance bound $C \in \mathbb{N}$ and order $\mathrm{L}(A) = \{w_1, w_2, \ldots\}$ such that $d(w_i, w_{i+1}) \leq C$ for all $i \geq 1$?
  - Here, d is the Levenshtein distance
- If yes, can we efficiently produce the sequence of edits?

Examples

a^*

Examples

a^* $\epsilon,\ a,\ aa,\ aaa,\ \ldots$

Examples

a^* $\epsilon,\ a,\ aa,\ aaa,\ \ldots$

$a^* b^*$

Examples

a^*	$\epsilon, a, aa, aaa, \ldots$
$a^* b^*$	$\epsilon, a, b,$

Examples

a^*	$\epsilon,\ a,\ aa,\ aaa,\ \ldots$
$a^* b^*$	$\epsilon,\ a,\ b,\ bb,\ ab,\ aa,$

Examples

a^* $\epsilon, a, aa, aaa, \ldots$

$a^* b^*$ $\epsilon, a, b, bb, ab, aa, aaa, aab, abb, bbb, \ldots$

Examples

a^* $\quad$ $\epsilon, a, aa, aaa, \ldots$

a^*b^* $\quad$ $\epsilon, a, b, bb, ab, aa, aaa, aab, abb, bbb, \ldots$

$a^*(c+d)b^*$

Examples

a^*	ϵ, a, aa, aaa, ...
$a^* b^*$	ϵ, a, b, bb, ab, aa, aaa, aab, abb, bbb, ...
$a^*(c+d)b^*$	c, d,

Examples

a^*	$\epsilon, a, aa, aaa, \ldots$
$a^* b^*$	$\epsilon, a, b, bb, ab, aa, aaa, aab, abb, bbb, \ldots$
$a^*(c + d)b^*$	$c, d, ac, ad, cb, db,$

Examples

a^*	ϵ, a, aa, aaa, ...
$a^* b^*$	ϵ, a, b, bb, ab, aa, aaa, aab, abb, bbb, ...
$a^*(c+d)b^*$	c, d, ac, ad, cb, db, cbb, dbb, acb, adb, aac, aad, ...

Examples

a^*	$\epsilon,\ a,\ aa,\ aaa,\ \ldots$
$a^* b^*$	$\epsilon,\ a,\ b,\ bb,\ ab,\ aa,\ aaa,\ aab,\ abb,\ bbb,\ \ldots$
$a^*(c+d)b^*$	$c,\ d,\ ac,\ ad,\ cb,\ db,\ cbb,\ dbb,\ acb,\ adb,\ aac,\ aad,\ \ldots$
$a^* + b^*$	

Examples

a^*	$\epsilon, a, aa, aaa, \ldots$
$a^* b^*$	$\epsilon, a, b, bb, ab, aa, aaa, aab, abb, bbb, \ldots$
$a^*(c+d)b^*$	$c, d, ac, ad, cb, db, cbb, dbb, acb, adb, aac, aad, \ldots$
$a^* + b^*$	Not possible! (or you need two threads)

Examples

a^*	$\epsilon, a, aa, aaa, \ldots$
$a^* b^*$	$\epsilon, a, b, bb, ab, aa, aaa, aab, abb, bbb, \ldots$
$a^*(c+d)b^*$	$c, d, ac, ad, cb, db, cbb, dbb, acb, adb, aac, aad, \ldots$
$a^* + b^*$	Not possible! (or you need two threads)
$(a+b)^*$	

Examples

a^*	ϵ, a, aa, aaa, ...
a^*b^*	ϵ, a, b, bb, ab, aa, aaa, aab, abb, bbb, ...
$a^*(c+d)b^*$	c, d, ac, ad, cb, db, cbb, dbb, acb, adb, aac, aad, ...
$a^* + b^*$	Not possible! (or you need two threads)
$(a+b)^*$	ϵ, a, b,

Examples

a^*	$\epsilon, a, aa, aaa, \ldots$
$a^* b^*$	$\epsilon, a, b, bb, ab, aa, aaa, aab, abb, bbb, \ldots$
$a^*(c+d)b^*$	$c, d, ac, ad, cb, db, cbb, dbb, acb, adb, aac, aad, \ldots$
$a^* + b^*$	Not possible! (or you need two threads)
$(a+b)^*$	$\epsilon, a, b, ab, aa, ba, bb,$

Examples

a^*	$\epsilon, a, aa, aaa, \ldots$
$a^* b^*$	$\epsilon, a, b, bb, ab, aa, aaa, aab, abb, bbb, \ldots$
$a^*(c+d)b^*$	$c, d, ac, ad, cb, db, cbb, dbb, acb, adb, aac, aad, \ldots$
$a^* + b^*$	Not possible! (or you need two threads)
$(a+b)^*$	$\epsilon, a, b, ab, aa, ba, bb, abb, aba, aaa, aab, bab, baa, bba, bbb, \ldots$ (Gray code)

Examples

a^*	ϵ, a, aa, aaa, ...
$a^* b^*$	ϵ, a, b, bb, ab, aa, aaa, aab, abb, bbb, ...
$a^*(c+d)b^*$	c, d, ac, ad, cb, db, cbb, dbb, acb, adb, aac, aad, ...
$a^* + b^*$	Not possible! (or you need two threads)
$(a+b)^*$	ϵ, a, b, ab, aa, ba, bb, abb, aba, aaa, aab, bab, baa, bba, bbb, ... (Gray code)

Can you characterize the orderable languages?

Results

Theorem

Given a DFA A, we can determine in PTIME whether its language $L(A)$ is orderable

Results

Theorem

Given a DFA A, we can determine in PTIME whether its language $L(A)$ is orderable

- *If yes, it suffices to use push-pop edit operations at the left and right endpoints*

Results

Theorem

Given a DFA A, we can determine in PTIME whether its language $L(A)$ is orderable

- *If yes, it suffices to use push-pop edit operations at the left and right endpoints*
- *Further, we can enumerate the infinite sequence of edit scripts in bounded delay (i.e., depending on A, not on word length)*

Results

Theorem

Given a DFA A, we can determine in PTIME whether its language $L(A)$ is orderable

- *If yes, it suffices to use push-pop edit operations at the left and right endpoints*
- *Further, we can enumerate the infinite sequence of edit scripts in bounded delay (i.e., depending on A, not on word length)*
- *If not, we can decompose $L(A) = L(A_1) \sqcup \cdots \sqcup L(A_k)$ in PTIME where each $L(A_i)$ is orderable and k is minimal (and finite)*

Results

Theorem

Given a DFA A, we can determine in PTIME whether its language $L(A)$ is orderable

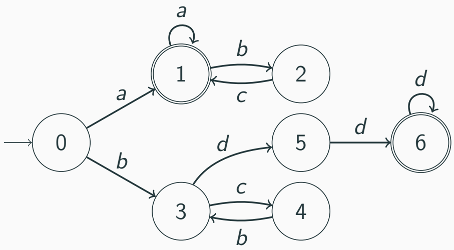
- *If yes, it suffices to use push-pop edit operations at the left and right endpoints*
- *Further, we can enumerate the infinite sequence of edit scripts in bounded delay (i.e., depending on A, not on word length)*
- *If not, we can decompose $L(A) = L(A_1) \sqcup \cdots \sqcup L(A_k)$ in PTIME where each $L(A_i)$ is orderable and k is minimal (and finite)*

Also:

- Characterization if we only allow edits at the right endpoint (= stack, not deque)
- Finding the minimal distance bound is NP-hard

Proof techniques

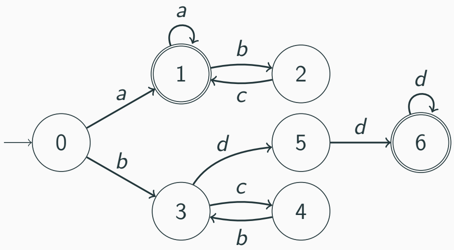
Pleasant (and elementary): orderability



- Equivalence relation on loopable states

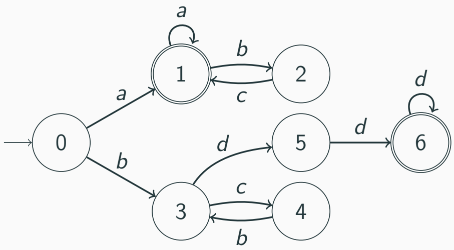
Proof techniques

Pleasant (and elementary): orderability



- Equivalence relation on loopable states
- Two loopable states are equivalent if they co-occur in a run

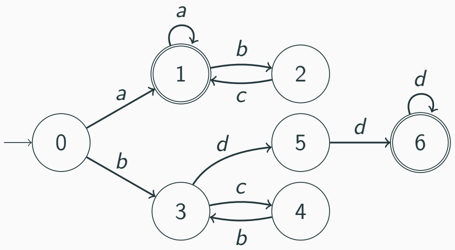
Pleasant (and elementary): orderability



- Equivalence relation on loopable states
- Two loopable states are equivalent if they co-occur in a run
- Two loopable states are equivalent if some word can loop on both of them

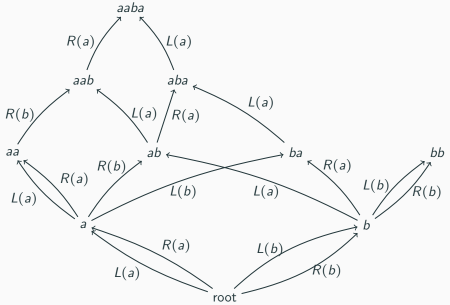
Proof techniques

Pleasant (and elementary): orderability



- Equivalence relation on loopable states
- Two loopable states are equivalent if they co-occur in a run
- Two loopable states are equivalent if some word can loop on both of them

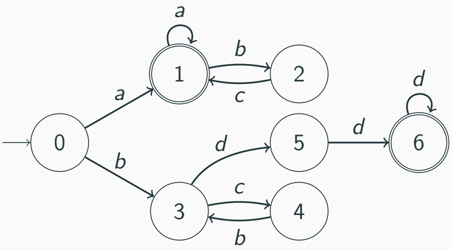
Unpleasant (and exponential): enumeration



- Pointer machine model because memory usage goes to infinity

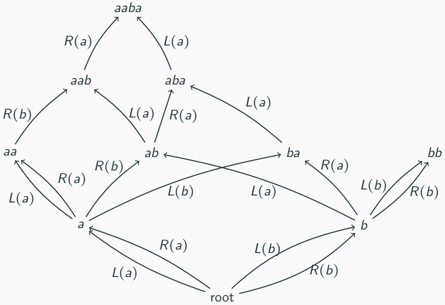
Proof techniques

Pleasant (and elementary): orderability



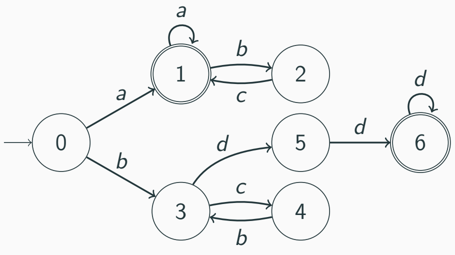
- Equivalence relation on loopable states
- Two loopable states are equivalent if they co-occur in a run
- Two loopable states are equivalent if some word can loop on both of them

Unpleasant (and exponential): enumeration



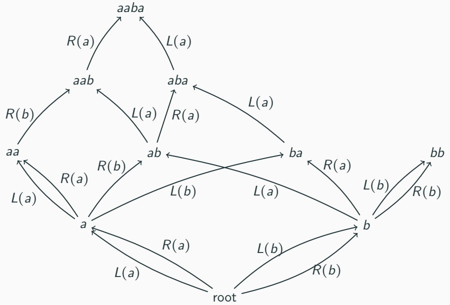
- Pointer machine model because memory usage goes to infinity
- Everything is exponential in the DFA

Pleasant (and elementary): orderability



- Equivalence relation on loopable states
- Two loopable states are equivalent if they co-occur in a run
- Two loopable states are equivalent if some word can loop on both of them

Unpleasant (and exponential): enumeration



- Pointer machine model because memory usage goes to infinity
- Everything is exponential in the DFA
- Probably simplifiable...

Future work

Open questions and future work:

- Make the delay polynomial in $|A|$? (currently it is exponential)
- What about the push-left pop-right distance? the padded Hamming distance?
- What about enumeration in radix order?
- What about regular tree languages?
- Can we go beyond regular languages?
- Other uses of the enumeration model?

Future work

Open questions and future work:

- Make the delay polynomial in $|A|$? (currently it is exponential)
- What about the push-left pop-right distance? the padded Hamming distance?
- What about enumeration in radix order?
- What about regular tree languages?
- Can we go beyond regular languages?
- Other uses of the enumeration model?

Thanks for your attention!