On the Membership (Word) Problem of Rational Trace Languages

Case of Local Well-Structured Loop-Only Automata

–

Luca Breveglieri[1], Stefano Crespi Reghizzi[1],
Massimiliano Goldwurm[2], Alessandra Savelli[1]

–

[1]Politecnico di Milano
[2]Universita' degli Studi di Milano

–

October 2008 - Cremona - Automatha - Workshop on Traces

Introduction

Objective

Objective

- improve the (worst case) time complexity of the membership problem of rational trace languages

Objective

- improve the (worst case) time complexity of the membership problem of rational trace languages
  - time complexity known to be polynomial

    Bertoni, Mauri and Sabadini - general problem

    Avellone and Goldwurm - problem in the average case

    Rytter - problem for context-free trace languages

Objective

- improve the (worst case) time complexity of the membership problem of rational trace languages
  - time complexity known to be polynomial

    Bertoni, Mauri and Sabadini - general problem

    Avellone and Goldwurm - problem in the average case

    Rytter - problem for context-free trace languages
  - but with a high exponent (dependences)

Objective

- improve the (worst case) time complexity of the membership problem of rational trace languages
  - time complexity known to be polynomial

    Bertoni, Mauri and Sabadini - general problem

    Avellone and Goldwurm - problem in the average case

    Rytter - problem for context-free trace languages
  - but with a high exponent (dependences)
- obtain a lower polynomial time complexity

Objective

- improve the (worst case) time complexity of the membership problem of rational trace languages
  - time complexity known to be polynomial

    Bertoni, Mauri and Sabadini - general problem

    Avellone and Goldwurm - problem in the average case

    Rytter - problem for context-free trace languages
  - but with a high exponent (dependences)
- obtain a lower polynomial time complexity
- possibly not a function of the dependences

Objective

- improve the (worst case) time complexity of the membership problem of rational trace languages
  - time complexity known to be polynomial

    Bertoni, Mauri and Sabadini - general problem

    Avellone and Goldwurm - problem in the average case

    Rytter - problem for context-free trace languages
  - but with a high exponent (dependences)
- obtain a lower polynomial time complexity
- possibly not a function of the dependences
- at least for some rational trace languages of applicative interest (e.g. compilation)

Approach

Approach

- restrict to a subfamily of rational trace languages

Approach

- restrict to a subfamily of rational trace languages
- generated by local well-structured automata consisting only of loops (called loop-only)

Approach

- restrict to a subfamily of rational trace languages
- generated by local well-structured automata consisting only of loops (called loop-only)
  - local: letter pairs and initial-final letters

Approach

- restrict to a subfamily of rational trace languages
- generated by local well-structured automata consisting only of loops (called loop-only)
  - local: letter pairs and initial-final letters
  - well-structured: modular hierarchical structure

Approach

- restrict to a subfamily of rational trace languages
- generated by local well-structured automata consisting only of loops (called loop-only)
  - local: letter pairs and initial-final letters
  - well-structured: modular hierarchical structure
  - loop-only: modules are loops (repeat-until style)

Approach

- restrict to a subfamily of rational trace languages
- generated by local well-structured automata consisting only of loops (called loop-only)
  - local: letter pairs and initial-final letters
  - well-structured: modular hierarchical structure
  - loop-only: modules are loops (repeat-until style)
- such restriction models the trace of a program execution (well-structured machine code generated by a compiler)

Approach

- restrict to a subfamily of rational trace languages
- generated by local well-structured automata consisting only of loops (called loop-only)
  - local: letter pairs and initial-final letters
  - well-structured: modular hierarchical structure
  - loop-only: modules are loops (repeat-until style)
- such restriction models the trace of a program execution (well-structured machine code generated by a compiler)

- this is a work in progress ...

Loop-Only Automaton

Automaton Model

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)
  - every loop body is run at least once

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)
  - every loop body is run at least once
  - does not have any fork-join branches

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)
  - every loop body is run at least once
  - does not have any fork-join branches
- automaton states and alphabet letters are in one-to-one correspondence (states are identified by letters)

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)
  - every loop body is run at least once
  - does not have any fork-join branches
- automaton states and alphabet letters are in one-to-one correspondence (states are identified by letters)
- such a model corresponds to a linear regexp with:

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)
  - every loop body is run at least once
  - does not have any fork-join branches
- automaton states and alphabet letters are in one-to-one correspondence (states are identified by letters)
- such a model corresponds to a linear regexp with:
  - concatenation

Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)
  - every loop body is run at least once
  - does not have any fork-join branches
- automaton states and alphabet letters are in one-to-one correspondence (states are identified by letters)
- such a model corresponds to a linear regexp with:
  - concatenation
  - Kleene cross

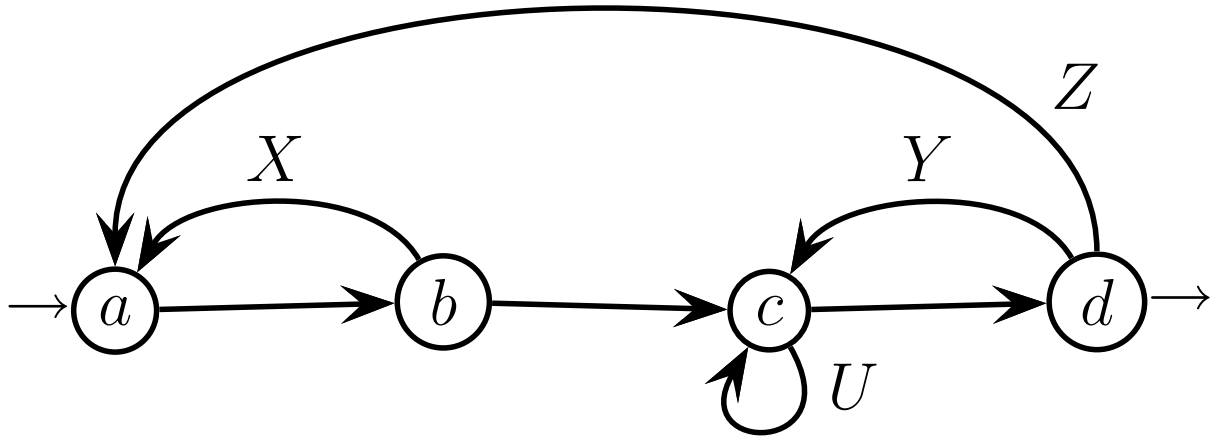
Automaton Model

- a local well-structured loop-only automaton (briefly a loop-only automaton) is characterised as follows:
  - recognises a local regular language
  - has unique initial and final states
  - loops are well-nested (a hierarchy)
  - every loop body is run at least once
  - does not have any fork-join branches
- automaton states and alphabet letters are in one-to-one correspondence (states are identified by letters)
- such a model corresponds to a linear regexp with:
  - concatenation
  - Kleene cross

  union and Kleene star are forbidden

Example

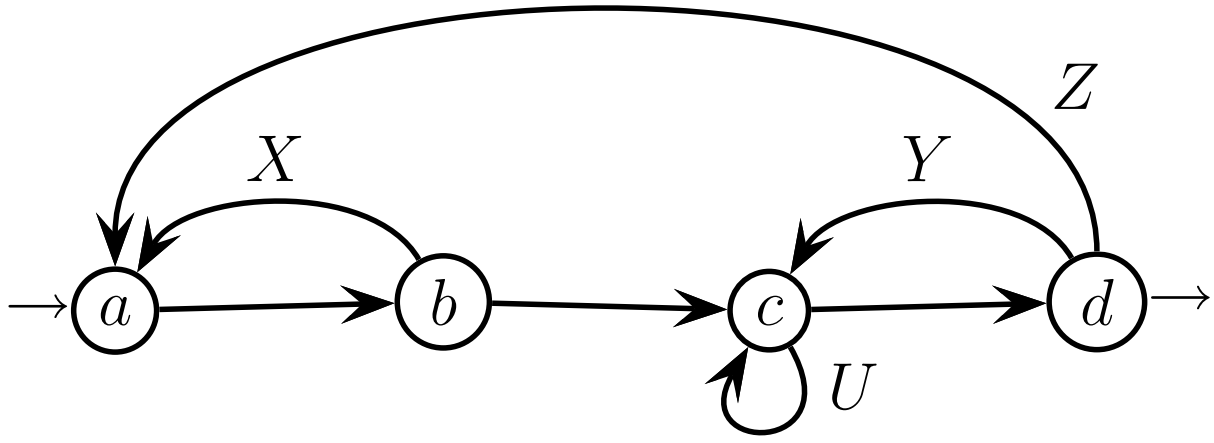
Example



loop-only automaton over alphabet $\{a, b, c, d\}$

(by convention loop names X, Y, Z and U label the backward arcs)

Example



loop-only automaton over alphabet $\{a, b, c, d\}$

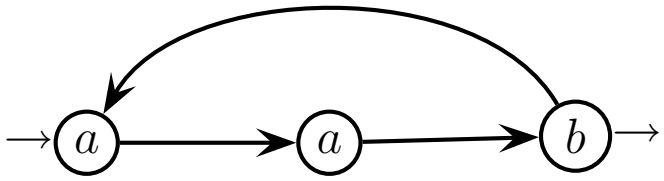
(by convention loop names X, Y, Z and U label the backward arcs)

$$\Big((\, a\, b\,)^{+}\ (\, c^{+}\, d\,)^{+} \Big)^{+}$$

linear regexp with concatenation and Kleene cross

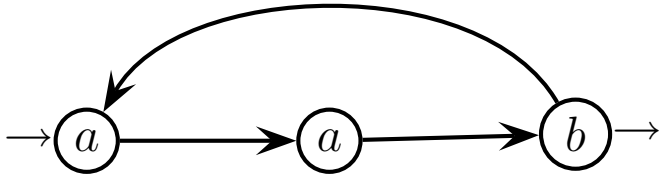
Counterexamples

Counterexamples

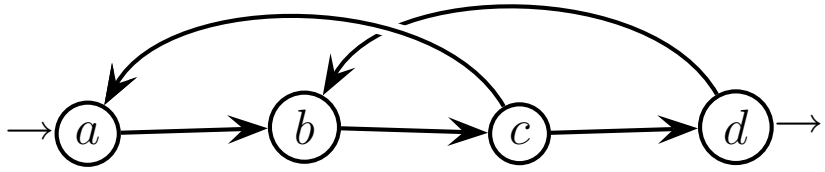


not local (counting)

Counterexamples

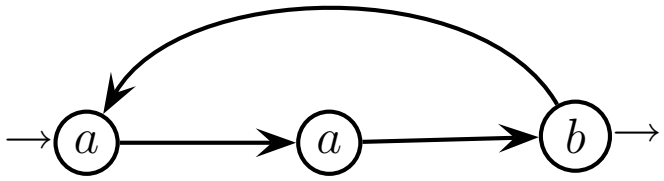


not local (counting)

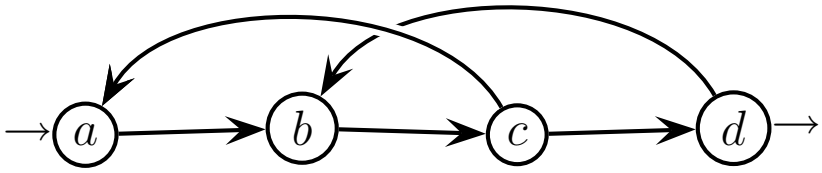


not well-structured

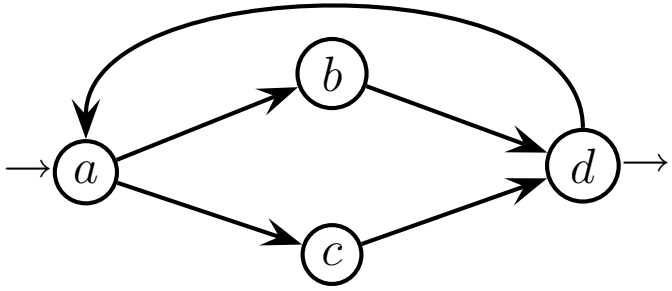
Counterexamples



not local (counting)



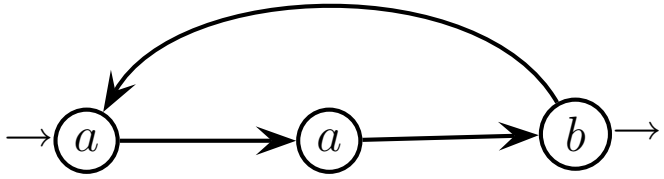
not well-structured



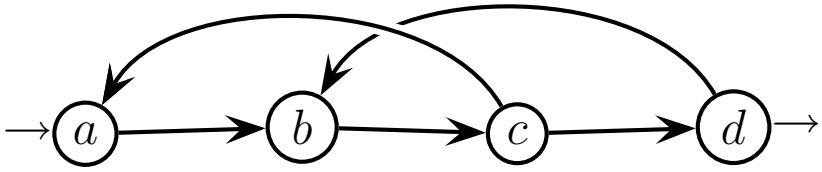
fork-join (union)

$$(\; a \; (\; b \; | \; c \;) \; d \;)^{+}$$

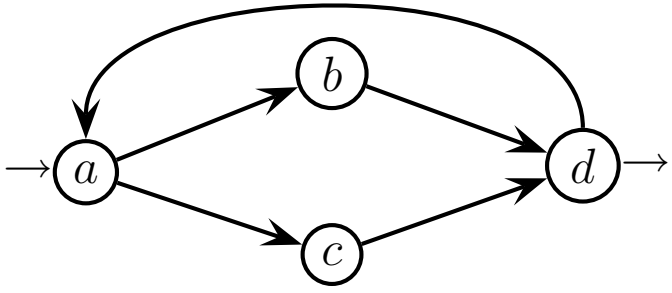
Counterexamples



not local (counting)

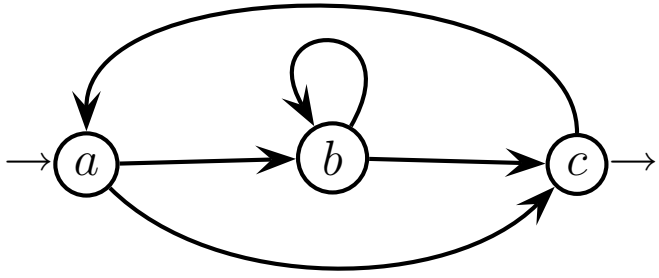


not well-structured



fork-join (union)

$$(\; a \; (\; b \; \mid \; c \;) \; d \;)^{+}$$

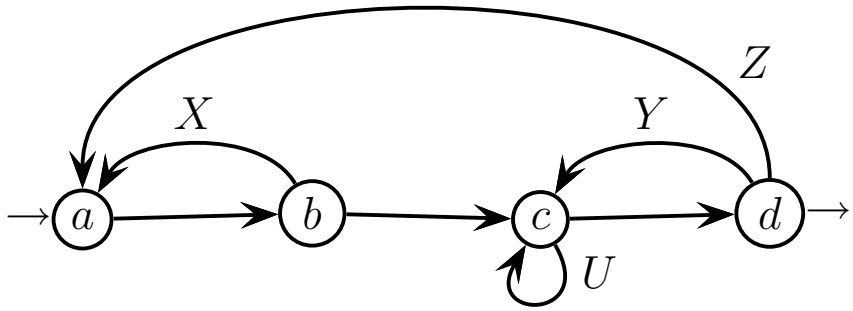


Kleene star

$$(\; a \; b^{*} \; c \;)^{+}$$

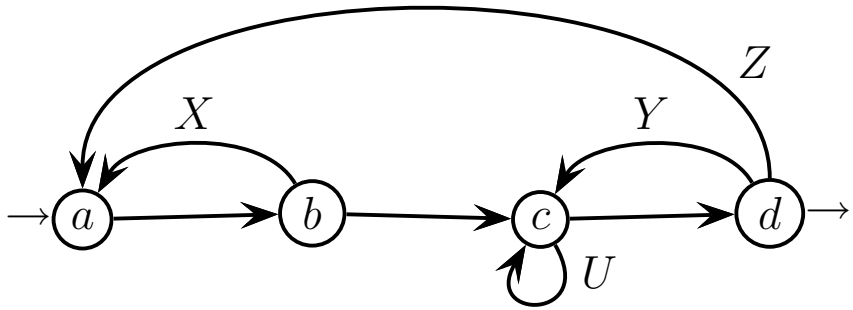
Hierarchy & Loop Identification

Hierarchy & Loop Identification

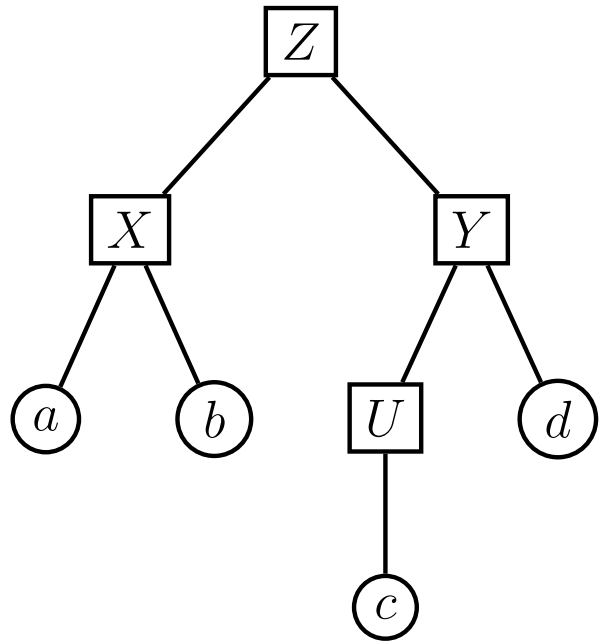


automaton

Hierarchy & Loop Identification



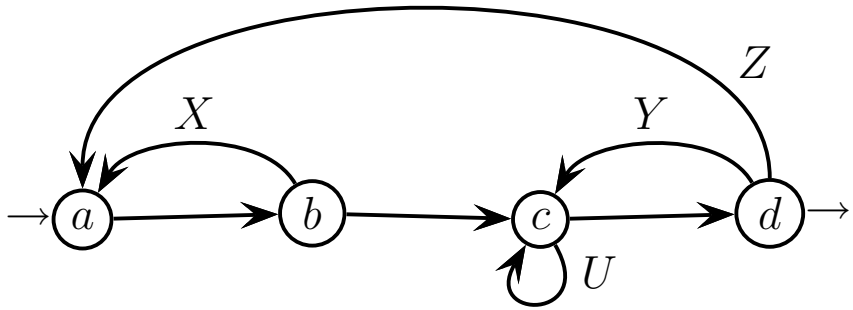
automaton



$U \lhd Y \lhd Z$ and $X \lhd Z$

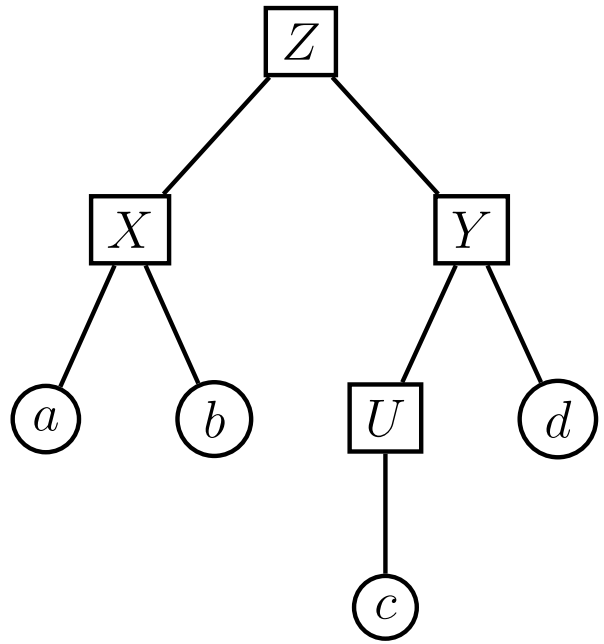
hierarchy tree

Hierarchy & Loop Identification



automaton

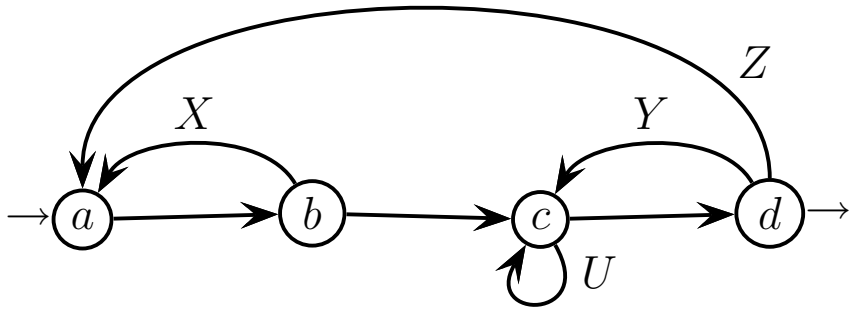
loop	identification			
Z	$a\,c$	$a\,d$	$b\,c$	$b\,d$
X	a	b	$a\,b$	
Y	d	$c\,d$		
U	c			



$U \lhd Y \lhd Z$ and $X \lhd Z$

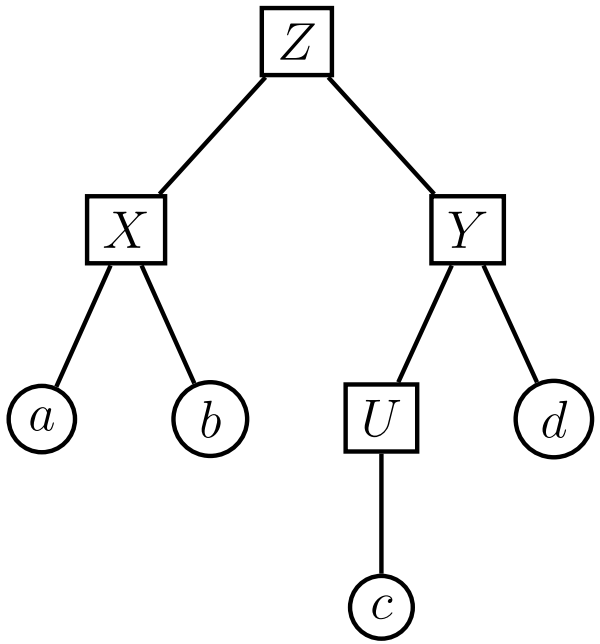
hierarchy tree

Hierarchy & Loop Identification



automaton

loop	identification			
Z	$a\,c$	$a\,d$	$b\,c$	$b\,d$
X	a	b	$a\,b$	
Y	d	$c\,d$		
U	c			



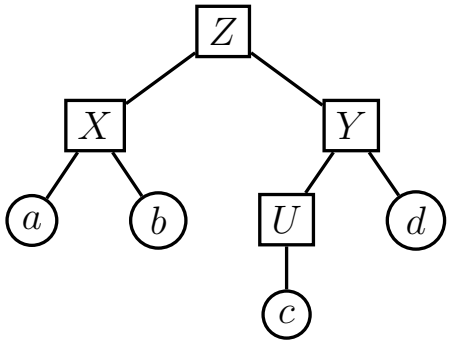
$U \lhd Y \lhd Z$ and $X \lhd Z$

hierarchy tree

every loop is identified (in general not uniquely)
by a single letter or at most by a letter pair

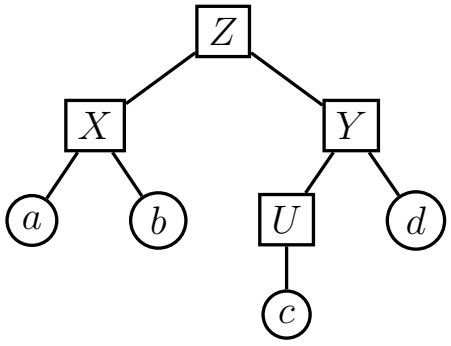
Syntax Tree

Syntax Tree



hierarchy tree

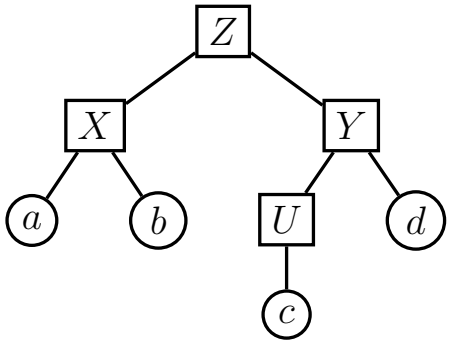
Syntax Tree



hierarchy tree

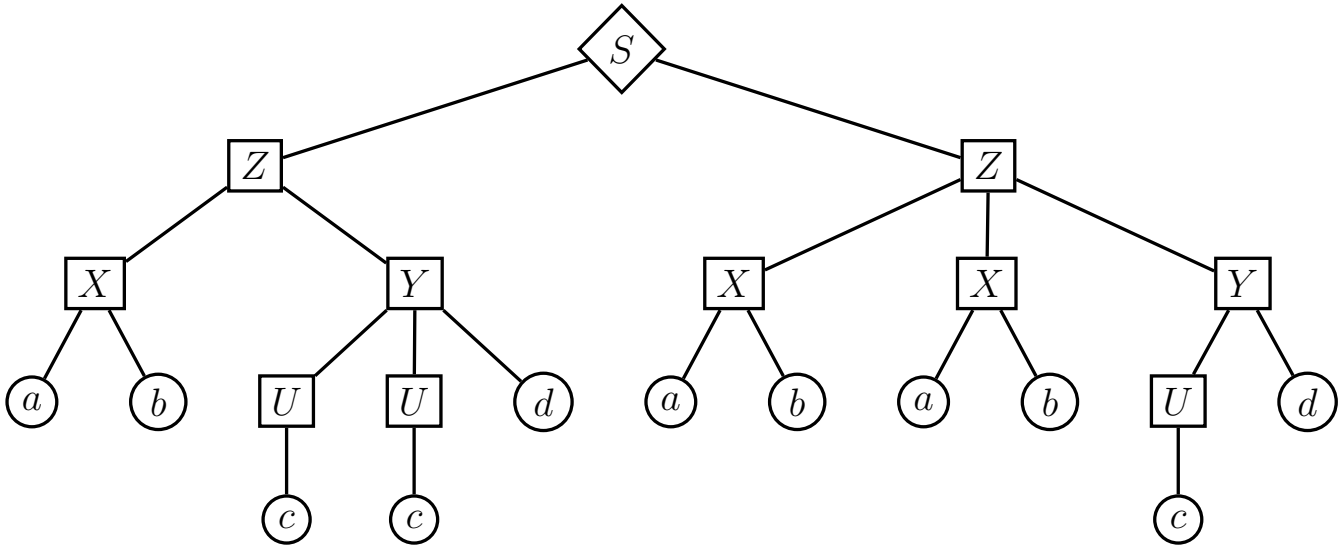
$$x = a\,b\,c\,c\,d\,a\,b\,a\,b\,c\,d$$

Syntax Tree



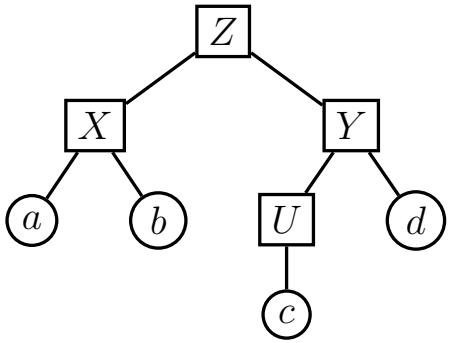
hierarchy tree

$$x = a\,b\,c\,c\,d\,a\,b\,a\,b\,c\,d$$



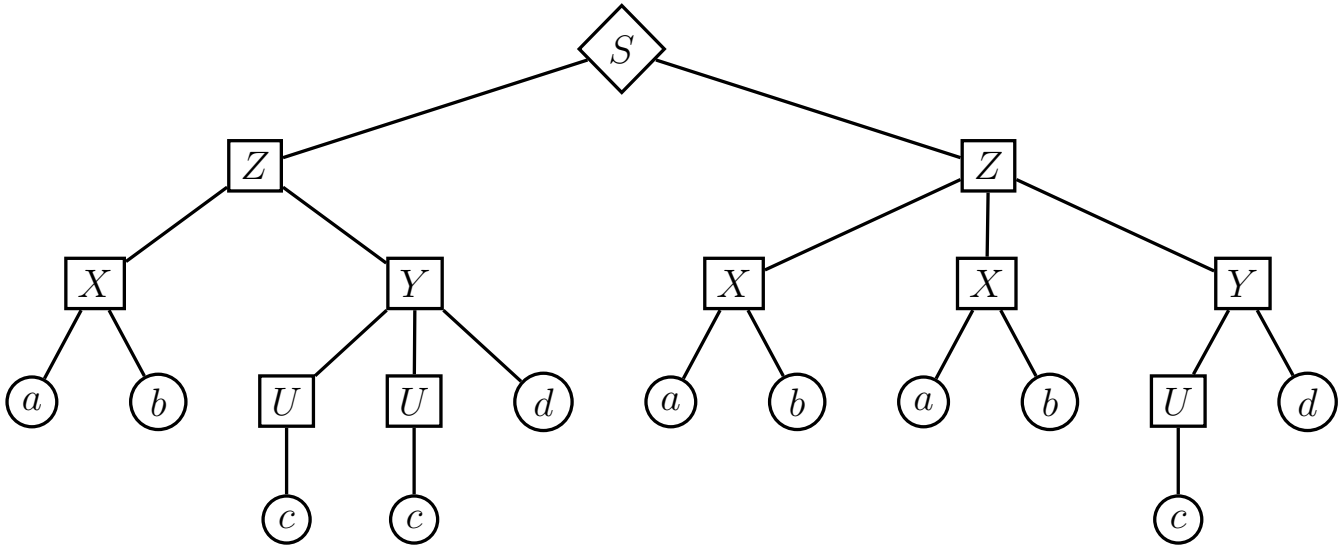
syntax tree of string x (S is a formal root)

Syntax Tree



hierarchy tree

$$x = a\,b\,c\,c\,d\,a\,b\,a\,b\,c\,d$$



syntax tree of string x (S is a formal root)

string x has a syntax tree iff it is recognised by the automaton

Membership Problem

Definition

Definition

- suppose there are:

Definition

- suppose there are:
  - loop-only automaton A

Definition

- suppose there are:
  - loop-only automaton A
  - dependence relation D

Definition

- suppose there are:
  - loop-only automaton A
  - dependence relation D
  - trace language L generated by $L(A)$ and D

$$L = \{\, x \mid \quad \exists\, y \in L(A) \quad x \cong_D y \,\}$$

Definition

- suppose there are:
  - loop-only automaton A
  - dependence relation D
  - trace language L generated by $L(A)$ and D

$$L = \{\, x \mid \quad \exists\, y \in L(A) \quad x \cong_D y \,\}$$

  - string x over the alphabet Σ

Definition

- suppose there are:
  - loop-only automaton A
  - dependence relation D
  - trace language L generated by $L(A)$ and D

$$L = \{\, x \mid \quad \exists\, y \in L(A) \quad x \cong_D y \,\}$$

  - string x over the alphabet Σ
- membership problem: decide if $x \in L$

Definition

- suppose there are:
  - loop-only automaton A
  - dependence relation D
  - trace language L generated by $L(A)$ and D

$$L = \{\, x \mid \quad \exists\, y \in L(A) \quad x \cong_D y \,\}$$

  - string x over the alphabet Σ
- membership problem: decide if $x \in L$
- that is check if $\exists\, y \quad y \in L(A) \ \wedge\ y \cong_D x$

Definition

- suppose there are:
  - loop-only automaton A
  - dependence relation D
  - trace language L generated by $L(A)$ and D

$$L = \{\, x \mid \quad \exists\, y \in L(A) \quad x \cong_D y \,\}$$

  - string x over the alphabet Σ
- membership problem: decide if $x \in L$
- that is check if $\exists\, y \quad y \in L(A) \ \wedge\ y \cong_D x$
- try to build the syntax tree of string y

Some Known Facts

Some Known Facts

- membership problem afforded by a $ad\ hoc$ procedure
  (Savelli et alii - Ph.D. thesis - 2006 - WORDS - Monreal - 2005)

Some Known Facts

- membership problem afforded by a $ad\ hoc$ procedure
  (Savelli et alii - Ph.D. thesis - 2006 - WORDS - Monreal - 2005)
- procedure does not use trace prefixes

Some Known Facts

- membership problem afforded by a $ad\ hoc$ procedure
  (Savelli et alii - Ph.D. thesis - 2006 - WORDS - Monreal - 2005)
- procedure does not use trace prefixes
- analyses in parallel input string x

Some Known Facts

- membership problem afforded by a $ad\ hoc$ procedure
  (Savelli et alii - Ph.D. thesis - 2006 - WORDS - Monreal - 2005)
- procedure does not use trace prefixes
- analyses in parallel input string x
- builds the syntax tree of string $y \cong_D x$

Some Known Facts

- membership problem afforded by a $ad\ hoc$ procedure
  (Savelli et alii - Ph.D. thesis - 2006 - WORDS - Monreal - 2005)
- procedure does not use trace prefixes
- analyses in parallel input string x
- builds the syntax tree of string $y \cong_D x$
- demonstrated to work on several examples

Some Known Facts

- membership problem afforded by a $ad\ hoc$ procedure
  (Savelli et alii - Ph.D. thesis - 2006 - WORDS - Monreal - 2005)
- procedure does not use trace prefixes
- analyses in parallel input string x
- builds the syntax tree of string $y \cong_D x$
- demonstrated to work on several examples
- but misses a fully rigorous formal setting

New Expected Results

New Expected Results

- new formulation of the $ad\ hoc$ procedure:

New Expected Results

- new formulation of the $ad\ hoc$ procedure:
  - with a more rigorous formal setting

New Expected Results

- new formulation of the $ad\ hoc$ procedure:
  - with a more rigorous formal setting
  - reduced to a numerical question related to the concept of integer composition

New Expected Results

- new formulation of the $ad\ hoc$ procedure:
  - with a more rigorous formal setting
  - reduced to a numerical question related to the concept of integer composition
  - with a more rigorous complexity analysis

New Expected Results

- new formulation of the $ad\ hoc$ procedure:
  - with a more rigorous formal setting
  - reduced to a numerical question related to the concept of integer composition
  - with a more rigorous complexity analysis
- potentiality for generalisation (union and star)

Simple Case

Dependent Successors

Dependent Successors

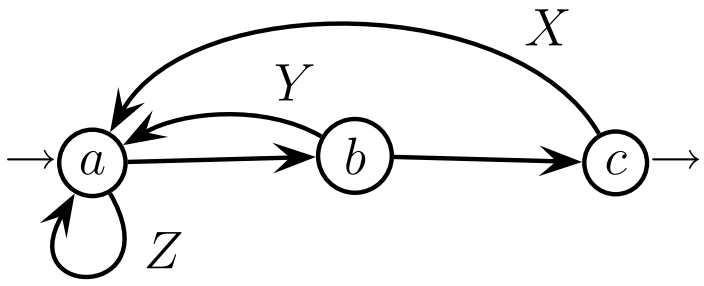
- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time

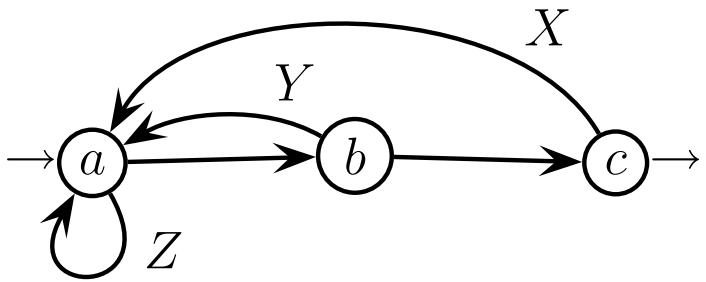


$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

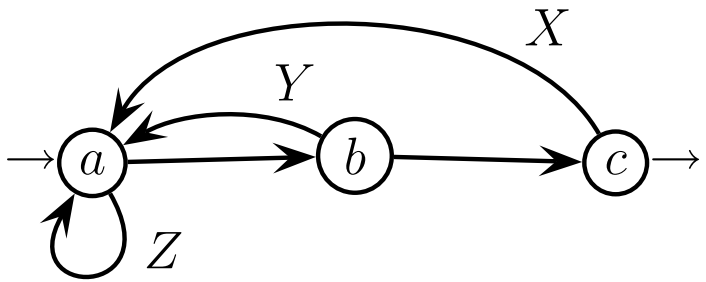
$$\pi_{a\,b}(x) = a_1 \quad b_1 \quad a_2 \quad a_3 \quad b_2 \quad a_4 \quad a_5 \quad b_3$$

$$\pi_{a\,c}(x) = a_1 \quad a_2 \quad a_3 \quad c_1 \quad a_4 \quad a_5 \quad c_2$$

$$y =$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



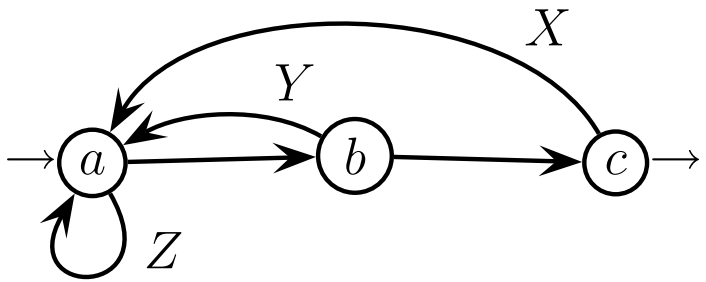
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & & & & & & & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & & & & & & & \\
y & = & a_1 & & & & & & &
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



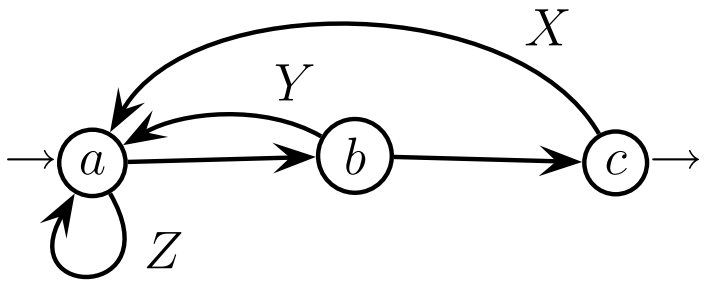
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & & & & & & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & & & & & & & \\
y & = & a_1 & b_1 & & & & & &
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



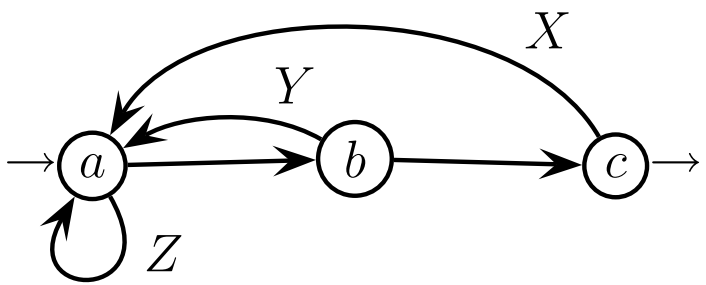
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & - & & & & & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & - & & & & & & \\
y & = & a_1 & b_1 & a_2 & & & & &
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



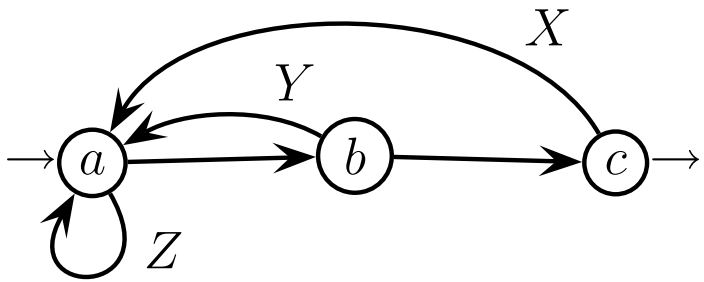
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & - & - & & & & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & - & - & & & & & \\
y & = & a_1 & b_1 & a_2 & a_3 & & & &
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



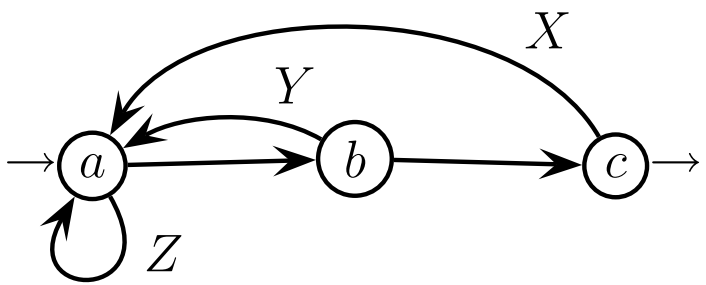
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & - & - & - & & & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & - & - & & & & & \\
y & = & a_1 & b_1 & a_2 & a_3 & b_2 & & &
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



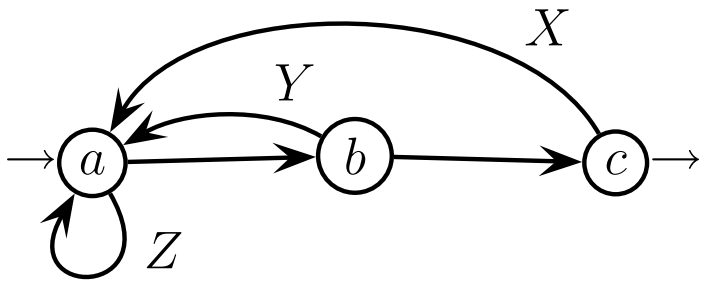
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & - & - & - & & & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & - & - & - & & & & \\
y & = & a_1 & b_1 & a_2 & a_3 & b_2 & c_1 & &
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



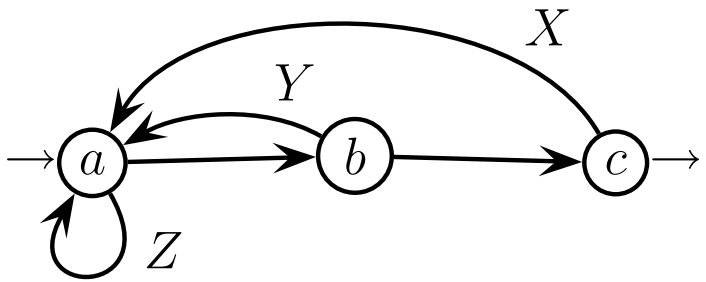
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & - & - & - & - & & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & - & - & - & - & & & \\
y & = & a_1 & b_1 & a_2 & a_3 & b_2 & c_1 & a_4 &
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



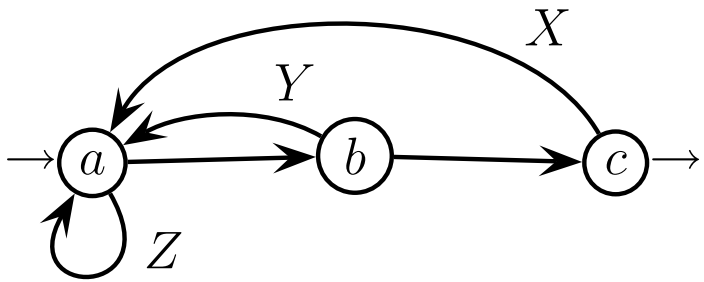
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & - & - & - & - & - & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & - & - & - & - & - & & \\
y & = & a_1 & b_1 & a_2 & a_3 & b_2 & c_1 & a_4 & a_5
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



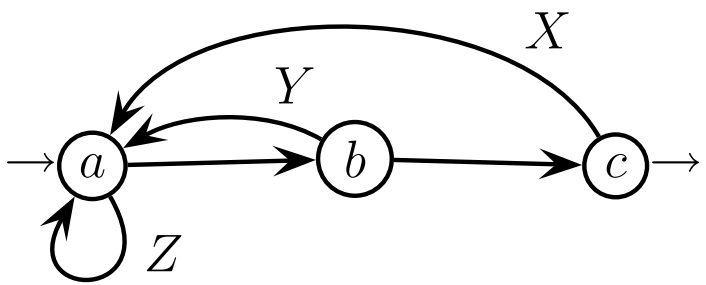
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 \\
 & & - & - & - & - & - & - & - & - \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & \\
 & & - & - & - & - & - & - & & \\
y & = & a_1 & b_1 & a_2 & a_3 & b_2 & c_1 & a_4 & a_5 & b_3
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



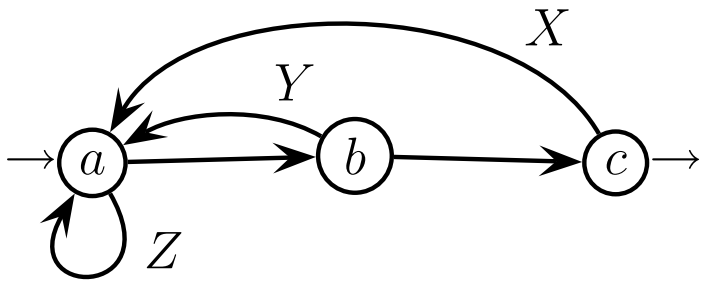
$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

$$
\begin{array}{rcccccccccc}
\pi_{a\,b}(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 & \\
 & & - & - & - & - & - & - & - & - & \\
\pi_{a\,c}(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & & \\
 & & - & - & - & - & - & - & - & & \\
y & = & a_1 & b_1 & a_2 & a_3 & b_2 & c_1 & a_4 & a_5 & b_3 & c_2
\end{array}
$$

Dependent Successors

- suppose for each state s of the automaton, the letters of the successor states of s are dependent of one another (i.e. are a clique of the dependence relation D)
- then membership problem can be solved in linear time



$x = a\,b\,a\,a\,c\,b\,a\,a\,c\,b$

dependent letters: $a\,b$ and $a\,c$

string $y \cong_D x$ with $y \in L\,(A)$ exists

string x is accepted

$$
\begin{array}{rcccccccccc}
\pi_{a\,b}\,(x) & = & a_1 & b_1 & a_2 & a_3 & b_2 & a_4 & a_5 & b_3 & \\
 & & - & - & - & - & - & - & - & - & \\
\pi_{a\,c}\,(x) & = & a_1 & a_2 & a_3 & c_1 & a_4 & a_5 & c_2 & & \\
 & & - & - & - & - & - & - & - & & \\
y & = & a_1 & b_1 & a_2 & a_3 & b_2 & c_1 & a_4 & a_5 & b_3 & c_2
\end{array}
$$

Considerations

Considerations

- algorithm works for any automaton type

Considerations

- algorithm works for any automaton type
- is very simple for loop-only automata
  - only loop final states have two or more successors
  - string projections drive directly loop scheduling

Considerations

- algorithm works for any automaton type
- is very simple for loop-only automata
  - only loop final states have two or more successors
  - string projections drive directly loop scheduling
- is based on parallel scan of projections

Considerations

- algorithm works for any automaton type
- is very simple for loop-only automata
  - only loop final states have two or more successors
  - string projections drive directly loop scheduling
- is based on parallel scan of projections
- similar idea is used in the general case
  - examine in parallel projections
  - build bottom-up the syntax tree

  parallelism is exploited more extensively

Composition and Syntax Tree

Integer Composition

Integer Composition

- give automaton A and hierarchy tree H

Integer Composition

- give automaton A and hierarchy tree H
- relation $Y \lhd X$ means that in the tree H node Y is descendant (possibly child) of node X

Integer Composition

- give automaton A and hierarchy tree H
- relation $Y \lhd X$ means that in the tree H node Y is descendant (possibly child) of node X
- give the syntax tree T of a string in $L(A)$

Integer Composition

- give automaton A and hierarchy tree H
- relation $Y \lhd X$ means that in the tree H node Y is descendant (possibly child) of node X
- give the syntax tree T of a string in $L(A)$
- define integer composition χ_Y^X to be a non-empty list of positive non-null integers as follows:

$$\chi_Y^X = \left(y_1, \ldots, y_i, \ldots, y_{l_\chi}\right)$$

Integer Composition

- give automaton A and hierarchy tree H
- relation $Y \lhd X$ means that in the tree H node Y is descendant (possibly child) of node X
- give the syntax tree T of a string in $L(A)$
- define integer composition χ_Y^X to be a non-empty list of positive non-null integers as follows:

$$\chi_Y^X = \left(y_1, \ldots, y_i, \ldots, y_{l_\chi}\right)$$

where

  - $l_\chi \geq 1$ is the number of nodes X in the tree T
  - y_i $(1 \leq i \leq l_\chi)$ is the number of nodes Y descendant (possibly children) of the i-th node X in the tree T

Integer Composition

- give automaton A and hierarchy tree H
- relation $Y \lhd X$ means that in the tree H node Y is descendant (possibly child) of node X
- give the syntax tree T of a string in $L(A)$
- define integer composition χ_Y^X to be a non-empty list of positive non-null integers as follows:

$$\chi_Y^X = \left(y_1, \ldots, y_i, \ldots, y_{l_\chi}\right)$$

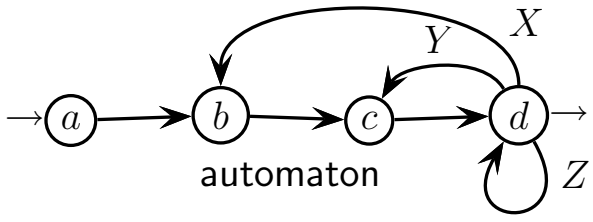
where

  - $l_\chi \geq 1$ is the number of nodes X in the tree T
  - y_i $(1 \leq i \leq l_\chi)$ is the number of nodes Y descendant (possibly children) of the i-th node X in the tree T

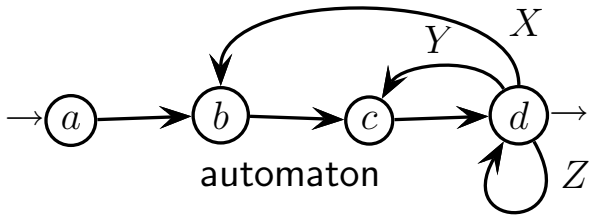
- χ_Y^X encodes the relationship between nodes X and Y

Example

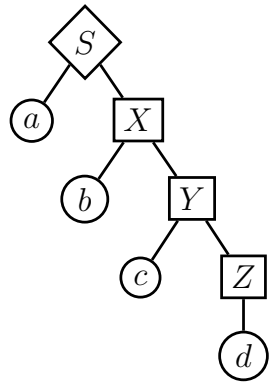
Example



Example

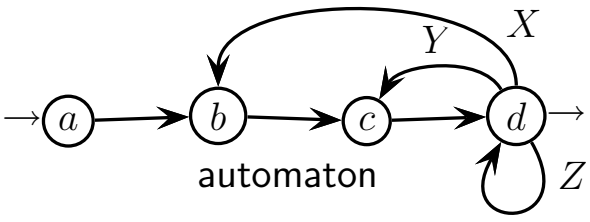


automaton

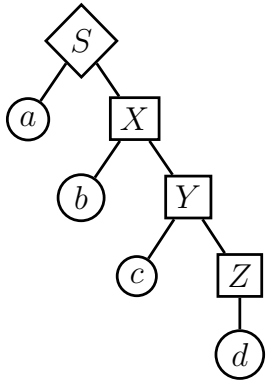


hierarchy tree

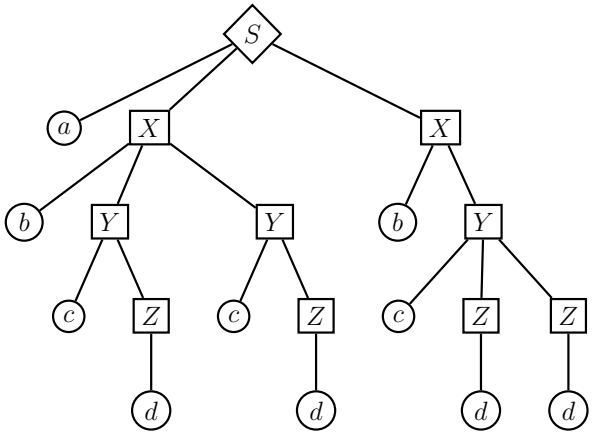
Example



automaton

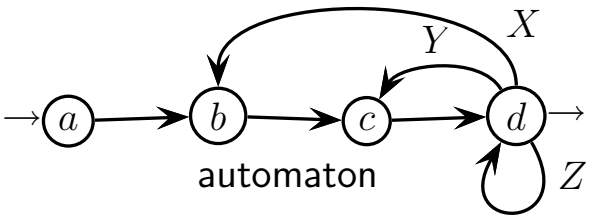


hierarchy tree

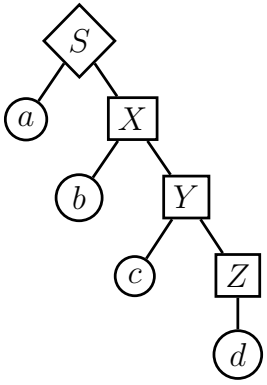


syntax tree of $a\,b\,c\,d\,c\,d\,b\,c\,d\,d$

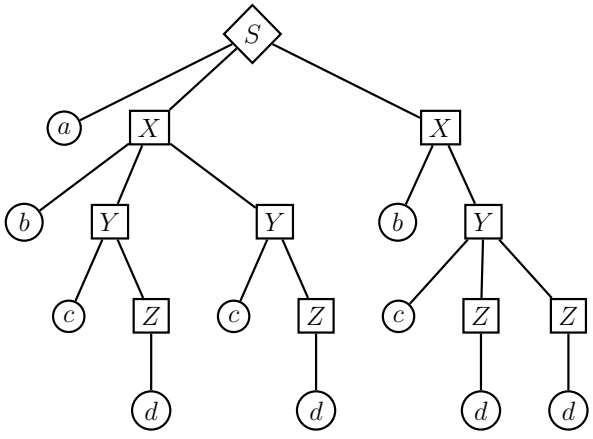
Example



automaton



hierarchy tree



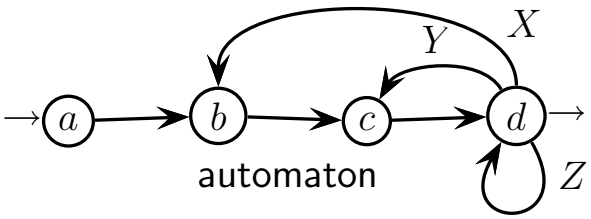
syntax tree of $a\,b\,c\,d\,c\,d\,b\,c\,d\,d$

$$\alpha_X^S = (2)$$
$$\beta_Z^Y = (1,1,2)$$

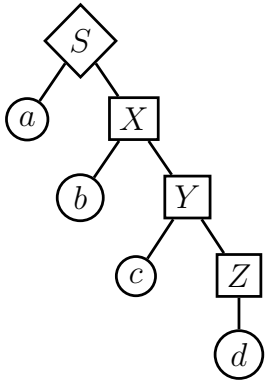
etc

integer compositions

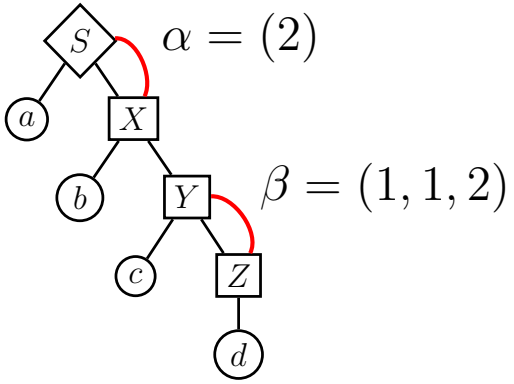
Example



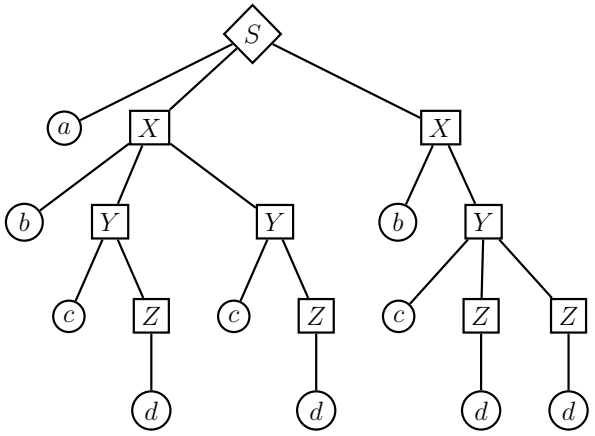
automaton



hierarchy tree



syntax tree (compact)



syntax tree of $a\,b\,c\,d\,c\,d\,b\,c\,d\,d$

$$\alpha_X^S = (2)$$
$$\beta_Z^Y = (1,1,2)$$

etc

integer compositions

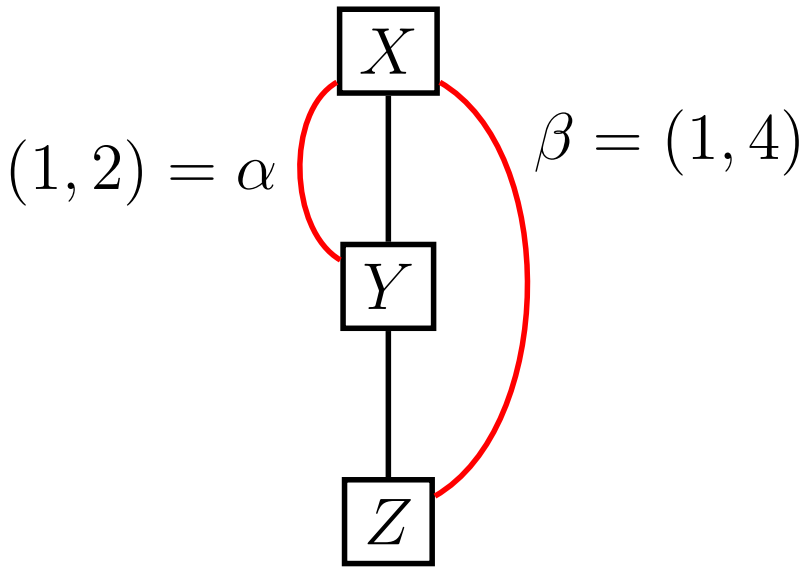
Orders on Compositions

Orders on Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree

Orders on Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- elements of α^X_Y are orderly smaller than those β^X_Z

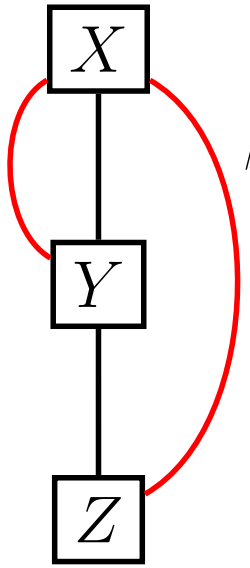


$$\alpha \leq \beta$$

Orders on Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- elements of α_Y^X are orderly smaller than those β_Z^X
- elements of α_Z^Y are splittings of those of β_Z^X

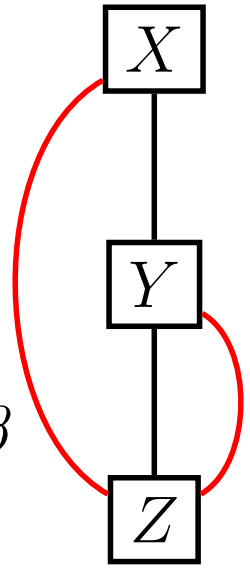
$(1,2) = \alpha$



$\beta = (1,4)$

$\alpha \leq \beta$

$(3,5) = \beta$

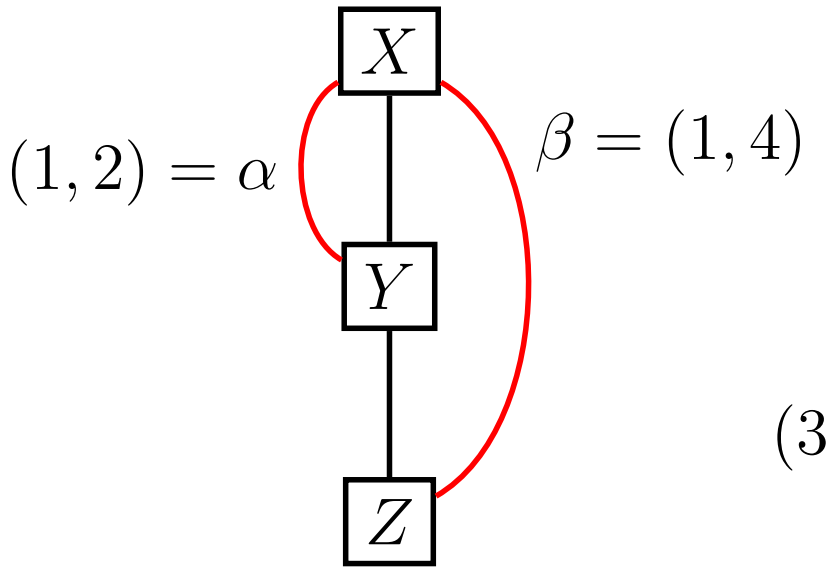


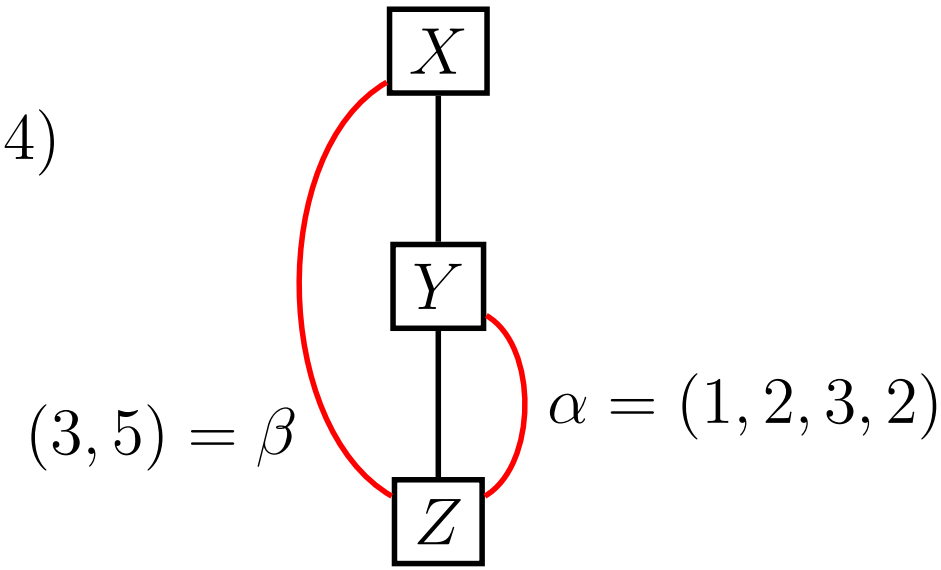
$\alpha = (1,2,3,2)$

$\alpha \preceq \beta$

Orders on Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- elements of α_Y^X are orderly smaller than those β_Z^X
- elements of α_Z^Y are splittings of those of β_Z^X





$\alpha \leq \beta$ $\qquad$ $\alpha \preceq \beta$

both relations can be verified in linear time

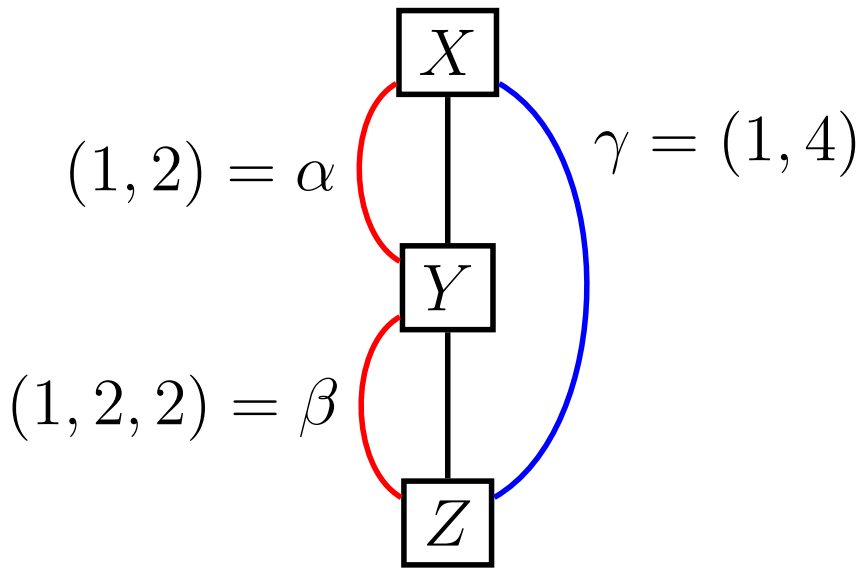
Operations on Compositions

Operations on Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree

Operations on Compositions

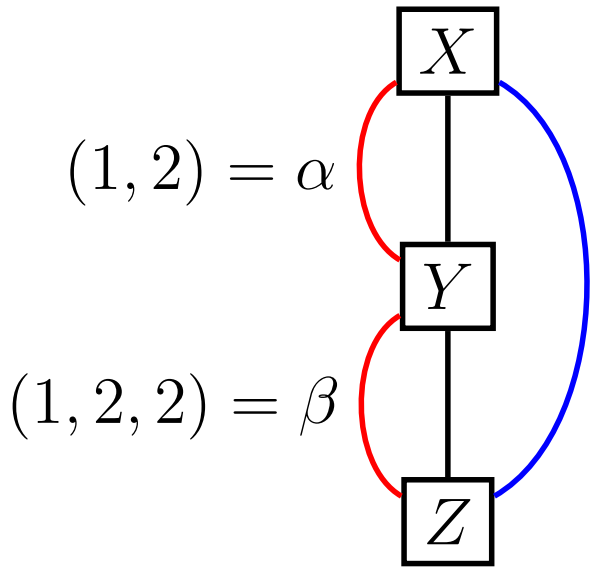
- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- product $\alpha_Y^X \cdot \beta_Z^Y = \gamma_Z^X$ - tree relation from X downto Z



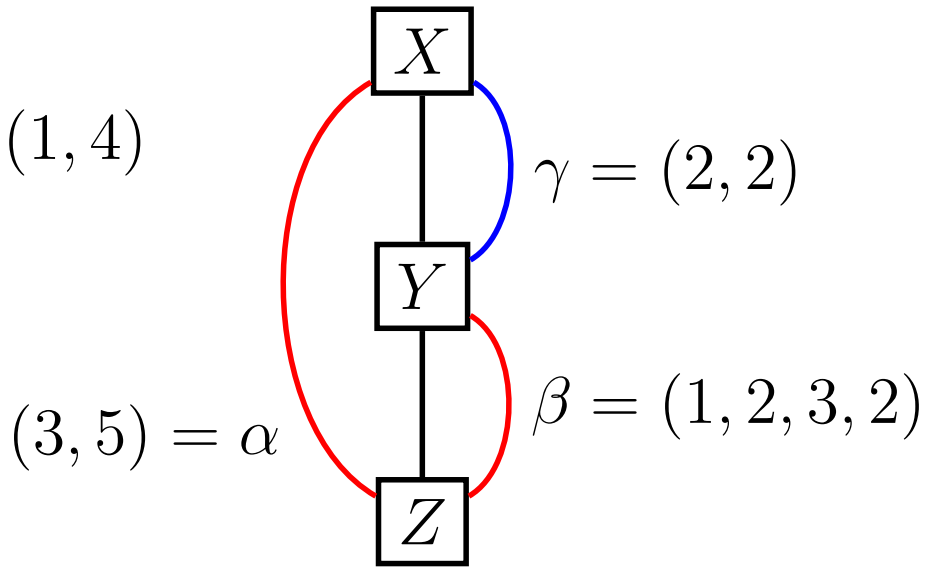
product

Operations on Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- product $\alpha_Y^X \cdot \beta_Z^Y = \gamma_Z^X$ - tree relation from X downto Z
- quotient $\alpha_Z^X / \beta_Z^Y = \gamma_Y^X$ - tree relation from X downto Y



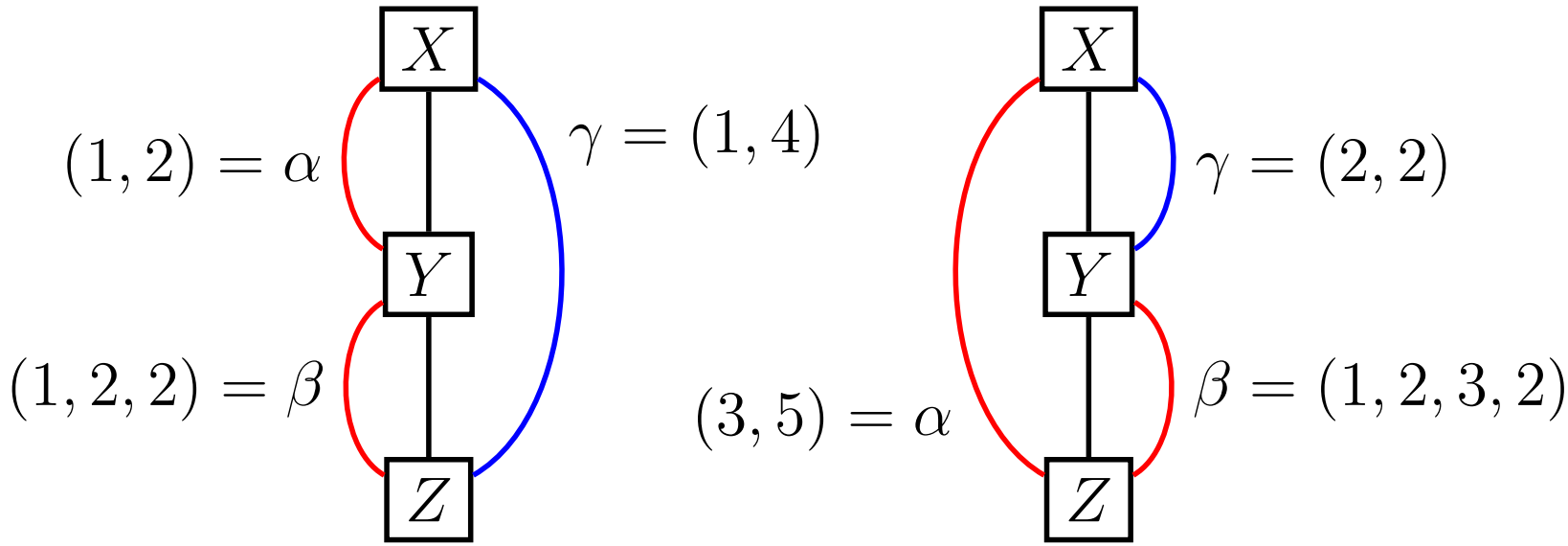
product



quotient

Operations on Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- product $\alpha_Y^X \cdot \beta_Z^Y = \gamma_Z^X$ - tree relation from X downto Z
- quotient $\alpha_Z^X / \beta_Z^Y = \gamma_Y^X$ - tree relation from X downto Y



product quotient

both operations can be computed in linear time

Coherence of Compositions

Coherence of Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree

Coherence of Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- coherence condition to encode a syntax tree:

Coherence of Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- coherence condition to encode a syntax tree:

$$\chi_Y^X \leq \chi_Z^X \qquad \text{i.e. } \chi_Y^X \text{ is smaller than } \chi_Z^X$$

Coherence of Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- coherence condition to encode a syntax tree:

$$\chi_Y^X \leq \chi_Z^X \qquad \text{i.e. } \chi_Y^X \text{ is smaller than } \chi_Z^X$$

$$\chi_Z^Y \preceq \chi_Z^X \qquad \text{i.e. } \chi_Z^Y \text{ is finer than } \chi_Z^X$$

Coherence of Compositions

- suppose $Z \lhd Y \lhd X$ in a hierarchy tree
- coherence condition to encode a syntax tree:

$$\chi_Y^X \leq \chi_Z^X \qquad \text{i.e. } \chi_Y^X \text{ is smaller than } \chi_Z^X$$

$$\chi_Z^Y \preceq \chi_Z^X \qquad \text{i.e. } \chi_Z^Y \text{ is finer than } \chi_Z^X$$

- a coherent set of compositions (i.e. a syntax tree) is closed with respect to both product and quotient

---

Membership Decision Procedure

Procedure Set-up

Procedure Set-up

- initial configuration of the decision procedure:

Procedure Set-up

- initial configuration of the decision procedure:
  - loop-only automaton A

Procedure Set-up

- initial configuration of the decision procedure:
  - loop-only automaton A
  - dependence relation D

Procedure Set-up

- initial configuration of the decision procedure:
  - loop-only automaton A
  - dependence relation D
- input: string x to be analysed for membership

Procedure Set-up

- initial configuration of the decision procedure:
  - loop-only automaton A
  - dependence relation D
- input: string x to be analysed for membership
- try to build syntax tree of a string y s.t.

$$y \cong_D x \ \wedge \ y \in L(A)$$

Procedure Set-up

- initial configuration of the decision procedure:
  - loop-only automaton A
  - dependence relation D
- input: string x to be analysed for membership
- try to build syntax tree of a string y s.t.

$$y \cong_D x \ \wedge \ y \in L(A)$$

- output: answer "yes" if the syntax tree of string y has been fully built; answer "no" otherwise

Procedure Set-up

- initial configuration of the decision procedure:
  - loop-only automaton A
  - dependence relation D
- input: string x to be analysed for membership
- try to build syntax tree of a string y s.t.

$$y \cong_D x \ \wedge \ y \in L(A)$$

- output: answer "yes" if the syntax tree of string y has been fully built; answer "no" otherwise

---

- project string x onto the pairs of dependent letters (and onto the completely independent letters if any)

Procedure Set-up

- initial configuration of the decision procedure:
  - loop-only automaton A
  - dependence relation D
- input: string x to be analysed for membership
- try to build syntax tree of a string y s.t.

$$y \cong_D x \ \wedge \ y \in L(A)$$

- output: answer "yes" if the syntax tree of string y has been fully built; answer "no" otherwise

---

- project string x onto the pairs of dependent letters (and onto the completely independent letters if any)
- projections of x (given) and y (to be found) are identical

Procedure Sketch

Procedure Sketch

- decision procedure consists of three phases:

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree
  2. associate initial compositions with tree nodes and close w.r.t. product and quotient

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree
  2. associate initial compositions with tree nodes and close w.r.t. product and quotient
  3. construct tree edges between nodes:

     repeat

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree
  2. associate initial compositions with tree nodes and close w.r.t. product and quotient
  3. construct tree edges between nodes:

     repeat

     - find unconnected nodes $Y \lhd X$ with Y at max distance from the root and find a matching χ_Y^X

       coherently with existing comp.s - if impossible, stop and reject

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree
  2. associate initial compositions with tree nodes and close w.r.t. product and quotient
  3. construct tree edges between nodes:

     repeat
     - find unconnected nodes $Y \lhd X$ with Y at max distance from the root and find a matching χ_Y^X

       coherently with existing comp.s - if impossible, stop and reject
     - close w.r.t. product and quotient

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree
  2. associate initial compositions with tree nodes and close w.r.t. product and quotient
  3. construct tree edges between nodes:

     repeat
     - find unconnected nodes $Y \lhd X$ with Y at max distance from the root and find a matching χ_Y^X

       coherently with existing comp.s - if impossible, stop and reject
     - close w.r.t. product and quotient
     - check coherence of compositions

       if coherence check fails, stop and reject

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree
  2. associate initial compositions with tree nodes and close w.r.t. product and quotient
  3. construct tree edges between nodes:

     repeat

     - find unconnected nodes $Y \lhd X$ with Y at max distance from the root and find a matching χ_Y^X

       coherently with existing comp.s - if impossible, stop and reject
     - close w.r.t. product and quotient
     - check coherence of compositions

       if coherence check fails, stop and reject

     until nothing changes any more

Procedure Sketch

- decision procedure consists of three phases:
  1. compute the numbers of node types of the tree
  2. associate initial compositions with tree nodes and close w.r.t. product and quotient
  3. construct tree edges between nodes:

     repeat

     - find unconnected nodes $Y \lhd X$ with Y at max distance from the root and find a matching χ_Y^X

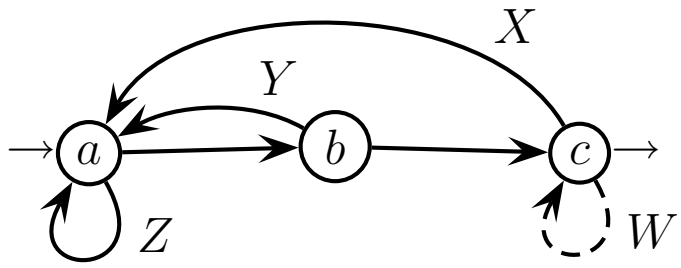
       coherently with existing comp.s - if impossible, stop and reject
     - close w.r.t. product and quotient
     - check coherence of compositions

       if coherence check fails, stop and reject

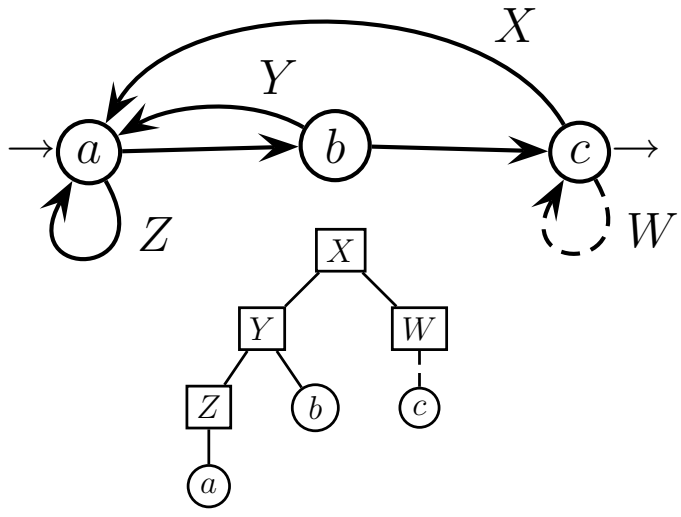
     until nothing changes any more
- now syntax tree is coherent and complete - accept

Node Count

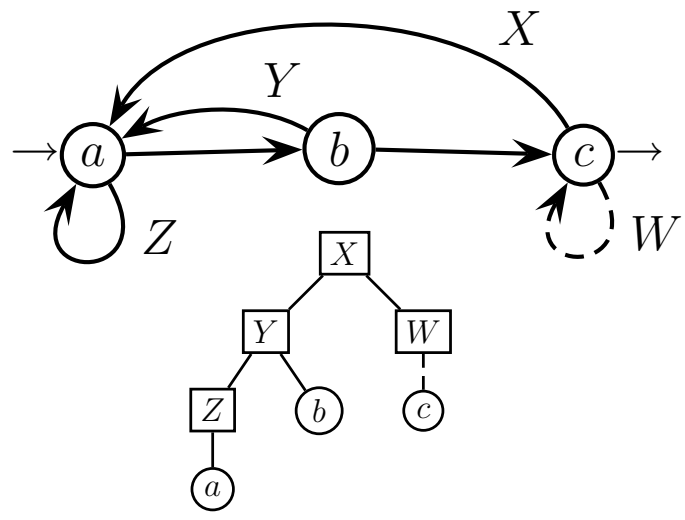
Node Count



Node Count



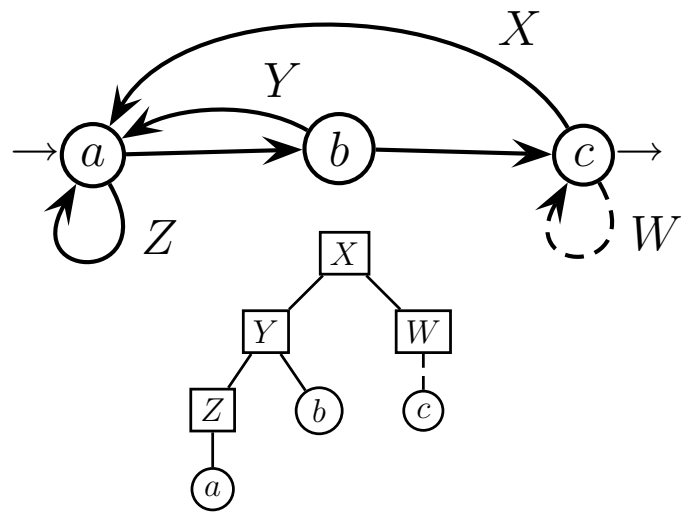
Node Count



dependent letters: $a\,b$ and $a\,c$

$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

Node Count

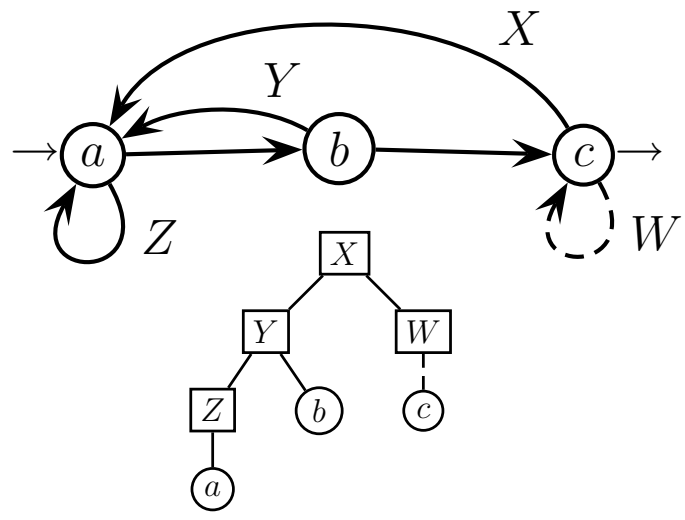


dependent letters: $a\,b$ and $a\,c$

loop	pair
X	$a\,c$
Y	$a\,b,\ b$
Z	a

$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

Node Count



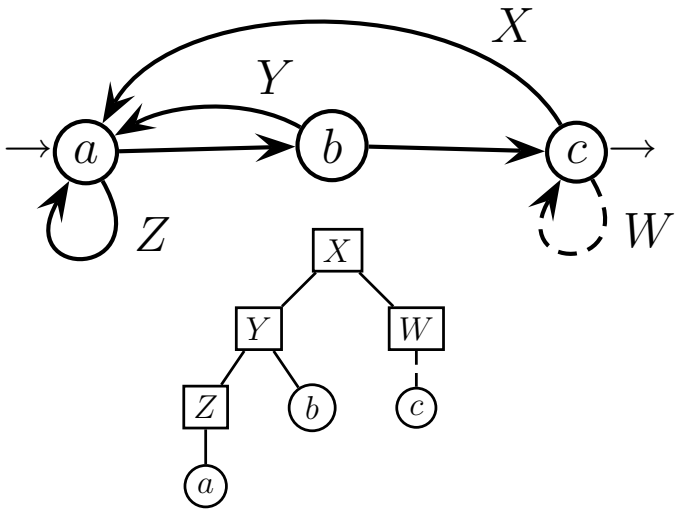
$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

dependent letters: $a\,b$ and $a\,c$

loop	pair
X	$a\,c$
Y	$a\,b,\ b$
Z	a

$$\begin{aligned} \pi_{a\,c}(x) &= a\,a\,a\,c\,a\,a\,c \\ \pi_{a\,b}(x) &= a\,b\,a\,a\,b\,a\,a\,b \\ \pi_{b}(x) &= b\,b\,b \\ \pi_{a}(x) &= a\,a\,a\,a\,a \end{aligned}$$

Node Count



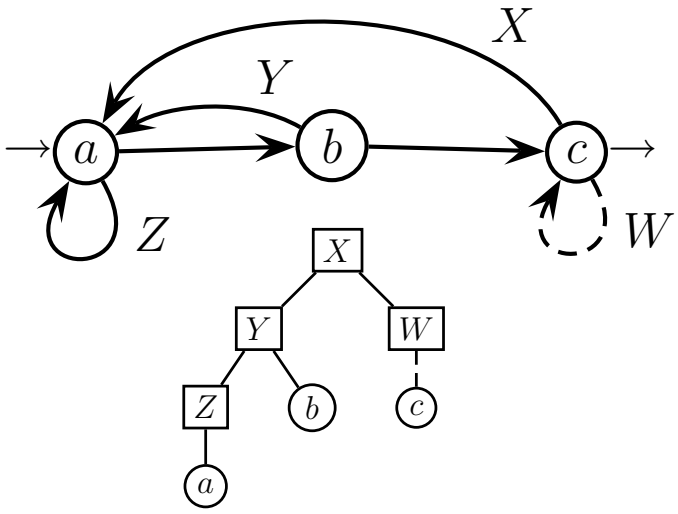
$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

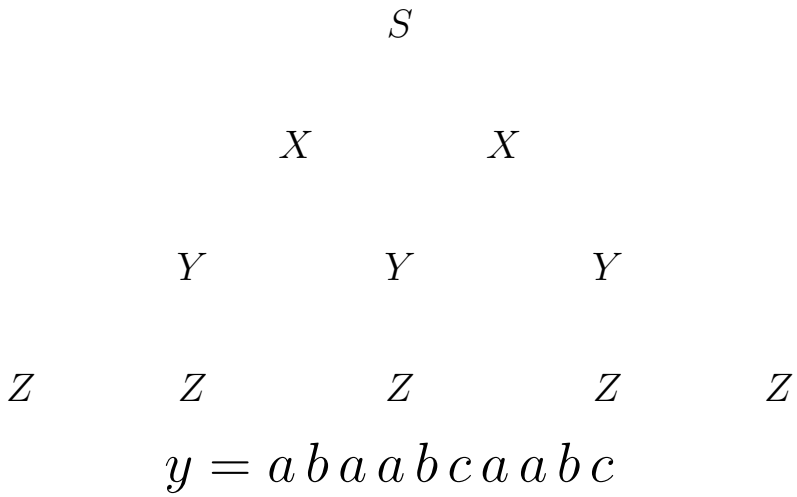
dependent letters: $a\,b$ and $a\,c$

loop	pair
X	$a\,c$
Y	$a\,b,\ b$
Z	a

$$\pi_{a\,c}(x) = a\,a\,a\,c\,a\,a\,c \qquad \#X = \#a\,c = 2$$
$$\pi_{a\,b}(x) = a\,b\,a\,a\,b\,a\,a\,b \qquad \#Y = \#a\,b = 3$$
$$\pi_{b}(x) = b\,b\,b \qquad \#Y = \#b = 3$$
$$\pi_{a}(x) = a\,a\,a\,a\,a \qquad \#Z = \#a = 5$$

Node Count



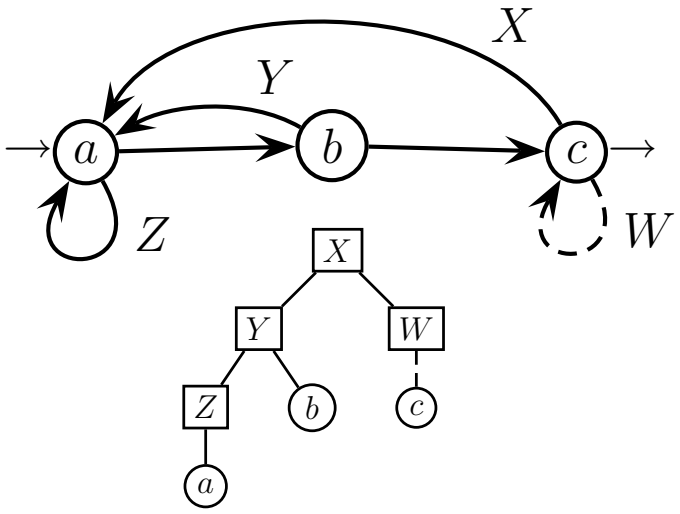


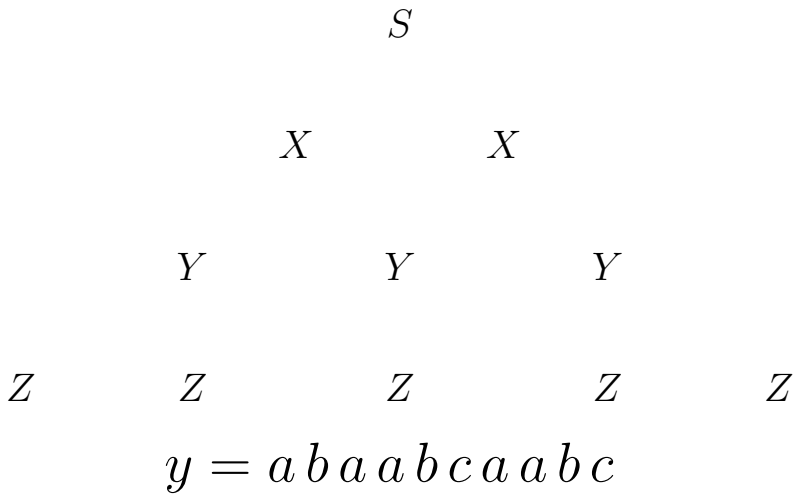
dependent letters: $a\,b$ and $a\,c$

loop	pair
X	$a\,c$
Y	$a\,b,\ b$
Z	a

$$\pi_{a\,c}(x) = a\,a\,a\,c\,a\,a\,c \qquad \#X = \#a\,c = 2$$
$$\pi_{a\,b}(x) = a\,b\,a\,a\,b\,a\,a\,b \qquad \#Y = \#a\,b = 3$$
$$\pi_{b}(x) = b\,b\,b \qquad \#Y = \#b = 3$$
$$\pi_{a}(x) = a\,a\,a\,a\,a \qquad \#Z = \#a = 5$$

Node Count





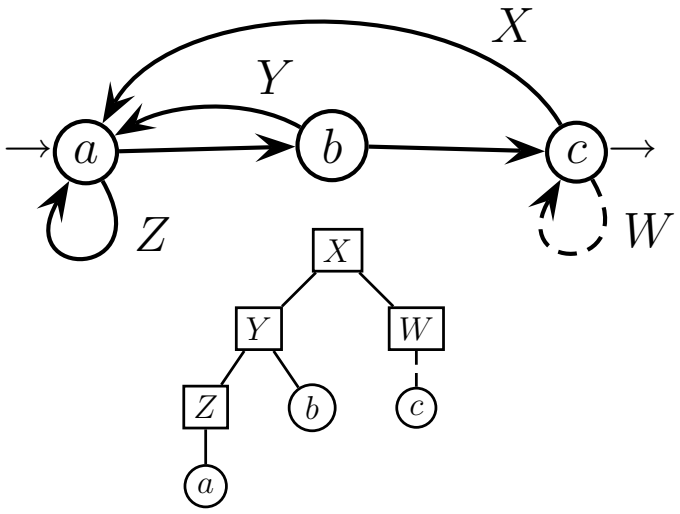
dependent letters: $a\,b$ and $a\,c$

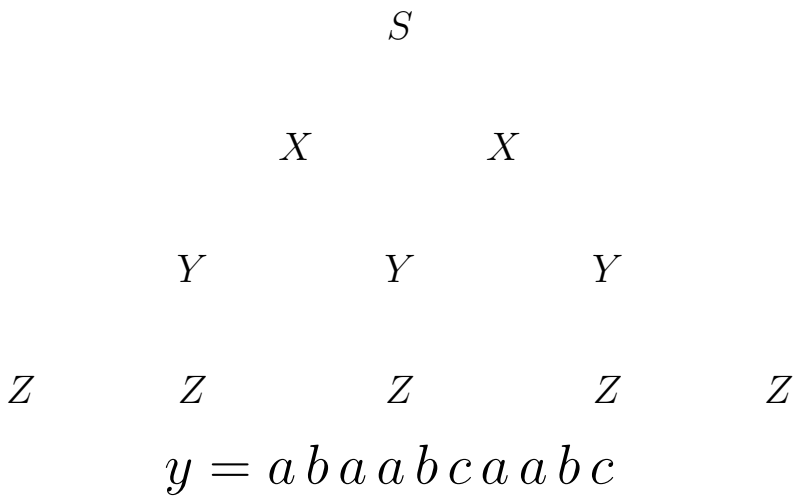
loop	pair
X	$a\,c$
Y	$a\,b,\ b$
Z	a

$$\pi_{a\,c}(x) = a\,a\,a\,c\,a\,a\,c \qquad \#X = \#a\,c = 2$$
$$\pi_{a\,b}(x) = a\,b\,a\,a\,b\,a\,a\,b \qquad \#Y = \#a\,b = 3$$
$$\pi_{b}(x) = b\,b\,b \qquad \#Y = \#b = 3$$
$$\pi_{a}(x) = a\,a\,a\,a\,a \qquad \#Z = \#a = 5$$

- to identify loops use dependent letter pairs

Node Count





dependent letters: $a\,b$ and $a\,c$

loop	pair
X	$a\,c$
Y	$a\,b,\ b$
Z	a

$$\pi_{a\,c}(x) = a\,a\,a\,c\,a\,a\,c \qquad \#X = \#a\,c = 2$$
$$\pi_{a\,b}(x) = a\,b\,a\,a\,b\,a\,a\,b \qquad \#Y = \#a\,b = 3$$
$$\pi_{b}(x) = b\,b\,b \qquad \#Y = \#b = 3$$
$$\pi_{a}(x) = a\,a\,a\,a\,a \qquad \#Z = \#a = 5$$

- to identify loops use dependent letter pairs
- for unidentified loops number of iterations is free

Composition Association

Composition Association

- use projections on dependent letter pairs:

Composition Association

- use projections on dependent letter pairs:

$$loop\,(a) = Z \qquad loop\,(a\,b) = Y$$

Composition Association

- use projections on dependent letter pairs:

$$loop\,(a) = Z \qquad loop\,(a\,b) = Y$$

$$\pi_{a\,b}\,(x) = \underbrace{\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}$$

Composition Association

- use projections on dependent letter pairs:

$$loop\,(a) = Z \qquad loop\,(a\,b) = Y$$

$$\pi_{a\,b}\,(x) = \underbrace{\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}$$

$$\chi_Z^Y = (1, 2, 2)$$

Composition Association

- use projections on dependent letter pairs:

$$loop\,(a) = Z \qquad loop\,(a\,b) = Y$$

$$\pi_{a\,b}\,(x) = \underbrace{\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}$$

$$\chi_Z^Y = (1, 2, 2)$$

$$loop\,(a) = Z \qquad loop\,(a\,c) = X$$

$$\pi_{a\,c}\,(x) = a\,a\,a\,c\,a\,a\,c \qquad \chi_Z^X = (3, 2)$$

Composition Association

- use projections on dependent letter pairs:

$$loop\,(a) = Z \qquad loop\,(a\,b) = Y$$

$$\pi_{a\,b}\,(x) = \underbrace{\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}\ \underbrace{\underbrace{a}_{Z}\,\underbrace{a}_{Z}\,b}_{Y}$$

$$\chi_Z^Y = (1, 2, 2)$$

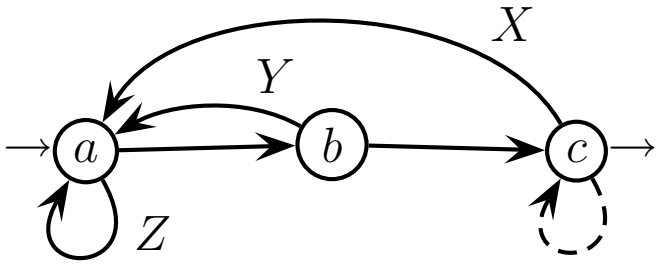
$$loop\,(a) = Z \qquad loop\,(a\,c) = X$$

$$\pi_{a\,c}\,(x) = a\,a\,a\,c\,a\,a\,c \qquad \chi_Z^X = (3, 2)$$

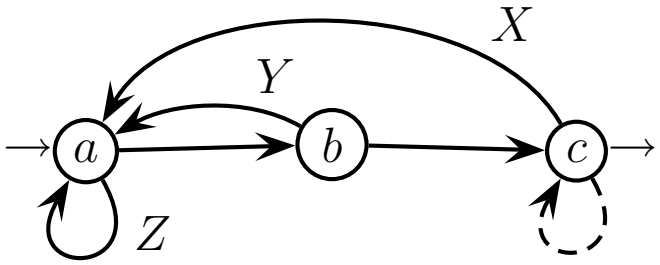
- do so for every pair of dependent letters

Tree Closure

Tree Closure



Tree Closure



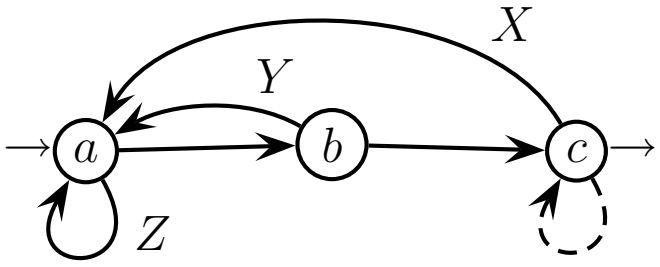
$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

$$Z \lhd Y \lhd X$$

$$\chi_Z^X = (3, 2)$$

$$\chi_Z^Y = (1, 2, 2)$$

Tree Closure



$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

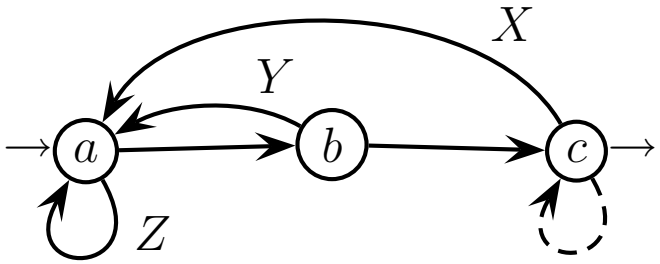
$$Z \lhd Y \lhd X$$

$$\chi_Z^X = (3, 2)$$

$$\chi_Z^Y = (1, 2, 2)$$

- quotient $\chi_Z^X / \chi_Z^Y = (3, 2) \,/\, (1, 2, 2) = (2, 1) = \chi_Y^X$

Tree Closure

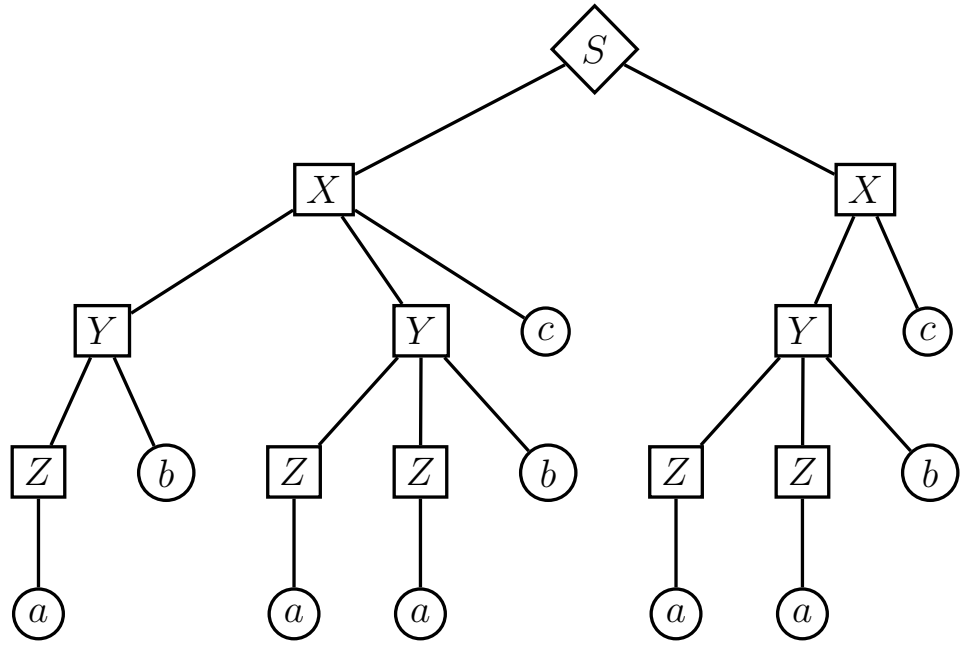


$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

$$Z \lhd Y \lhd X$$

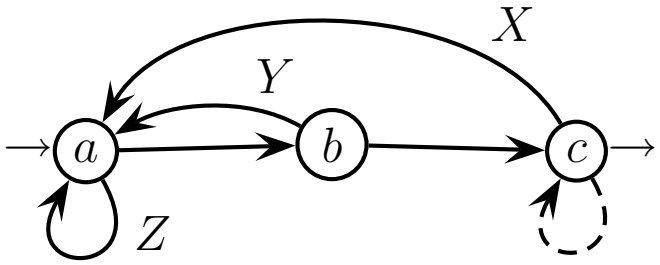
$$\chi_Z^X = (3, 2)$$

$$\chi_Z^Y = (1, 2, 2)$$



- quotient $\chi_Z^X / \chi_Z^Y = (3, 2) \,/\, (1, 2, 2) = (2, 1) = \chi_Y^X$

Tree Closure

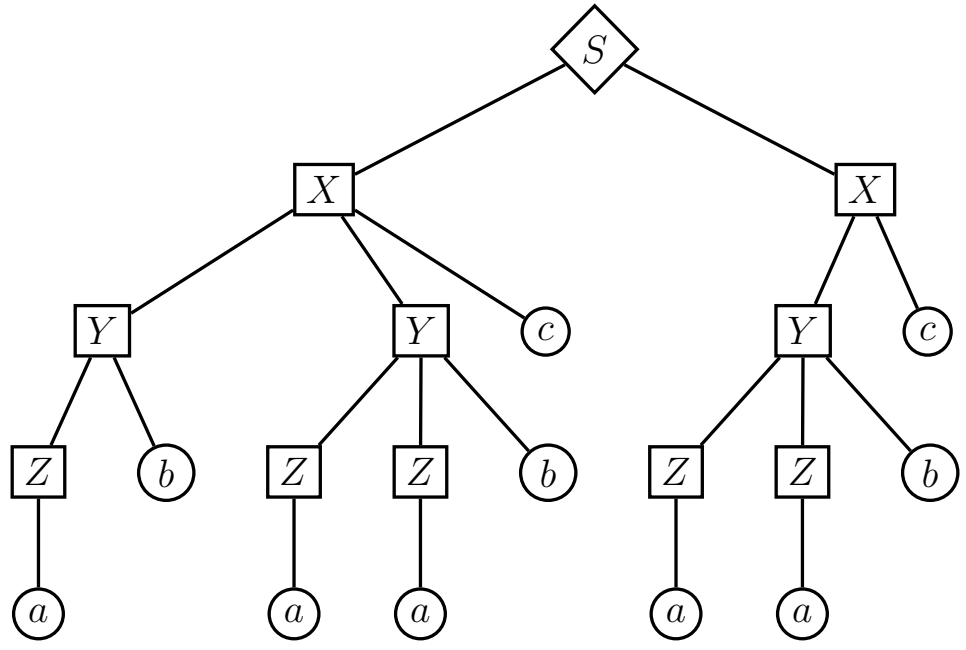


$$y = a\,b\,a\,a\,b\,c\,a\,a\,b\,c$$

$$Z \lhd Y \lhd X$$

$$\chi_Z^X = (3, 2)$$

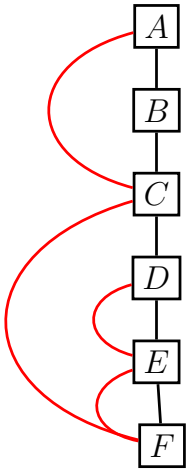
$$\chi_Z^Y = (1, 2, 2)$$



- quotient $\chi_Z^X / \chi_Z^Y = (3, 2) \,/\, (1, 2, 2) = (2, 1) = \chi_Y^X$
- tree complete - procedure end - accept string

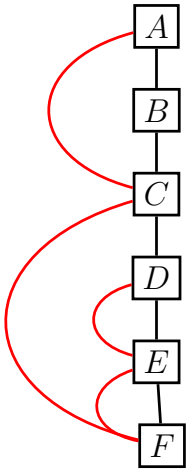
Example with Matching

Example with Matching

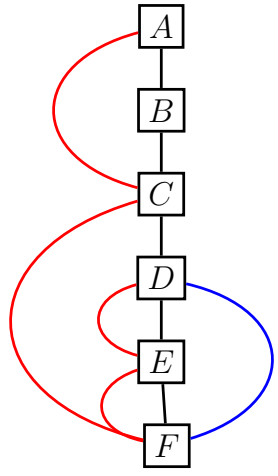


initial

Example with Matching

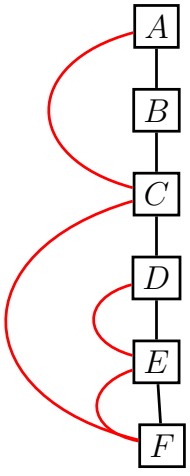


initial

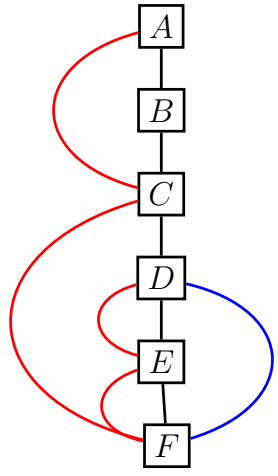


product $\chi_E^D \cdot \chi_F^E = \chi_F^D$

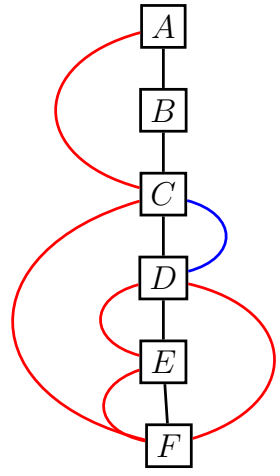
Example with Matching



initial

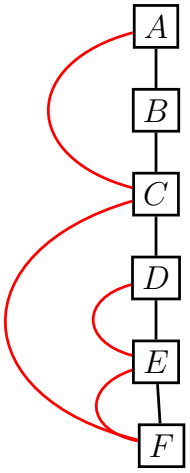


product $\chi_E^D \cdot \chi_F^E = \chi_F^D$

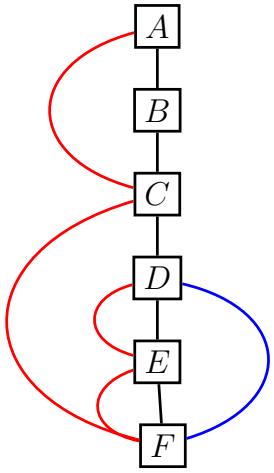


quotient $\chi_F^C / \chi_F^D = \chi_D^C$

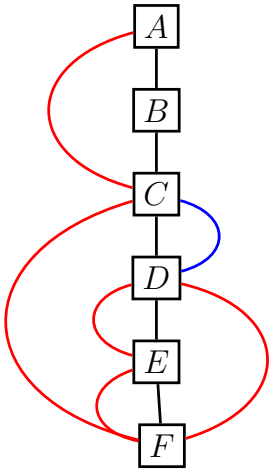
Example with Matching



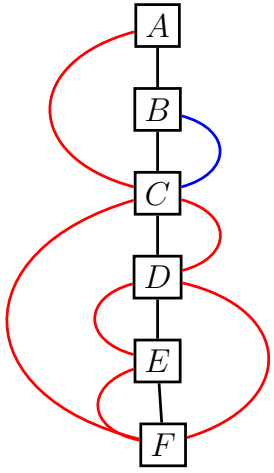
initial



product $\chi_E^D \cdot \chi_F^E = \chi_F^D$

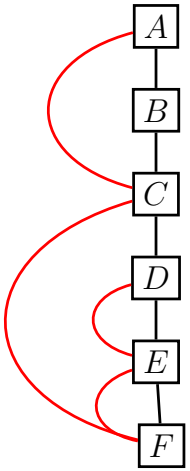


quotient $\chi_F^C / \chi_F^D = \chi_D^C$

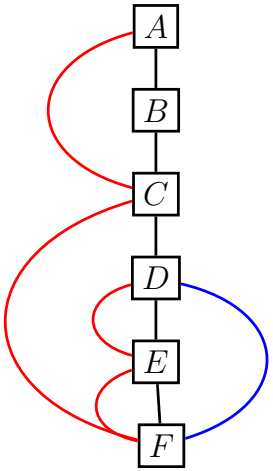


matching $\chi_C^B \preceq \chi_C^A$

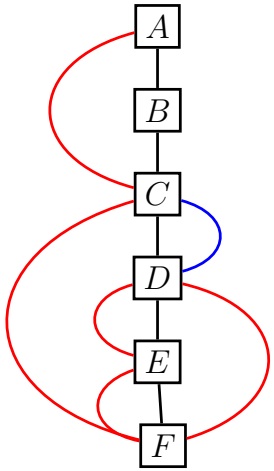
Example with Matching



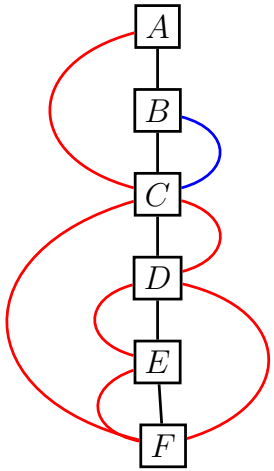
initial



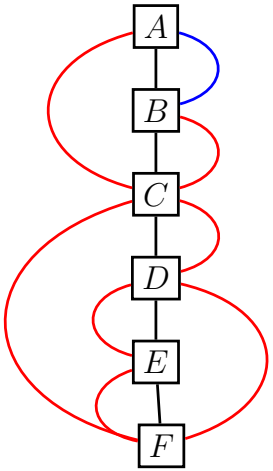
product $\chi_E^D \cdot \chi_F^E = \chi_F^D$



quotient $\chi_F^C / \chi_F^D = \chi_D^C$

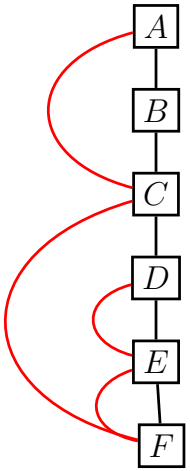


matching $\chi_C^B \preceq \chi_C^A$

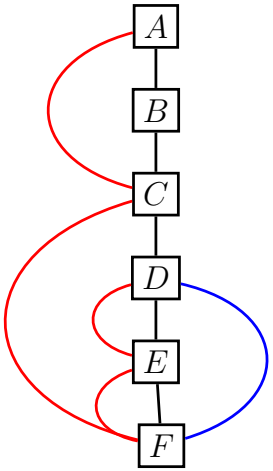


quotient $\chi_C^A / \chi_C^B = \chi_B^A$

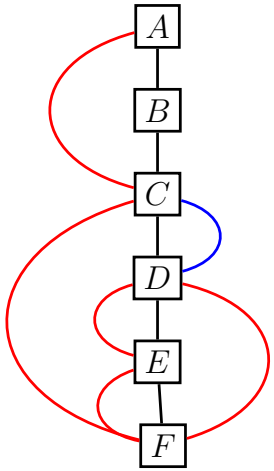
Example with Matching



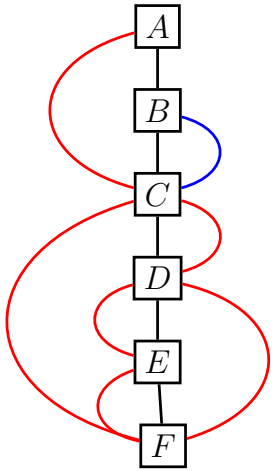
initial



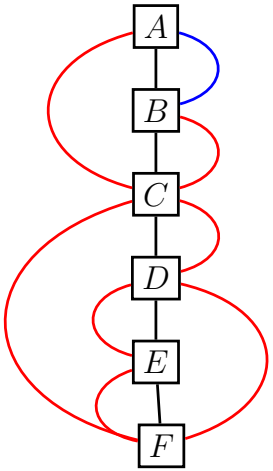
product $\chi_E^D \cdot \chi_F^E = \chi_F^D$



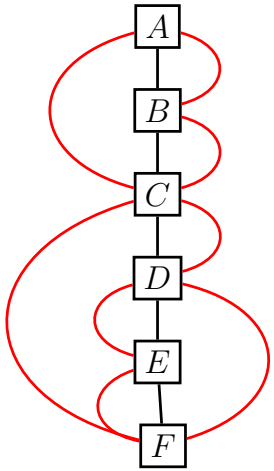
quotient $\chi_F^C / \chi_F^D = \chi_D^C$



matching $\chi_C^B \preceq \chi_C^A$

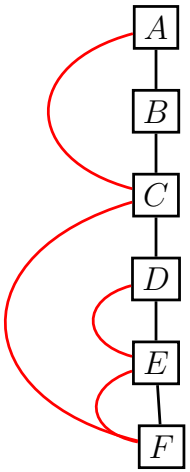


quotient $\chi_C^A / \chi_C^B = \chi_B^A$

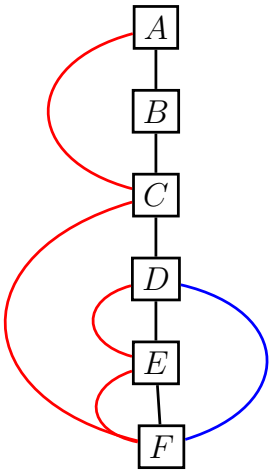


final

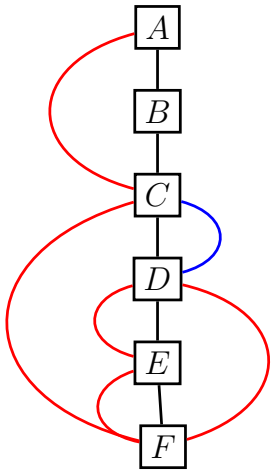
Example with Matching



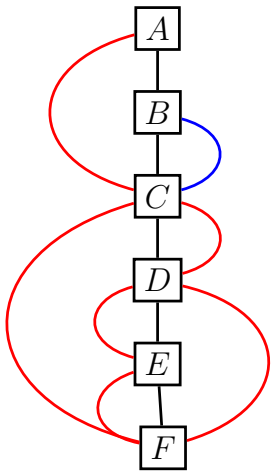
initial



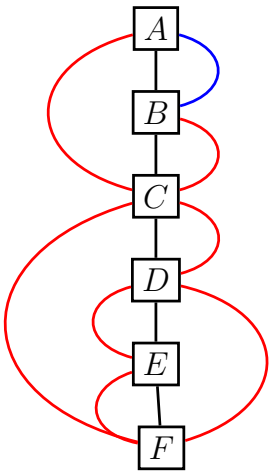
product $\chi_E^D \cdot \chi_F^E = \chi_F^D$



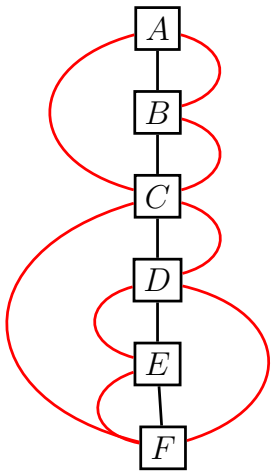
quotient $\chi_F^C / \chi_F^D = \chi_D^C$



matching $\chi_C^B \preceq \chi_C^A$



quotient $\chi_C^A / \chi_C^B = \chi_B^A$



final

only the relevant products and quotients are shown

Matching Operation

What is Matching ?

What is Matching ?

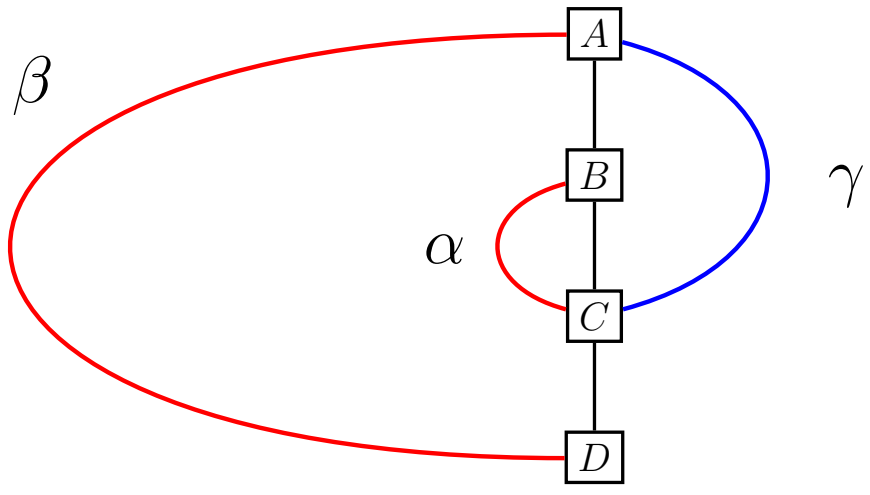
- consider four ordered nodes $D \lhd C \lhd B \lhd A$

What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A

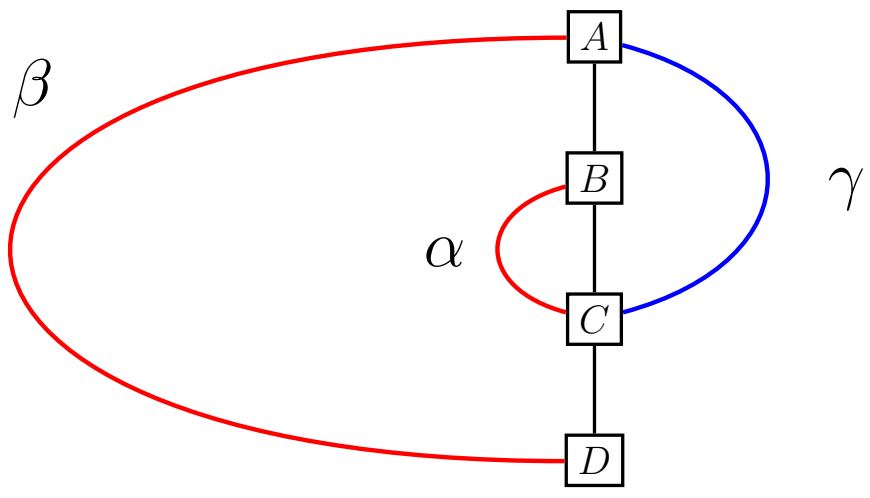
What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A



What is Matching ?

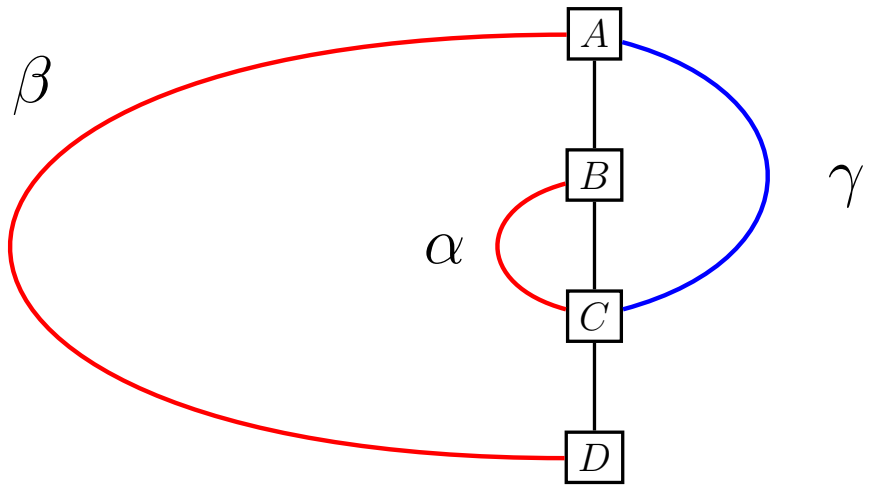
- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A
- by coherence composition γ has to satisfy:



What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A
- by coherence composition γ has to satisfy:

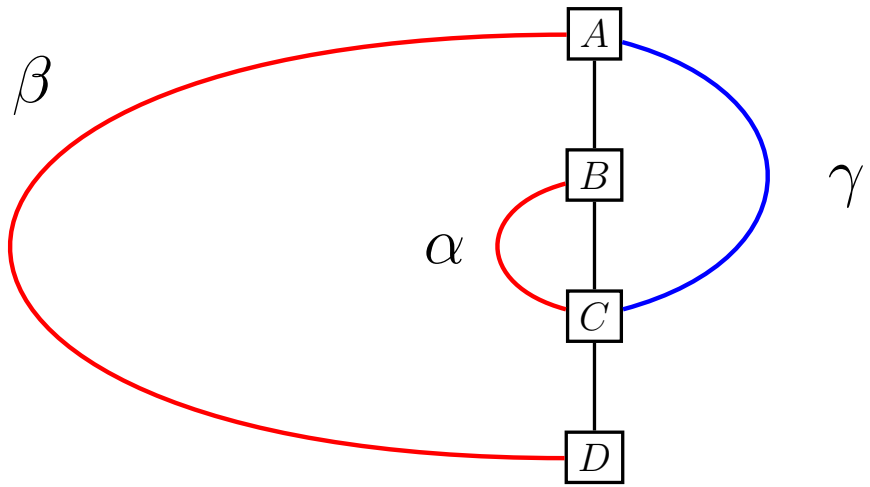
$$\gamma \leq \beta$$



What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A
- by coherence composition γ has to satisfy:

$$\gamma \leq \beta \quad \text{and} \quad \alpha \preceq \gamma$$

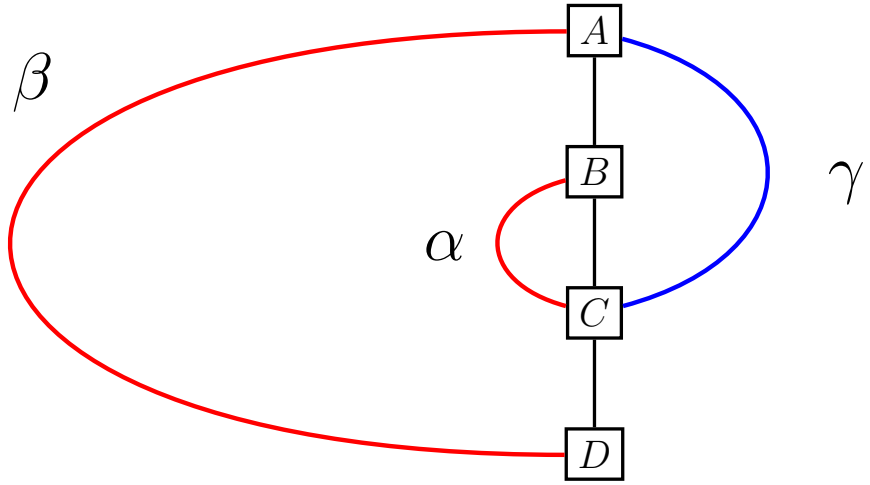


What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A
- by coherence composition γ has to satisfy:

$$\gamma \leq \beta \quad \text{and} \quad \alpha \preceq \gamma$$

- γ is a match of α and β - then complete tree



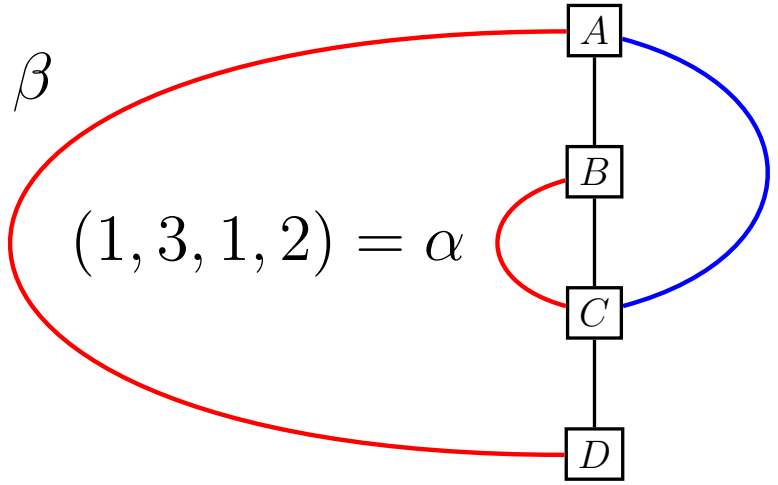
What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A
- by coherence composition γ has to satisfy:

$$\gamma \leq \beta \quad \text{and} \quad \alpha \preceq \gamma$$

- γ is a <u>match</u> of α and β - then complete tree

$(1,5,3) = \beta$



$(1,3,1,2) = \alpha$

$\gamma = (1,4,2)$

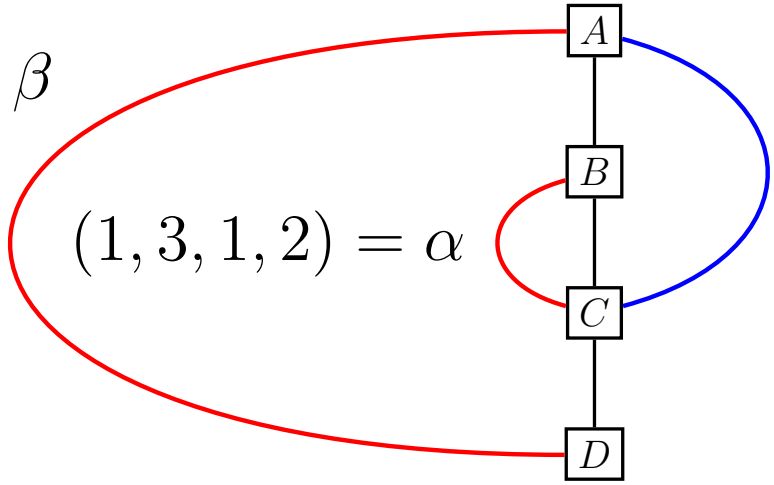
What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A
- by coherence composition γ has to satisfy:

$$\gamma \leq \beta \quad \text{and} \quad \alpha \preceq \gamma$$

- γ is a match of α and β - then complete tree

$(1,5,3) = \beta$



two matches

$\gamma = (1,4,2)$

$\gamma = (1,3,3)$

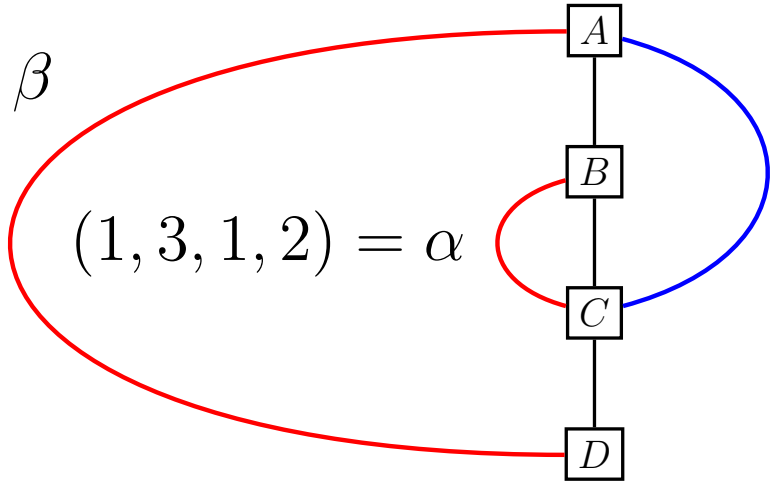
What is Matching ?

- consider four ordered nodes $D \lhd C \lhd B \lhd A$
- want to "match" two comp.s α_C^B and β_D^A by γ_C^A
- by coherence composition γ has to satisfy:

$$\gamma \leq \beta \quad \text{and} \quad \alpha \preceq \gamma$$

- γ is a match of α and β - then complete tree

$(1,5,3) = \beta$



$(1,3,1,2) = \alpha$

two matches

$\gamma = (1,4,2)$

$\gamma = (1,3,3)$

- one can flip $\leq$ and $\preceq$ in the other three ways

Complexity of Matching

Complexity of Matching

Basic Algorithm:

Complexity of Matching

Basic Algorithm:

- match existence is decidable in cubic time
  (Savelli et alii - Ph.D. thesis 2006 - WORDS Monreal 2005)

Complexity of Matching

Basic Algorithm:

- match existence is decidable in cubic time
  (Savelli et alii - Ph.D. thesis 2006 - WORDS Monreal 2005)
- one match can be found in cubic time as well

Complexity of Matching

Basic Algorithm:

- match existence is decidable in cubic time
  (Savelli et alii - Ph.D. thesis 2006 - WORDS Monreal 2005)
- one match can be found in cubic time as well
- all matches can be represented in cubic space

Complexity of Matching

Basic Algorithm:

- match existence is decidable in cubic time
  (Savelli et alii - Ph.D. thesis 2006 - WORDS Monreal 2005)
- one match can be found in cubic time as well
- all matches can be represented in cubic space

Optimised Algorithm:

Complexity of Matching

Basic Algorithm:

- match existence is decidable in cubic time
  (Savelli et alii - Ph.D. thesis 2006 - WORDS Monreal 2005)
- one match can be found in cubic time as well
- all matches can be represented in cubic space

Optimised Algorithm:

- match existence is decidable in quadratic time
  (Goldwurm, Breveglieri, Crespi - 2007-08)

Complexity of Matching

Basic Algorithm:

- match existence is decidable in cubic time
  (Savelli et alii - Ph.D. thesis 2006 - WORDS Monreal 2005)
- one match can be found in cubic time as well
- all matches can be represented in cubic space

Optimised Algorithm:

- match existence is decidable in quadratic time
  (Goldwurm, Breveglieri, Crespi - 2007-08)
- one match can be found in quadratic time as well

Observations on Matching

Observations on Matching

- the matching operation is partial
  - may have no result

Observations on Matching

- the matching operation is partial
  - may have no result
- the matching operation is multi-valued
  - may have two or more results

Observations on Matching

- the matching operation is partial
  - may have no result
- the matching operation is multi-valued
  - may have two or more results
- a given integer composition can be obtained from two or even more different matching operations

Observations on Matching

- the matching operation is partial
  - may have no result
- the matching operation is multi-valued
  - may have two or more results
- a given integer composition can be obtained from two or even more different matching operations
- if every matching operation has either one or no result (but not two or more), then the decision procedure answers the membership problem in quadratic time
  - the procedure builds the syntax tree as well

Conclusion

Results and Open Problems

Results and Open Problems

- formalisation of the membership problem for a family of non-regular trace languages of applicative interest

  (e.g. in the parallel scheduling of machine instructions)

Results and Open Problems

- formalisation of the membership problem for a family of non-regular trace languages of applicative interest

  (e.g. in the parallel scheduling of machine instructions)

- original approach to the problem (i.e. numerical) and different from the standard one (i.e. trace prefixes)

Results and Open Problems

- formalisation of the membership problem for a family of non-regular trace languages of applicative interest

  (e.g. in the parallel scheduling of machine instructions)

- original approach to the problem (i.e. numerical) and different from the standard one (i.e. trace prefixes)
- membership procedure that works in polynomial time with a fixed exponent (quadratic) for non-trivial cases

Results and Open Problems

- formalisation of the membership problem for a family of non-regular trace languages of applicative interest

  (e.g. in the parallel scheduling of machine instructions)

- original approach to the problem (i.e. numerical) and different from the standard one (i.e. trace prefixes)
- membership procedure that works in polynomial time with a fixed exponent (quadratic) for non-trivial cases
- open problem: when the matching operation has two or more solutions, procedure becomes indeterministic

  (not obvious that time complexity is still polynomial)

Results and Open Problems

- formalisation of the membership problem for a family of non-regular trace languages of applicative interest

  (e.g. in the parallel scheduling of machine instructions)

- original approach to the problem (i.e. numerical) and different from the standard one (i.e. trace prefixes)
- membership procedure that works in polynomial time with a fixed exponent (quadratic) for non-trivial cases
- open problem: when the matching operation has two or more solutions, procedure becomes indeterministic

  (not obvious that time complexity is still polynomial)

- open problem: find a sufficient condition to grant that the matching operation has at most one solution