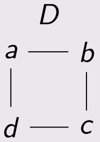
Algorythmics of posets generated by words over partially commutative alphabets

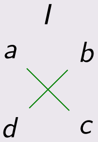
Łukasz Mikulski, Marcin Piątkowski, Sebastian Smyczyński

Prague, 29-31 August 2011

Concurrent alphabet

To the alphabet e.g. $\Sigma = \{ a, b, c, d\}$ we add the dependency relation or, equivalently, the indepedency relation





Words *abbaacd* and *abbcaad* are equivalent.

One word can be obtained from the other using transitions of the two consecutive letters.

Concurrent words and Poset

The dependency graph of a word

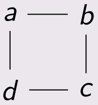
$$\Sigma = \{\, a, b, c, d \}\qquad D = \begin{array}{ccc} a & — & b \\ | & & | \\ d & — & c \end{array}$$

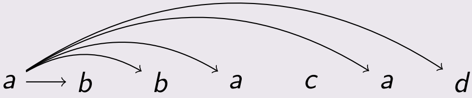
$$a \quad b \quad b \quad a \quad c \quad a \quad d$$

Concurrent words and Poset

The dependency graph of a word

$\Sigma = \{\ a, b, c, d\}$ $\quad D =$

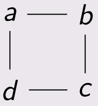


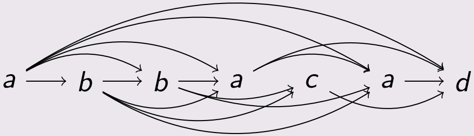


Concurrent words and Poset

The dependency graph of a word

$$\Sigma = \{\ a, b, c, d\}$$

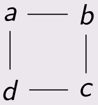
$D =$ 

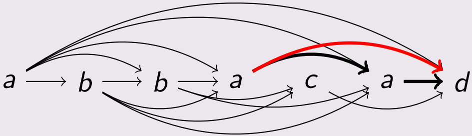


Concurrent words and Poset

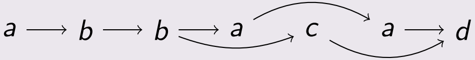
The dependency graph of a word

$\Sigma = \{\ a, b, c, d\}$ $D =$





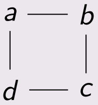
Transitive reduction (Hasse diagram)

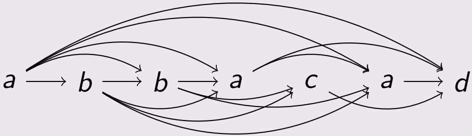


Concurrent words and Poset

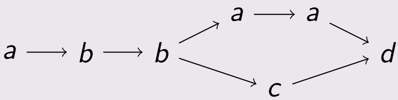
The dependency graph of a word

$\Sigma = \{\ a, b, c, d\}$ $\qquad D =$





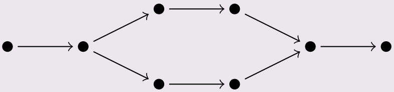
Transitive reduction (Hasse diagram)



Concurrent words and Poset

Compressed version of a poset

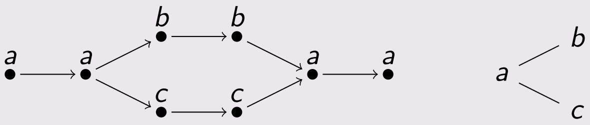
Every partially ordered set is uniquelly identified by the Hasse diagram



Concurrent words and Poset

Compressed version of a poset

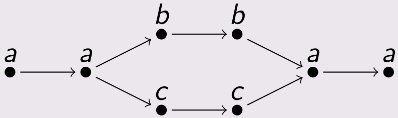
Every partially ordered set is uniquelly identified by the Hasse diagram

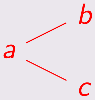


Concurrent words and Poset

Compressed version of a poset

Every partially ordered set is uniquelly identified by the Hasse diagram





Every word obtained from topological sorting of this graph (the linearisation of a poset) may be used as a representative:

aabbccaa, *aabccbaa*, *aabcbcaa*, *aacbbcaa*, *aacbcbaa*, *aaccbbaa*

Definition of the problem

For given concurrent alphabet (Σ, D) and word $w \in \Sigma^*$ reproduce Hasse diagram of a poset (transitive reduction of word's dependency graph).

Definition of the problem

For given concurrent alphabet (Σ, D) and word $w \in \Sigma^*$ reproduce Hasse diagram of a poset (transitive reduction of word's dependency graph).

Facts

- Edges of dependency graph are given by the relation $w_i E w_j \Leftrightarrow i < j \land w_i D w_j$.

Definition of the problem

For given concurrent alphabet (Σ, D) and word $w \in \Sigma^*$ reproduce Hasse diagram of a poset (transitive reduction of word's dependency graph).

Facts

- Edges of dependency graph are given by the relation $w_i E w_j \Leftrightarrow i < j \wedge w_i D w_j$.
- For a word w of length n the dependency graph may have $O(n^2)$ edges.

Definition of the problem

For given concurrent alphabet (Σ, D) and word $w \in \Sigma^*$ reproduce Hasse diagram of a poset (transitive reduction of word's dependency graph).

Facts

- Edges of dependency graph are given by the relation $w_i E w_j \Leftrightarrow i < j \wedge w_i D w_j$.
- For a word w of length n the dependency graph may have $O(n^2)$ edges.
- General algorithms of graph's transitive reduction have time complexity of $O(|V||E|) \sim O(n^3)$.

Definition of the problem

For given concurrent alphabet (Σ, D) and word $w \in \Sigma^*$ reproduce Hasse diagram of a poset (transitive reduction of word's dependency graph).

Facts

- Edges of dependency graph are given by the relation $w_i E w_j \Leftrightarrow i < j \wedge w_i D w_j$.
- For a word w of length n the dependency graph may have $O(n^2)$ edges.
- General algorithms of graph's transitive reduction have time complexity of $O(|V||E|) \sim O(n^3)$.
- It is possible to reduce this problem to the multiplication of boolean matrices - $O(n^{2.81\cdots})$.

Fact

If there exists an edge between $w_i = a$ and $w_j = b$ in the Hasse diagram, then letters a, b do not appear in the word w between indices i, j.

Fact

If there exists an edge between $w_i = a$ and $w_j = b$ in the Hasse diagram, then letters a, b do not appear in the word w between indices i, j.

Conclusions

- For every vertex there can be at most $k = |\Sigma|$ outgoing edges.
- For every vertex there can be at most k ingoing edges.

Fact

If there exists an edge between $w_i = a$ and $w_j = b$ in the Hasse diagram, then letters a, b do not appear in the word w between indices i, j.

Conclusions

- For every vertex there can be at most $k = |\Sigma|$ outgoing edges.
- For every vertex there can be at most k ingoing edges.
- Ingoing edges are determined by the last occurrences of the letters dependent with the label of a vertex.

Online algorithm of constructing Hasse diagram

Idea of the algorithm

$$D = \begin{array}{ccc} a & \text{---} & b \\ | & & | \\ d & \text{---} & c \end{array}, \quad w = adbcb$$

	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(b, a, c)	$\{a, b\}$	3
c	(b, d, c)	$\emptyset$	0
d	(d, a, c)	$\{a, d\}$	2

$V = \emptyset$

Online algorithm of constructing Hasse diagram

Idea of the algorithm

$$D = \begin{array}{ccc} a & — & b \\ | & & | \\ d & — & c \end{array}, \quad w = adbcb$$

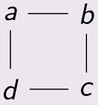
	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(b, a, c)	$\{a, b\}$	3
c	(b, d, c)	$\emptyset$	0
d	(d, a, c)	$\{a, d\}$	2

$$V = \emptyset$$

$$a_1 \longrightarrow d_2 \qquad b_3 \longrightarrow c_4 \quad (\text{arc } a_1 \to b_3)$$

Online algorithm of constructing Hasse diagram

Idea of the algorithm

$D =$ 

$, w = adbcb$

	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(b, a, c)	$\{a, b\}$	3
c	(b, d, c)	$\emptyset$	0
d	(d, a, c)	$\{a, d\}$	2

$V = \{a, b\}$

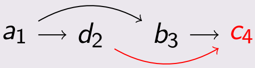
Online algorithm of constructing Hasse diagram

Idea of the algorithm

$$D = \begin{array}{ccc} a & — & b \\ | & & | \\ d & — & c \end{array}, \; w = adbcb$$

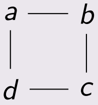
	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(b, a, c)	$\{a, b\}$	3
c	(b, d, c)	$\emptyset$	0
d	(d, a, c)	$\{a, d\}$	2

$V = \{a, b\}$



Online algorithm of constructing Hasse diagram

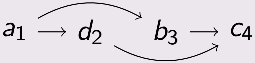
Idea of the algorithm

$D =$ 

$, w = adbcb$

	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(b, a, c)	$\{a, b\}$	3
c	(b, d, c)	$\emptyset$	0
d	(d, a, c)	$\{a, d\}$	2

$V = \{a, b, d\}$



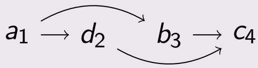
Online algorithm of constructing Hasse diagram

Idea of the algorithm

$$D = \begin{array}{ccc} a & — & b \\ | & & | \\ d & — & c \end{array}, \quad w = adbcb$$

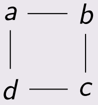
	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(b, a, c)	$\{a, b\}$	3
c	(b, d, c)	$\emptyset$	0
d	(d, a, c)	$\{a, d\}$	2

$$V = \{a, b, d\}$$



Online algorithm of constructing Hasse diagram

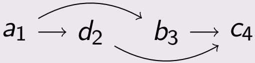
Idea of the algorithm

$D =$ 

$, w = adbcb$

	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(c, b, a)	$\{a, b\}$	3
c	(c, b, d)	$\{a, b, c, d\}$	4
d	(c, d, a)	$\{a, d\}$	2

$V = \{a, b, d\}$



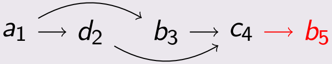
Online algorithm of constructing Hasse diagram

Idea of the algorithm

$$D = \begin{array}{ccc} a & — & b \\ | & & | \\ d & — & c \end{array}, \quad w = adbcb$$

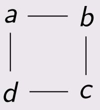
	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(c, b, a)	$\{a, b\}$	3
c	(c, b, d)	$\{a, b, c, d\}$	4
d	(c, d, a)	$\{a, d\}$	2

$V = \emptyset$



Online algorithm of constructing Hasse diagram

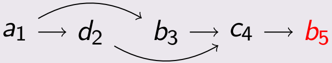
Idea of the algorithm

$D =$ 

$, \; w = adbcb$

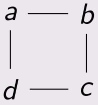
	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(c, b, a)	$\{a, b\}$	3
c	(c, b, d)	$\{a, b, c, d\}$	4
d	(c, d, a)	$\{a, d\}$	2

$V = \{a, b, c, d\}$



Online algorithm of constructing Hasse diagram

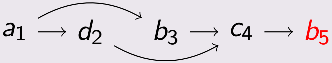
Idea of the algorithm

$D =$ 

$, \; w = adbcb$

	D	V	Pos
a	(b, d, a)	$\{a\}$	1
b	(b, c, a)	$\{a, b, c, d\}$	5
c	(b, c, d)	$\{a, c, d\}$	4
d	(c, d, a)	$\{a, d\}$	2

$V = \{a, b, c, d\}$



Algorithm 1: Hasse(w,(Σ, D))

```
foreach a ∈ Σ do
    L_a := 0; V_a := ∅;
for i := 1 to n do
    a := w_i; V := ∅;
    foreach b ∈ D_a in the order of last occ. do
        if L_b ≠ 0 and b ∉ V then
            Add the edge w_{L_b} → w_i;
            V := V ∪ V_b;
    foreach b ∈ Σ do
        V_b := V_b \ {a};
    foreach b ∈ D_a do
        Move a to the beginning of D_b;
    V_a := V ∪ a; L_a := i;
```

Online algorithm of constructing Hasse diagram

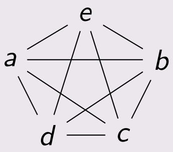
Time complexity

Given algorithm has time complexity $O(nk^2)$, where k is a size of the alphabet and n is the length of the word.

Definition of the problem

For given concurrent alphabet (Σ, D) and a natural number n generate every pairwise nonequivalent words of length n (lexicographically minimal representatives of posets).

Example

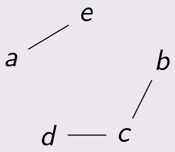


aa	*ba*	*ca*	*da*	*ea*
ab	*bb*	*cb*	*db*	*eb*
ac	*bc*	*cc*	*dc*	*ec*
ad	*bd*	*cd*	*dd*	*ed*
ae	*be*	*ce*	*de*	*ee*

Definition of the problem

For given concurrent alphabet (Σ, D) and a natural number n generate every pairwise nonequivalent words of length n (lexicographically minimal representatives of posets).

Example



aa	ba	ca	da	ea
ab	bb	cb	db	eb
ac	bc	cc	dc	ec
ad	bd	cd	dd	ed
ae	be	ce	de	ee

Definition of the problem

For given concurrent alphabet (Σ, D) and a natural number n generate every pairwise nonequivalent words of length n (lexicographically minimal representatives of posets).

Example

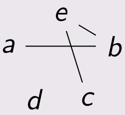
e

a b

d c

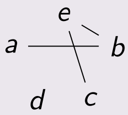
aa	*ba*	*ca*	*da*	*ea*
ab	*bb*	*cb*	*db*	*eb*
ac	*bc*	*cc*	*dc*	*ec*
ad	*bd*	*cd*	*dd*	*ed*
ae	*be*	*ce*	*de*	*ee*

Idea of the algoritm



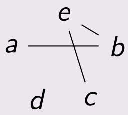
$ebceee \longrightarrow ebdaaa$

Idea of the algoritm



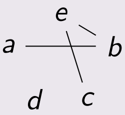
$$ebceee \longrightarrow ebeaaa$$

Idea of the algoritm



$ebceee \longrightarrow ebebaa$

Idea of the algoritm



$ebceee \longrightarrow ebebaa$

"Naive" algorithm

Preprocessing: $D_{min} =$

a	b	c	d	e
a	a	c	d	b

1. Find the latest letter w_i that is different that the largest
2. Change w_i to the next letter
3. Find out if actual prefix is minimal
4. Using the table D_{min} generate the sufix

Algorithm "with oracle"

$$V_1(a) = a$$

$$V_i(a) = \begin{cases} a & \text{if } aDw_{i-1} \\ \max(w_{i-1}, V_{i-1}(a)) & \text{otherwise} \end{cases}$$

- Cost of the preprocessing - $O(kn)$
- Finding out if the change is valid in time $O(1)$
- While generate sufix we must actualise the oracle

Algorithm "with oracle"

$$V_1(a) = a$$

$$V_i(a) = \begin{cases} a & \text{if } aDw_{i-1} \\ \max(w_{i-1}, V_{i-1}(a)) & \text{otherwise} \end{cases}$$

- Cost of the preprocessing - $O(kn)$
- Finding out if the change is valid in time $O(1)$
- While generate sufix we must actualise the oracle
- We have at most k different colums of an oracle, the cost of actualisation is $O(k \min(k, \#suff) + \#suff)$

Complexity comparison

Work of both algorithms is dependent on the length of changing suffix. Pesymistic cost of naive algorithm

$$O(k\ \#pref + \#suff)$$

cost of algorithm "with oracle"

$$O(k \min(k, \#suff) + \#suff)$$

Complexity comparison

Work of both algorithms is dependent on the length of changing suffix. Pesymistic cost of naive algorithm

$$O(n)$$

cost of algorithm "with oracle"

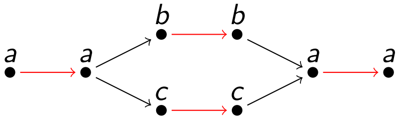
$$O(\#suff)$$

Definition of the problem

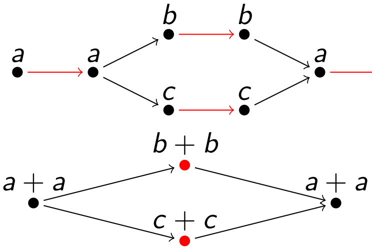
For given Hasse diagram write out in lexicographical order all linearisations.

- There are efective algorithms that do not preserve lexicographic order.
- Using technic similar to the algorithm of Hasse diagram generation, we can efectively write out linearisations in lexicographical order.

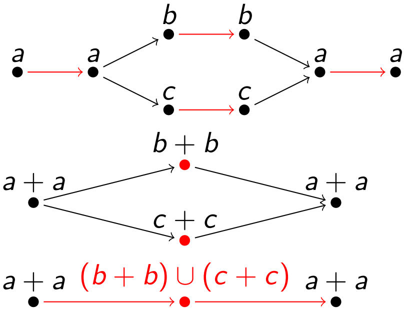
Combining neighbour actions into blocks



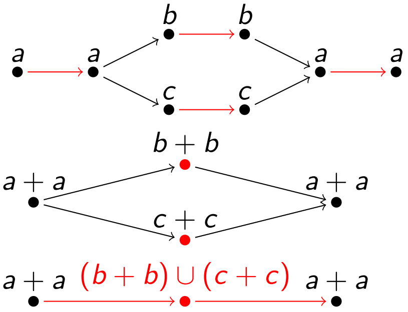
Combining neighbour actions into blocks



Combining neighbour actions into blocks



Combining neighbour actions into blocks



$$a + a + ((b + b) \cup (c + c)) + a + a$$

Series-parallel graphs

Definition

Series-parallel graph is a graph consisting of single vertex and no edges or is constructed form two series-parallel graphs using series and parallel composition.

Fact

Diagrams that may be compressed in presented way are precisely the series-parallel graphs.

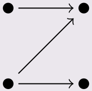
Fact

Many algorithmic problems are much simpler in the case of series-parallel graphs (posets).

N-free graphs

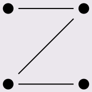
Fact

Graph is series-parallel if and only if it is N-free



Fakt

Hasse diagram of poset, which alphabet is N-free (P4-avoiding) is also N-free.



Thank you
for your attention.