Context unification is in PSPACE

Artur Jeż

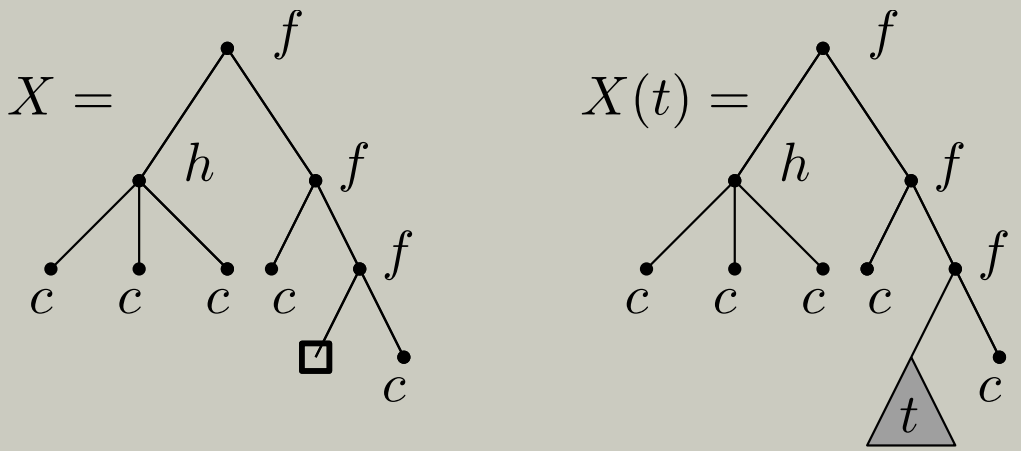
Max Planck Institute fur Informatik

10.07.2014

Context unification

Definition (Context unification)

Substitution for X uses argument exactly once.



$$X(f(Y(a), b) = f(Y(b), Y(a))$$

More formally

- signature (fixed arities) $[f, a, c]$
- context variables denoting terms with one 'hole' $[X()]$
- variables denoting closed terms $[x]$
- equations built with them

More formally

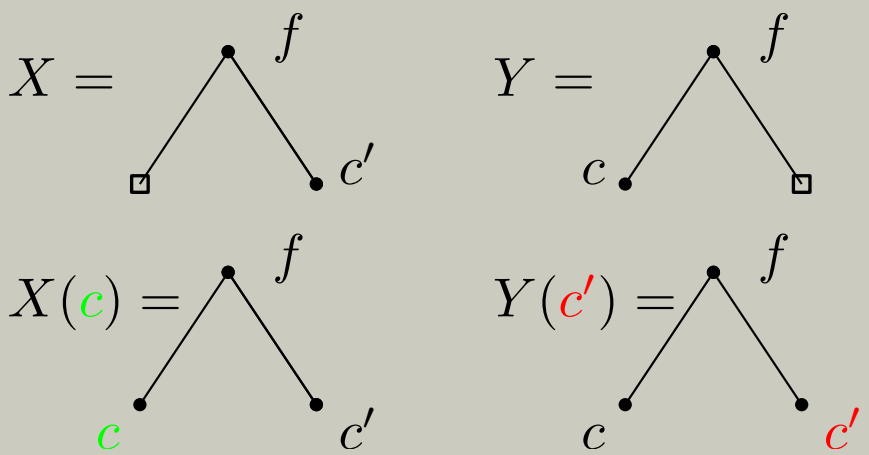
- signature (fixed arities) $[f, a, c]$
- context variables denoting terms with one 'hole' $[X()]$
- variables denoting closed terms $[x]$
- equations built with them

- f: arity 2, c, c': arity 0
- $X(c) = Y(c')$

More formally

- signature (fixed arities) $[f, a, c]$
- context variables denoting terms with one 'hole' $[X()]$
- variables denoting closed terms $[x]$
- equations built with them

- f: arity 2, c, c': arity 0
- $X(c) = Y(c')$



Why

In between important problems:

- word equations , term unification
- second-order unification

Why

In between important problems:

- word equations in PSPACE, term unification in PTIME
- second-order unification undecidable

Why

In between important problems:

- word equations in PSPACE, term unification in PTIME
- second-order unification undecidable

- Unknown status
- The problem in between.

Why

In between important problems:

- word equations in PSPACE, term unification in PTIME
- second-order unification undecidable

- Unknown status
- The problem in between.

Other connections

- one-step term rewriting
- natural language processing
- linear second-order unification

This talk

Context unification is in PSPACE.
(trivially NP-hard).

This talk

Context unification is in PSPACE.
(trivially NP-hard).

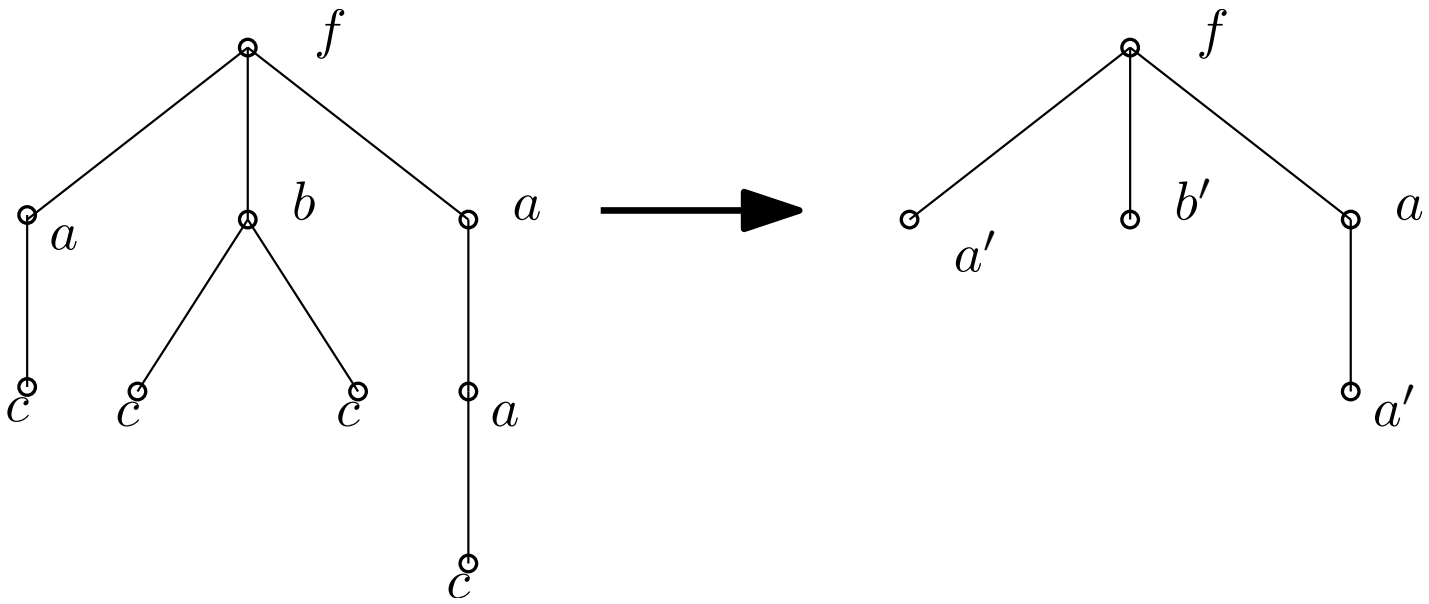
- generalises solution for word equations
- apply compression to both sides
- preserve equality

Leaf compression

- 'Absorb' each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k)$ with $f'(t_2, t_3, \ldots, t_k)$

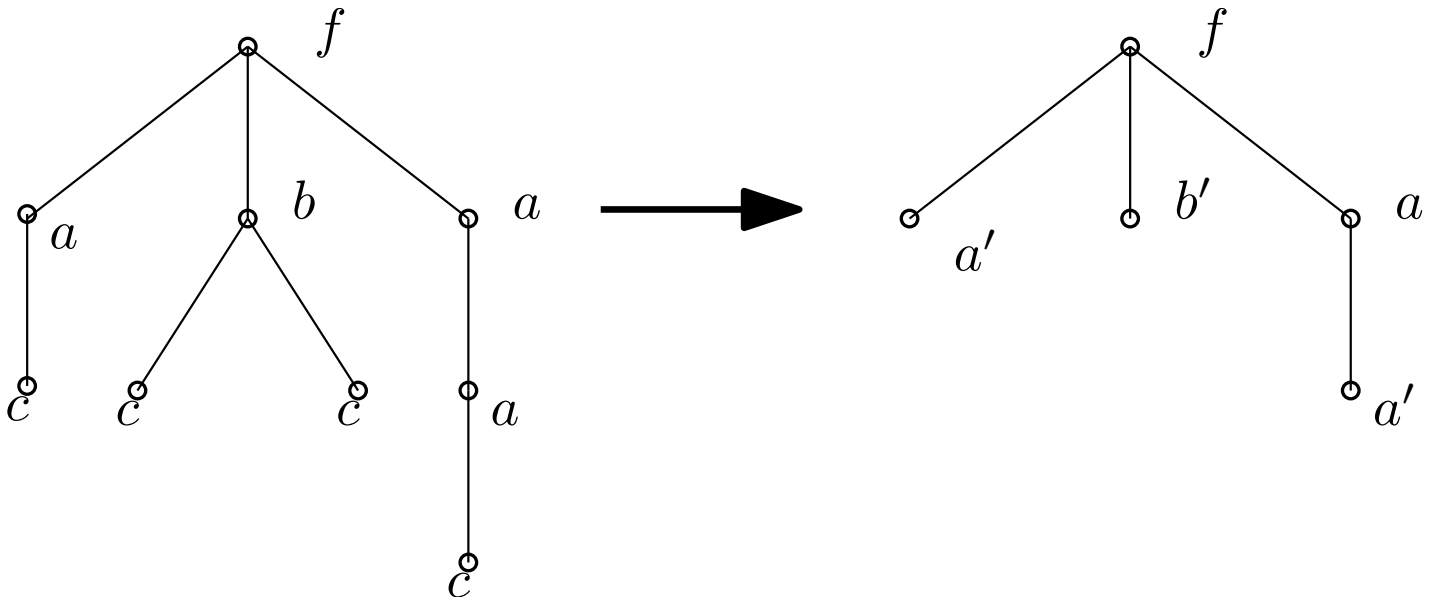
Leaf compression

- ‘Absorb’ each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k)$ with $f'(t_2, t_3, \ldots, t_k)$



Leaf compression

- 'Absorb' each leaf by its father (and change the labels).
- Replace $f(c_1, t_2, t_3, \ldots, c_2, \ldots, t_k)$ with $f'(t_2, t_3, \ldots, t_k)$



- half of nodes are leaves
- $\mathcal{O}(\log N)$ rounds

Chains compression

- Long chains are not really affected

Chains compression

- Long chains are not really affected
- those are almost strings
- for strings we know what to do

Chains compression

- Long chains are not really affected
- those are almost strings
- for strings we know what to do
  - 2-chain compression: replace ab with a'
  - a-chain compression: replace a^k with a_k

Chains compression

- Long chains are not really affected
- those are almost strings
- for strings we know what to do
  - 2-chain compression: replace ab with a'
  - a-chain compression: replace a^k with a_k

Lemma

1/3 of the nodes are compressed.

Algorithm

1: **while** $|T| > 1$ **do**

Algorithm

1: **while** $|T| > 1$ **do**
2: $L \leftarrow$ list of unary letters in T
3: **for** each $a \in L$ **do** ▷ a-chain compression
4: compress maximal chains of a

Algorithm

1: **while** $|T| > 1$ **do**
2: $L \leftarrow$ list of unary letters in T
3: **for** each $a \in L$ **do** ▷ a-chain compression
4: compress maximal chains of a
5: $P \leftarrow$ list of 2-chains
6: **for** $ab \in P$ **do**
7: compress 2-chain ab ▷ 2-chains compression

Algorithm

1: **while** $|T| > 1$ **do**
2: $L \leftarrow$ list of unary letters in T
3: **for** each $a \in L$ **do** ▷ a-chain compression
4: compress maximal chains of a
5: $P \leftarrow$ list of 2-chains
6: **for** $ab \in P$ **do**
7: compress 2-chain ab ▷ 2-chains compression
8: $L_0 \leftarrow$ list of constants, $L_{\geq 1} \leftarrow$ list of other letters in T
9: **for** all $f \in L_{\geq 1}$ and all $c \in L_0$ **do**
10: perform leaf compression (in parallel)

Size analysis

Lemma

In one phase the size drops by a constant factor.

Size analysis

Lemma

In one phase the size drops by a constant factor.

no chain	we remove all leaves, size halves
single chain	a string, size drops by a constant factor
general	some mix of above

Apply compression on both sides

Apply compression on both sides

This preserves equality

Apply compression on both sides

This preserves equality

But how?

Easy case (2-chain compression)

- we replace every explicit ab
- every ab in $S(X)$ ($S(x)$) is replaced implicitly

Easy case (2-chain compression)

- we replace every explicit ab
- every ab in $S(X)$ ($S(x)$) is replaced implicitly

Definition (2-chain types)

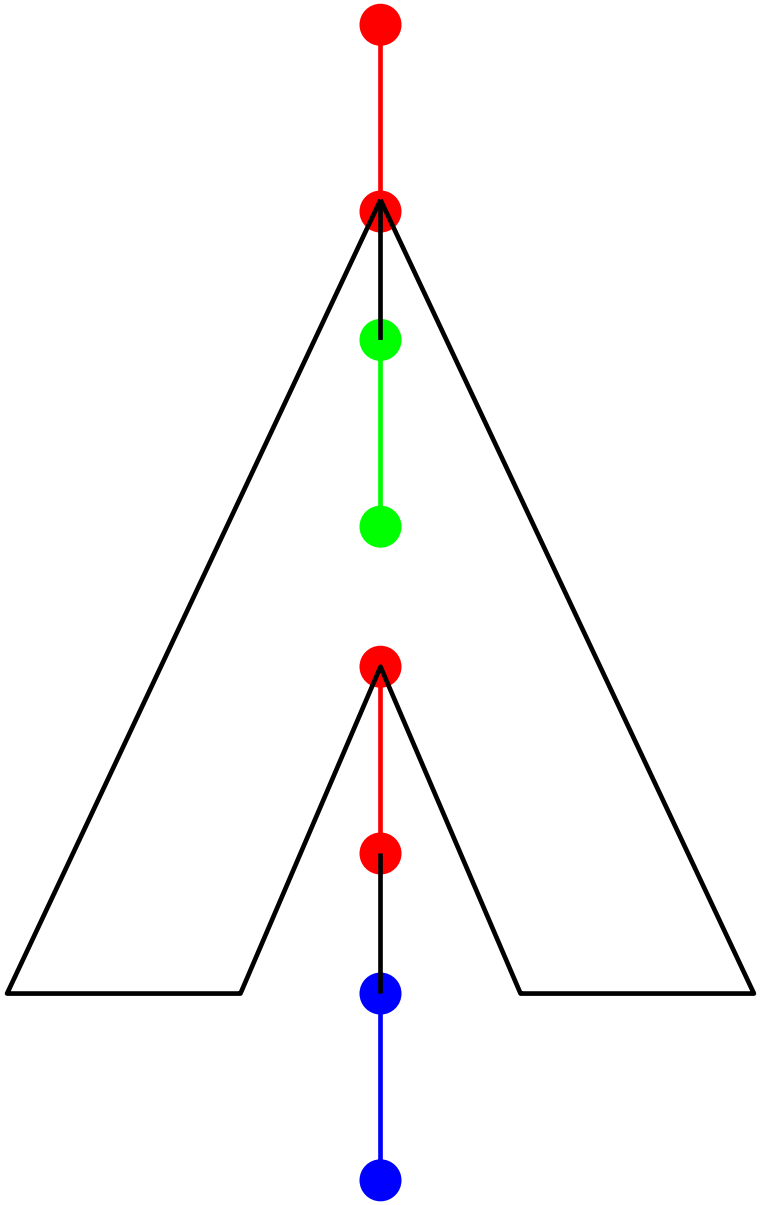
For a solution S the occurrence of ab is

explicit it comes from the equation;

implicit comes solely from $S(X)$ (or $S(x)$);

crossing in other case.

ab is crossing (for S) if it has a crossing occurrence (for S), non-crossing (for S) otherwise.



Non-crossing 2chain Compression

2ChainNCrComp

1: let $c \in \Sigma$ be an unused letter
2: replace each explicit ab by c

Non-crossing 2chain Compression

2ChainNCrComp

1: let $c \in \Sigma$ be an unused letter
2: replace each explicit ab by c

Lemma

2ChainNCrComp(a, b) *properly compresses noncrossing 2chains.*

Non-crossing 2chain Compression

2ChainNCrComp

1: let $c \in \Sigma$ be an unused letter
2: replace each explicit ab by c

Lemma

2ChainNCrComp(a, b) *properly compresses noncrossing 2chains.*

Proof.

Every ab is replaced:

explicit pairs replaced explicitly

implicit pairs replaced implicitly (in the solution)

crossing there are none □

a-chains

Definition (*a*-chain types)

For a solution S the occurrence of a maximal a-chain is

- explicit: it comes from the equation;
- implicit: comes solely from $S(X)$ (or $S(x)$);
- crossing: in other case.

a has crossing chains (for S) if it has a crossing a-chain occurrence (for S), non-crossing (for S) otherwise.

a-chains

Definition (a-chain types)

For a solution S the occurrence of a maximal a-chain is

- explicit: it comes from the equation;
- implicit: comes solely from $S(X)$ (or $S(x)$);
- crossing: in other case.

a has crossing chains (for S) if it has a crossing a-chain occurrence (for S), non-crossing (for S) otherwise.

MaxChainNCr

1: **for** $\ell > 1$ **do**
2: replace each explicit maximal a^ℓ by a_ℓ

a-chains

Definition (*a*-chain types)

For a solution S the occurrence of a maximal a-chain is

- **explicit** it comes from the equation;
- **implicit** comes solely from $S(X)$ (or $S(x)$);
- **crossing** in other case.

a has crossing chains (for S) if it has a crossing a-chain occurrence (for S), non-crossing (for S) otherwise.

MaxChainNCr

1: **for** $\ell > 1$ **do**
2: replace each explicit maximal a^ℓ by a_ℓ

Lemma

MaxChainNCr(a) *compresses noncrossing a-chains.*

Leaf compression

Definition (father-leaf pair)

For a solution S the occurrence of *father-leaf pair* (f, c) is

- **explicit** it comes from the equation;
- **implicit** comes solely from $S(X)$ (or $S(x)$);
- **crossing** in other case.

(f, c) is crossing (for S) if it has a crossing occurrence (for S), non-crossing (for S) otherwise.

Leaf compression

Definition (father-leaf pair)

For a solution S the occurrence of *father-leaf pair* (f, c) is

- **explicit** it comes from the equation;
- **implicit** comes solely from $S(X)$ (or $S(x)$);
- **crossing** in other case.

(f, c) is crossing (for S) if it has a crossing occurrence (for S), non-crossing (for S) otherwise.

LeafNCr

1: replace f with $c_1, \ldots, c_k$ at positions $(i_1, i_2, \ldots, i_k)$ by $f_{c_1,i_1,c_2,i_2,\ldots,c_k,i_k}$ ▷ No other leaves

Leaf compression

Definition (father-leaf pair)

For a solution S the occurrence of *father-leaf pair* (f, c) is

- explicit: it comes from the equation;
- implicit: comes solely from $S(X)$ (or $S(x)$);
- crossing: in other case.

(f, c) is crossing (for S) if it has a crossing occurrence (for S), non-crossing (for S) otherwise.

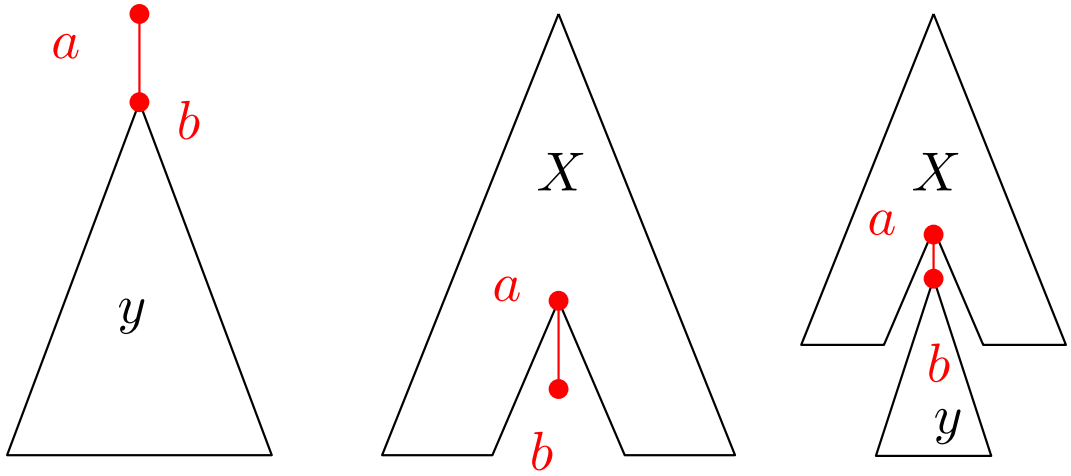
LeafNCr

1: replace f with $c_1, \ldots, c_k$ at positions $(i_1, i_2, \ldots, i_k)$ by $f_{c_1,i_1,c_2,i_2,\ldots,c_k,i_k}$ ▷ No other leaves

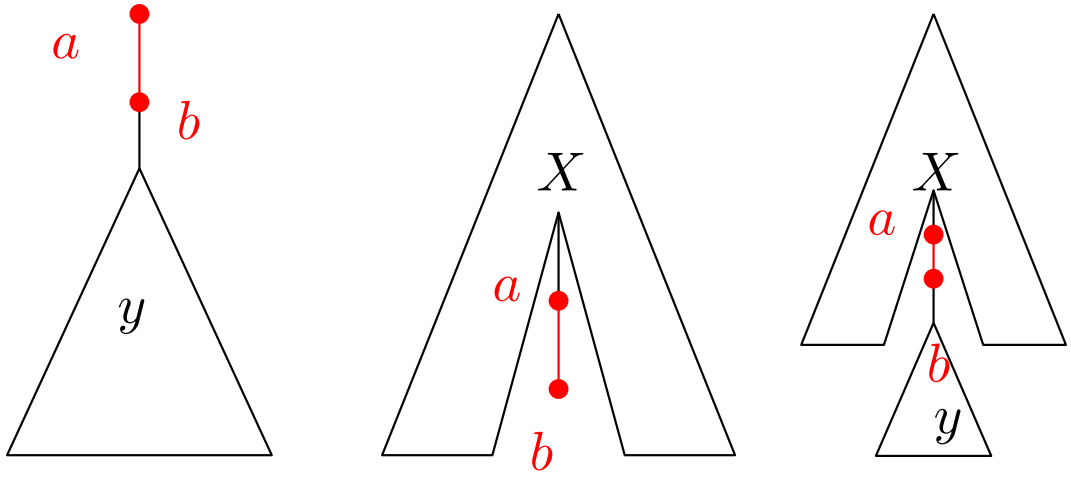
Lemma

LeafNCr *performs leaf compression when there is no crossing father-leaf pair* (f, c).

Uncrossing

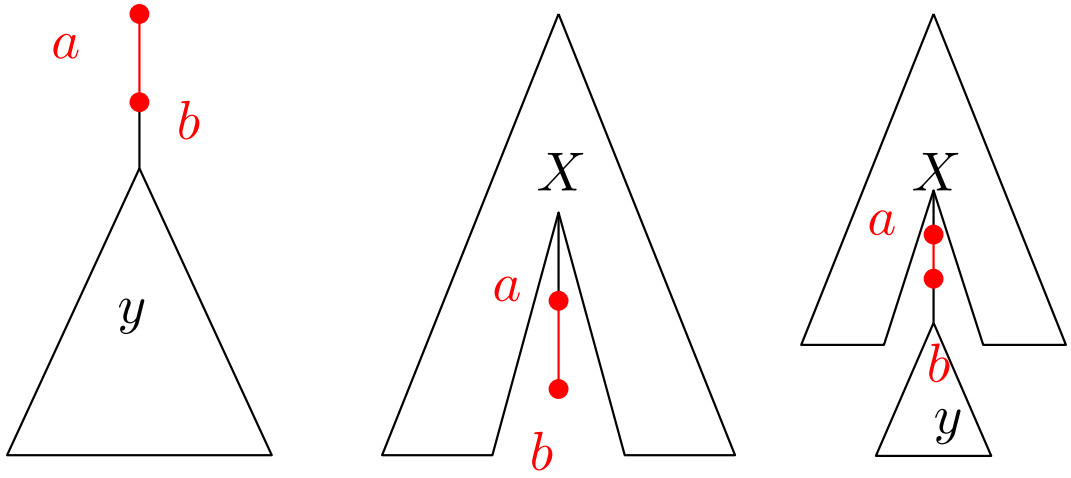


Uncrossing



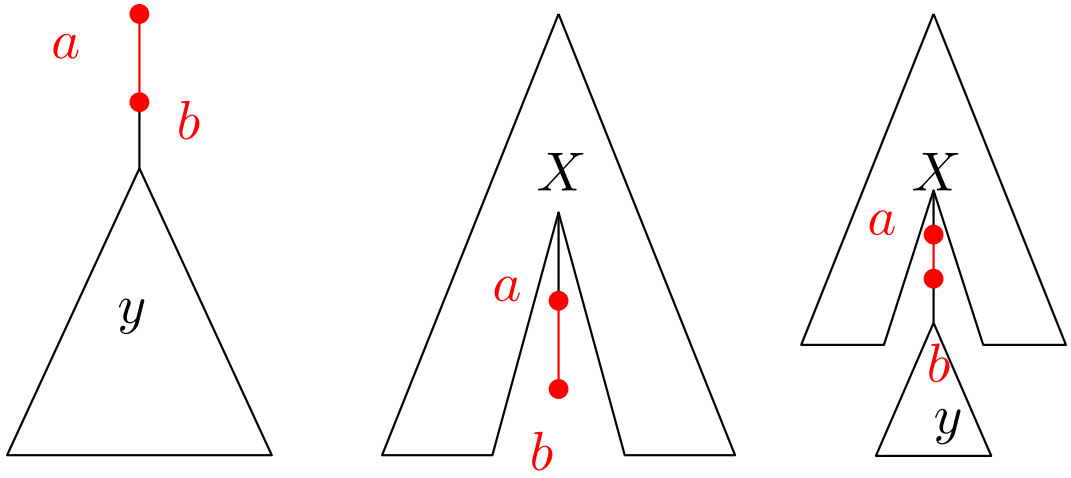
- replace Y with bY and replace X with Xa
  implicitly change $S(Y) = bt, S(X) = t'a$ to $S(Y) = t$, $S(X) = t'$

Uncrossing



- replace Y with bY and replace X with Xa
  implicitly change $S(Y) = bt, S(X) = t'a$ to $S(Y) = t$, $S(X) = t'$
- If $S(Y) = \Omega$ then remove Y.

Uncrossing

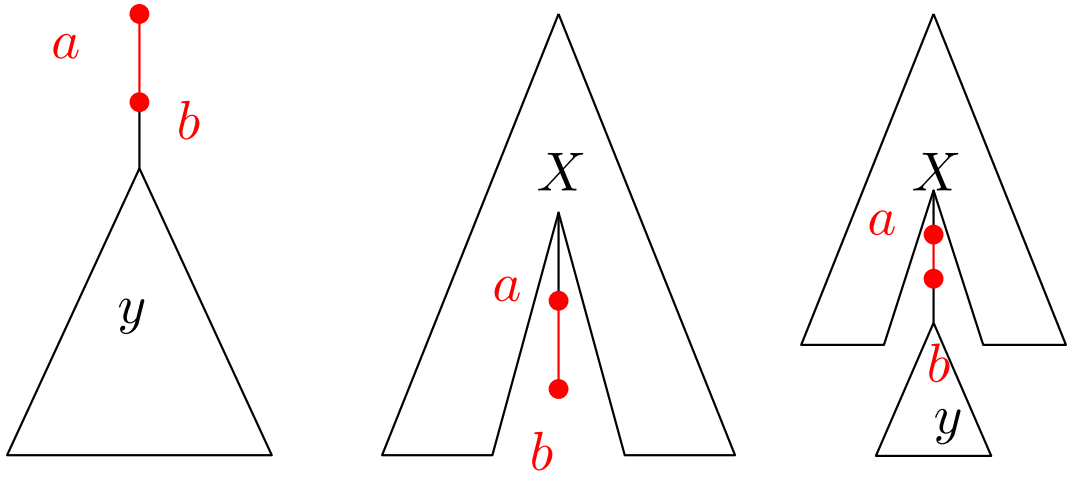


- replace Y with bY and replace X with Xa
  implicitly change $S(Y) = bt, S(X) = t'a$ to $S(Y) = t$, $S(X) = t'$
- If $S(Y) = \Omega$ then remove Y.

Lemma

After performing this for all variables, ab is no longer crossing.

Uncrossing



- replace Y with bY and replace X with Xa
  implicitly change $S(Y) = bt, S(X) = t'a$ to $S(Y) = t$, $S(X) = t'$
- If $S(Y) = \Omega$ then remove Y.

Lemma

After performing this for all variables, ab is no longer crossing.

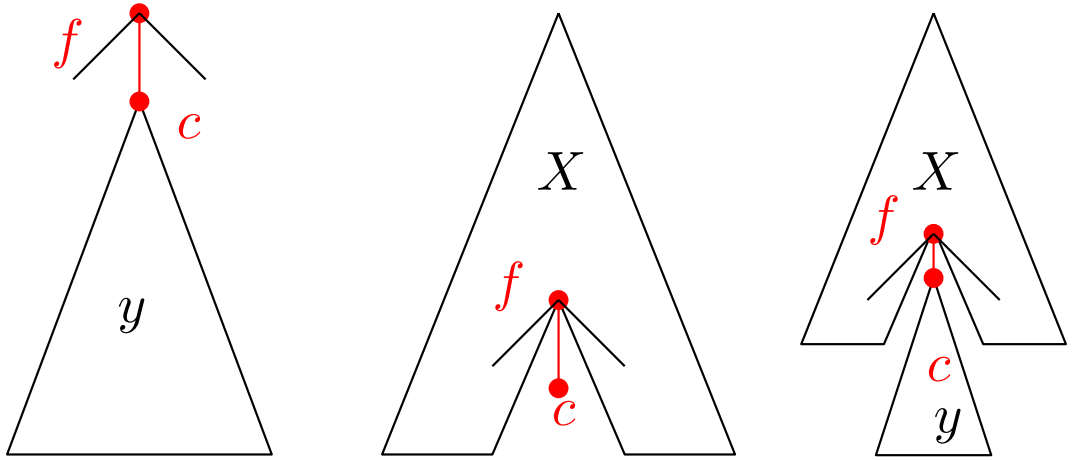
Compress it!

Crossing *a*-chains?

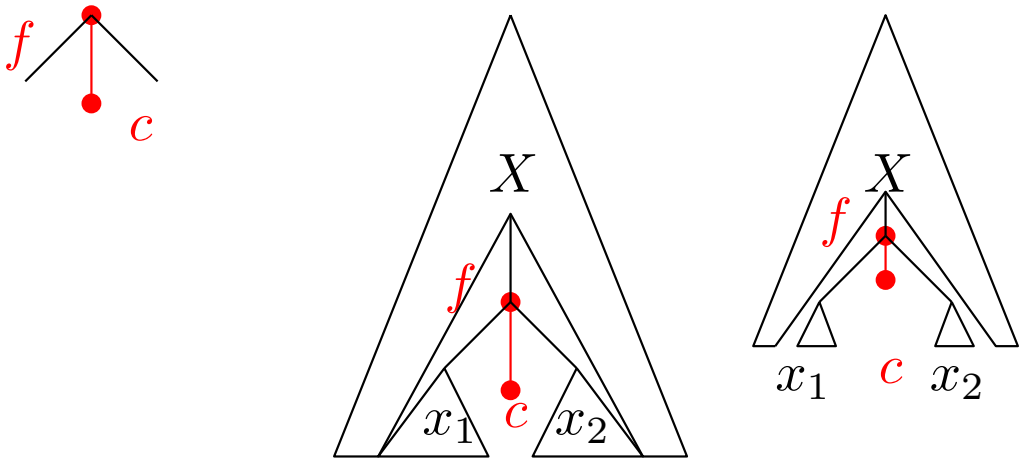
a is a crossing chain $\Leftrightarrow$ *aa* is a crossing 2-chain

- Crossing *a*-block: similar to crossing *ab* (equiv. to crossing *aa*).
- pop whole *a*-‘prefix’ and *a*-‘suffix’

Uncrossing father-leaf pair

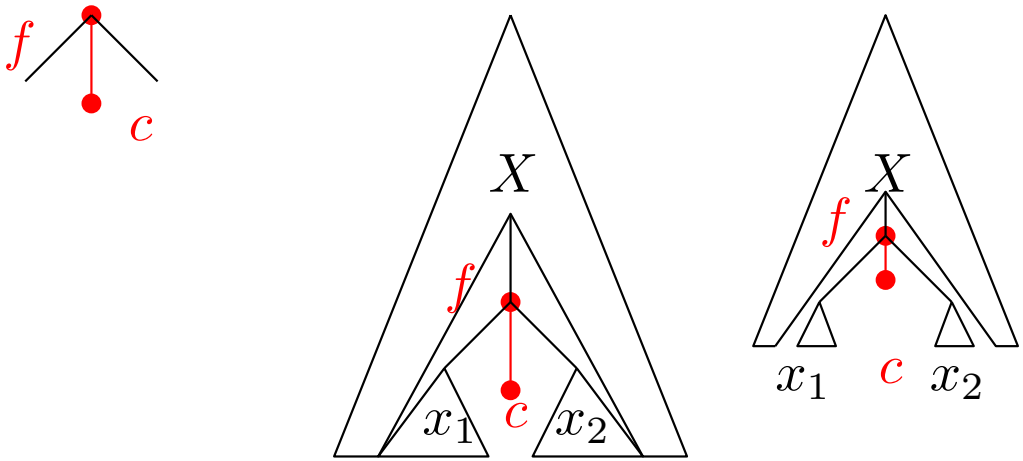


Uncrossing father-leaf pair



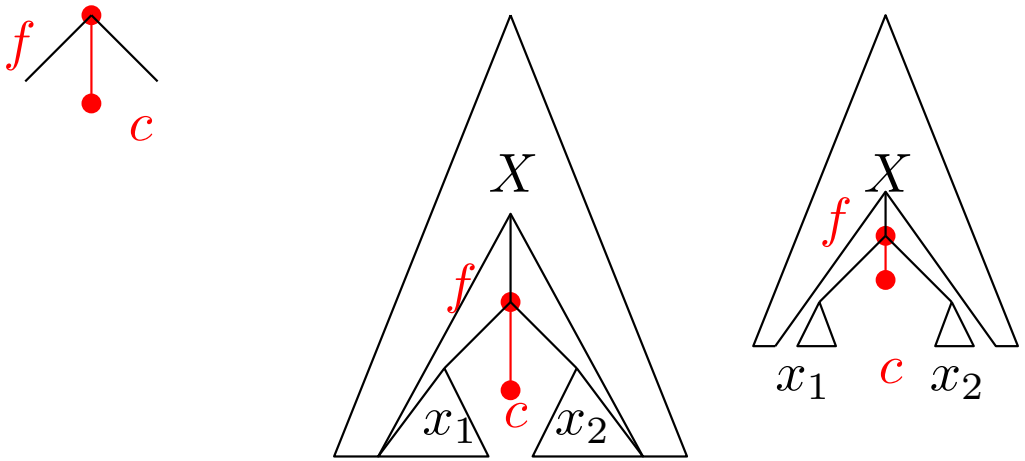
- replace y with c
- replace X with $X(f(x_1, \ldots x_{i-1}, \cdot, x_{i+1}, \ldots, x_\ell))$

Uncrossing father-leaf pair



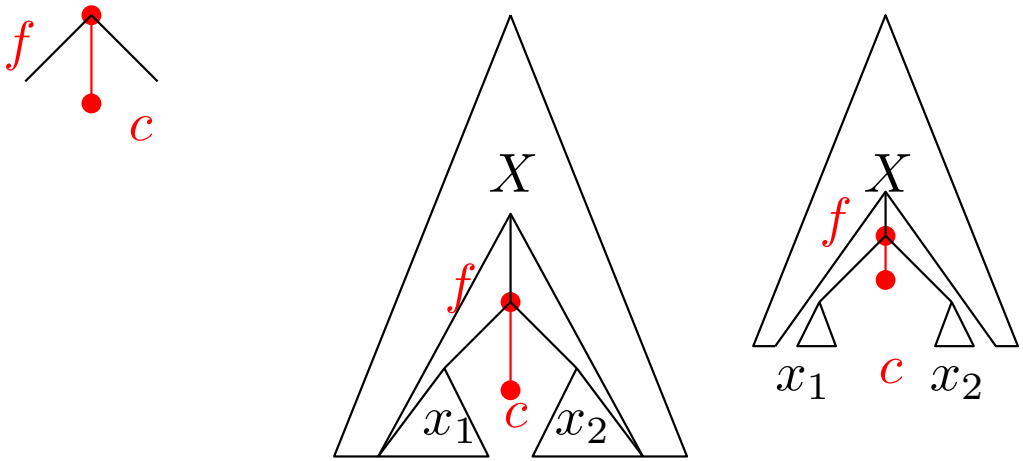
- replace y with c
- replace X with $X(f(x_1, \ldots x_{i-1}, \cdot, x_{i+1}, \ldots, x_\ell))$
  new variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell$

Uncrossing father-leaf pair



- replace y with c
- replace X with $X(f(x_1, \ldots x_{i-1}, \cdot, x_{i+1}, \ldots, x_\ell))$
  new variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell$
- remove X when $S(X)$ is empty

Uncrossing father-leaf pair



- replace y with c
- replace X with $X(f(x_1, \ldots x_{i-1}, \cdot, x_{i+1}, \ldots, x_\ell))$
  new variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_\ell$
- remove X when $S(X)$ is empty

- there are no crossing (f, c) father-leaf pairs
- compress them

1: **while** $|T| > 1$ **do**

1: **while** $|T| > 1$ **do**
2: $L \leftarrow$ list of unary letters in T
3: **for** noncrossing $a \in L$ **do**
4: compress maximal chains of a
5: **for** remaining $a \in L$ **do**
6: uncross and compress maximal chains of a

```
1: while |T| > 1 do
2:     L ← list of unary letters in T
3:     for noncrossing a ∈ L do
4:         compress maximal chains of a
5:     for remaining a ∈ L do
6:         uncross and compress maximal chains of a
7:     P ← list of 2-chains
8:     for noncrossing ab ∈ P do
9:         compress 2-chain ab
10:    for remaining ab ∈ P do
11:        uncross and compress 2-chain ab
```

```
1: while |T| > 1 do
2:     L ← list of unary letters in T
3:     for noncrossing a ∈ L do
4:         compress maximal chains of a
5:     for remaining a ∈ L do
6:         uncross and compress maximal chains of a
7:     P ← list of 2-chains
8:     for noncrossing ab ∈ P do
9:         compress 2-chain ab
10:    for remaining ab ∈ P do
11:        uncross and compress 2-chain ab
12:    L_0 ← list of constants, L_≥1 ← list of other letters in T
13:    for noncrossing f ∈ L_≥1, c_1, ..., c_k ∈ L_0 do
14:        leaf compressions for f, c_1, ..., c_k
15:    for remaining f ∈ L_≥1 and c_1, ..., c_k ∈ L_0 do
16:        uncross and leaf compress f, c_1, ..., c_k
```

Controlling the new variables

Lemma

There are at most

- *n context variables*
- *kn variables*

Controlling the new variables

Lemma

There are at most

- *n context variables*
- *kn variables*

Proof.

- we do not introduce new context variables
- we can associate each ‘new’ variable with context variable at most $(k - 1)$ are associated with one context variable □

Equation size

- there are kn variables and n context variables
- each uncrossing: k letters per variable (so $\mathcal{O}(k^2 n)$)
- there are $\mathcal{O}(kn)$ uncrossings
- $\mathcal{O}(k^3 n^2)$ new letters

Equation size

- there are kn variables and n context variables
- each uncrossing: k letters per variable (so $\mathcal{O}(k^2 n)$)
- there are $\mathcal{O}(kn)$ uncrossings
- $\mathcal{O}(k^3 n^2)$ new letters

But size of the tree (also: equation) drops by a constant!

$$|u' = v'| \leq \tfrac{3}{4}|u = v| + \mathcal{O}(k^3 n^2)$$

Equation size

- there are kn variables and n context variables
- each uncrossing: k letters per variable (so $\mathcal{O}(k^2 n)$)
- there are $\mathcal{O}(kn)$ uncrossings
- $\mathcal{O}(k^3 n^2)$ new letters

But size of the tree (also: equation) drops by a constant!

$$|u' = v'| \leq \tfrac{3}{4}|u = v| + \mathcal{O}(k^3 n^2)$$
$$\text{so } |u = v| = \mathcal{O}(k^3 n^2)$$