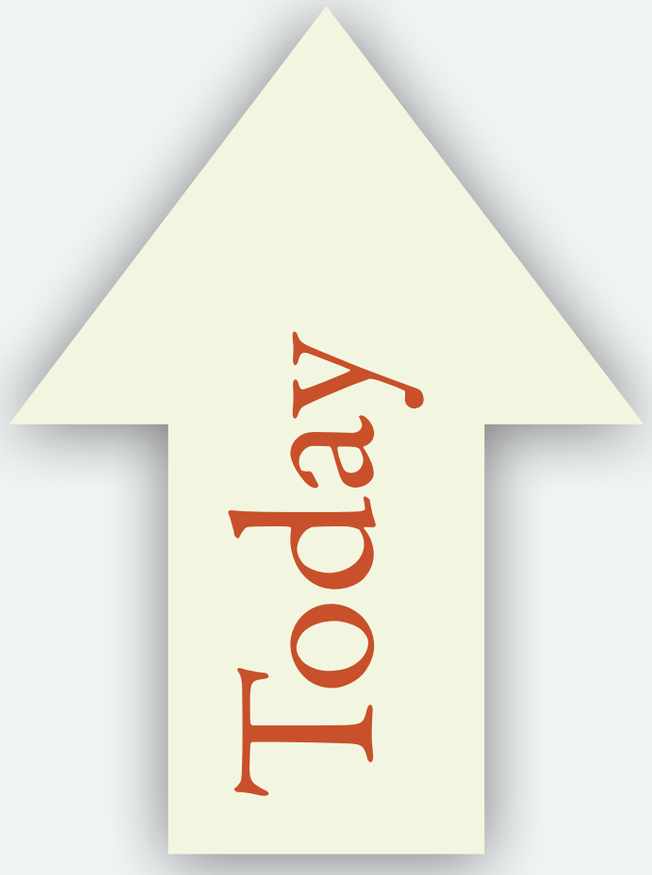
Logics on words and trees with data

Diego Figueira & Ranko Lazić

ESSLLI 2016 - Bolzano

Logics on words and trees with data



Diego Figueira & Ranko Lazić

ESSLLI 2016 - Bolzano

A final comment on $FO^2(<,\sim)$ ...

A final comment on $FO^2(<,\sim)$ ...

$$\text{SAT-}FO^2(<,\sim) \in \text{NExpTime}$$

SAT-FO^2(<,~) ∈ NExpTime

Scott normal form

$$\forall x \forall y\ \chi(x,y) \wedge \bigwedge_i \forall x \exists y\ \chi_i(x,y)$$

where χ and χ_i are quantifier-free.

Note: it uses some new unary predicates.

a	b	a	b	a	a	a	b	b	a	b	a	b	a	b	a	a	b	a	b	a	b	b	a
5	1	3	5	2	1	4	2	5	1	1	4	1	3	1	5	5	4	1	1	5	4	1	4

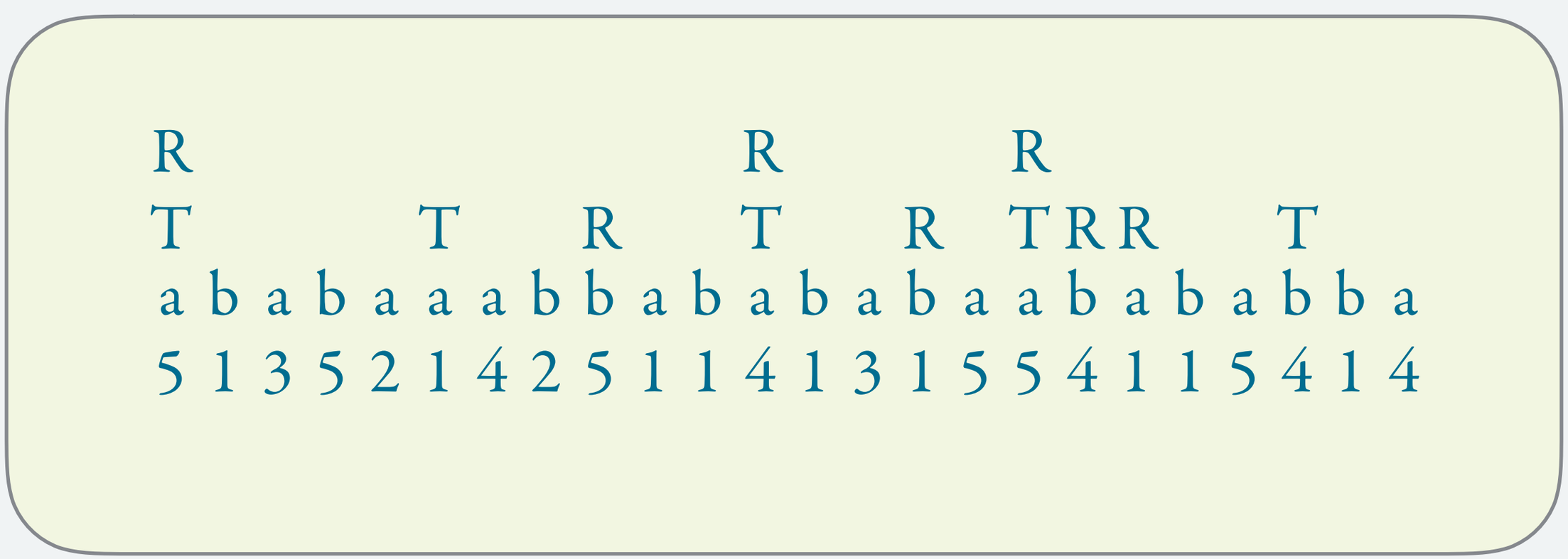
SAT-FO^2(<,~) ∈ NExpTime

Scott normal form

$$\forall x \forall y\ \chi(x,y) \wedge \bigwedge_i \forall x \exists y\ \chi_i(x,y)$$

where χ and χ_i are quantifier-free.

Note: it uses some new unary predicates.

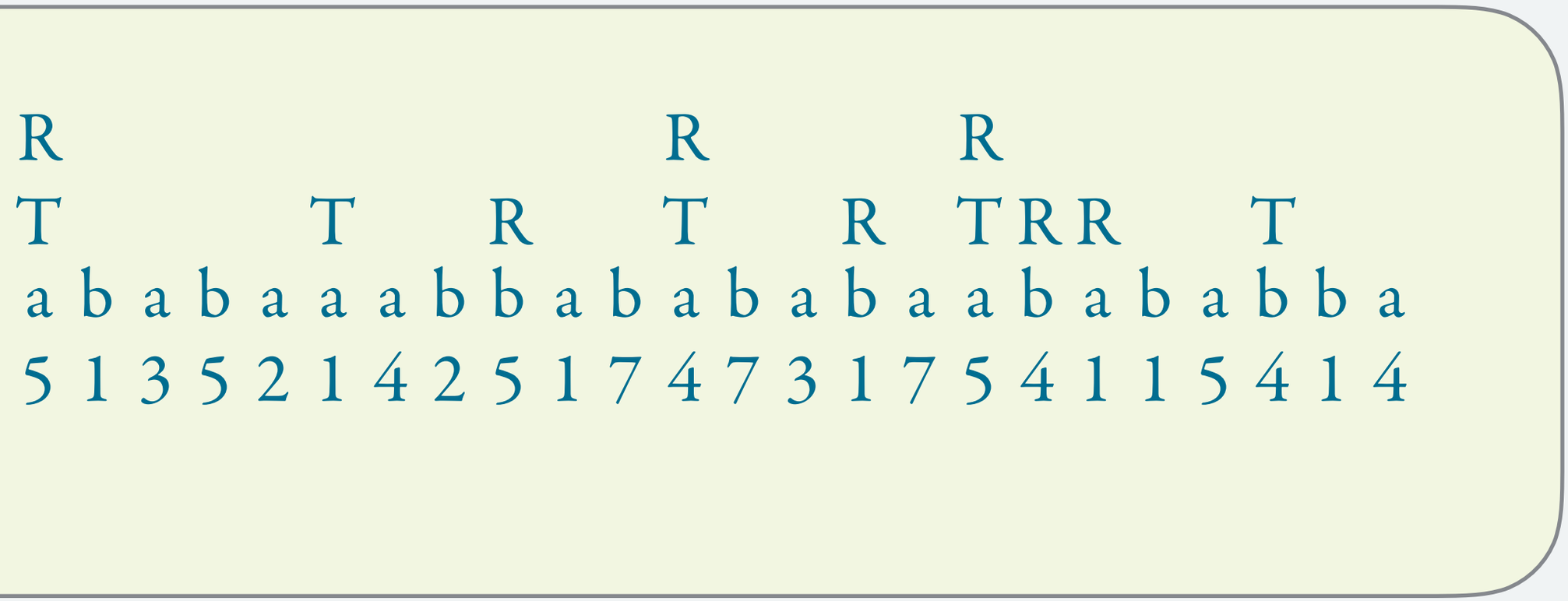


SAT-FO^2(<,~) ∈ NExpTime

Suppose $w \models \phi$.

Collect in D all data values d so that for some complete type τ (valuation for all unary predicates) d is equal to:

- the last data value appearing with type τ
- next to last value appearing with type τ
- first value appearing with type τ
- next to first value appearing with type τ



SAT-FO^2(<,~) ∈ NExpTime

Suppose $w \models \phi$.

Collect in D all data values d so that for some complete type τ (valuation for all unary predicates) d is equal to:

- the last data value appearing with type τ
- next to last value appearing with type τ
- first value appearing with type τ
- next to first value appearing with type τ

```
R                   R       R
T       T   R       T   R   T R R       T
a b a b a a a b b a b a b a b a a b a b a b b a
5 1 3 5 2 1 4 2 5 1 7 4 7 3 1 7 5 4 1 1 5 4 1 4
```

D= { 5, 4, 1}

SAT-FO^2(<,~) ∈ NExpTime

Let w' be w after removing all positions a data value not in D
We still have $w' \models \phi$.

We can code a data word with 2^n data values with a word with n new unary relations.
Polytime reduction:

SAT-FO^2(<,~) ⇝ SAT-FO^2(<) ∈ NExpTime

											R					R			R				
T				T			R			T			R		T	R	R			T		
a	b	a	b	a	a	a	b	b	a	b	a	b	a	b	a	a	b	a	b	a	b	b	a
5	1	3	5	2	1	4	2	5	1	7	4	7	3	1	7	5	4	1	1	5	4	1	4

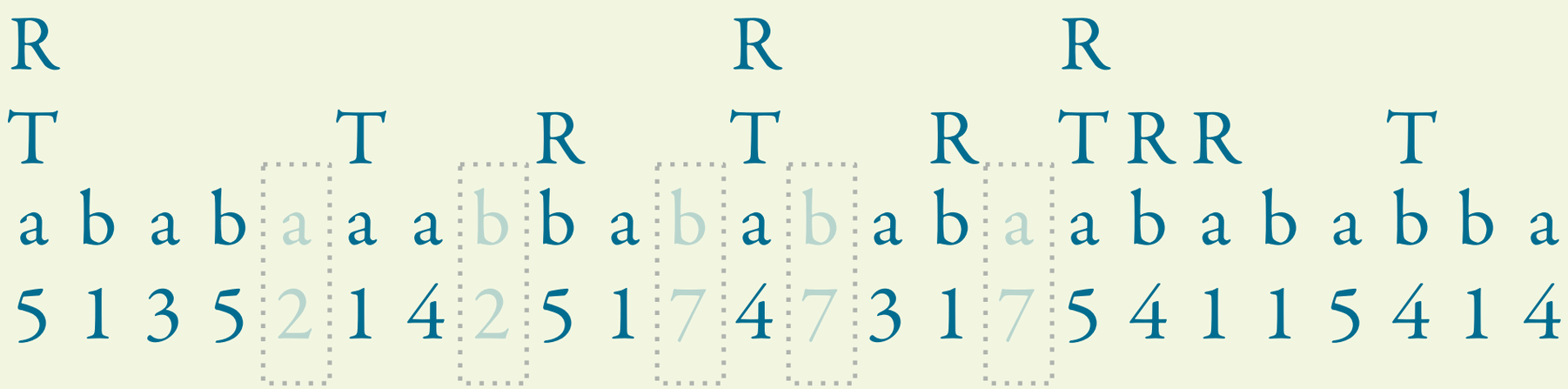
D= { 5, 4, 1}

SAT-FO^2(<,~) ∈ NExpTime

Let w' be w after removing all positions a data value not in D
We still have $w' \models \phi$.

We can code a data word with 2^n data values with a word with n new unary relations.
Polytime reduction:

SAT-FO^2(<,~) ⇝ SAT-FO^2(<) ∈ NExpTime



D= { 5, 4, 1}

Agenda

- Monday: Introduction, motivation
- Tuesday: Data words, first-order logic
- Wednesday: Data words, temporal logics
- Thursday: Data trees, path-based logics
- Friday: Data trees, other formalisms for data trees

data trees

...or XML documents.

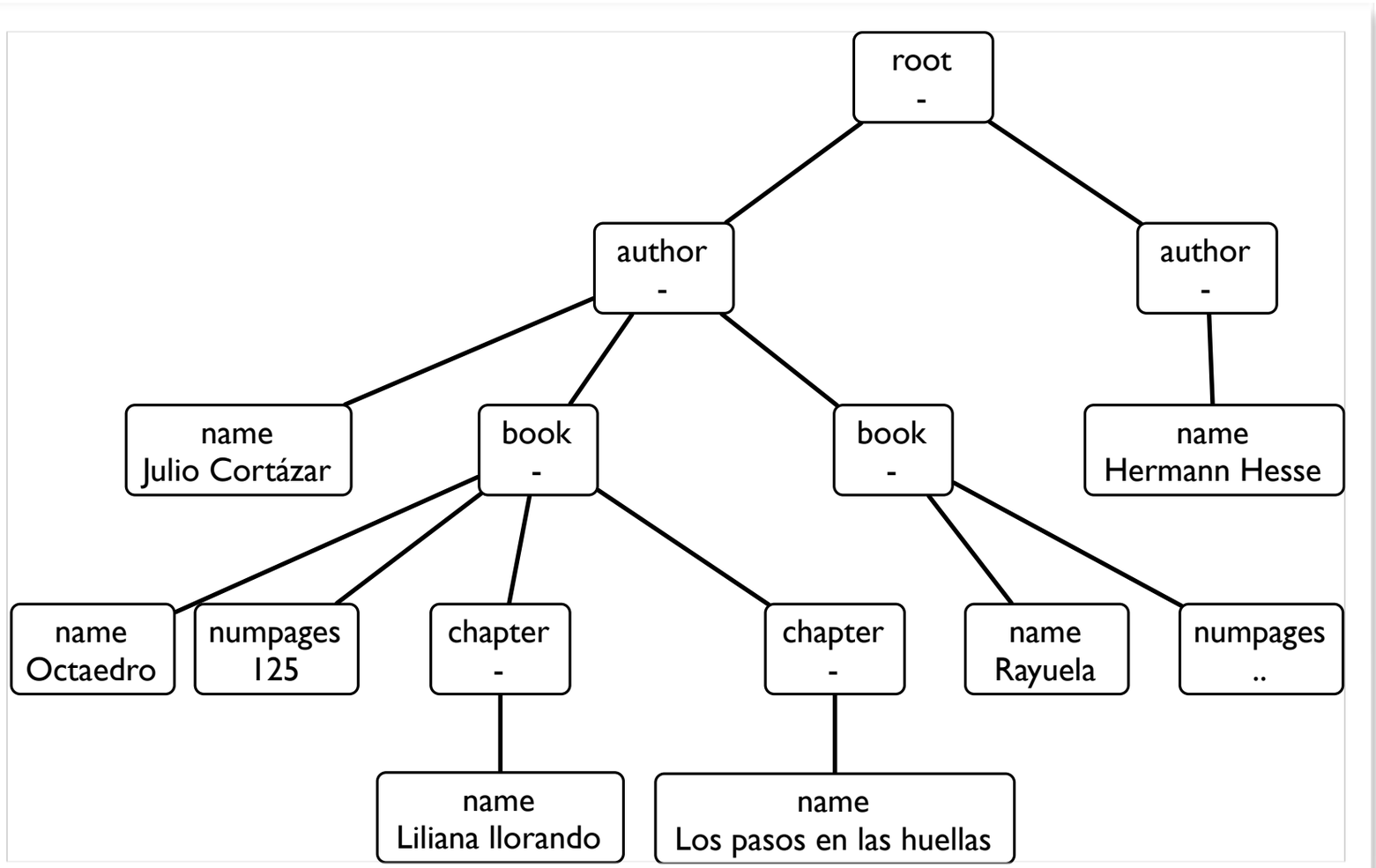
```
<author name = “Julio Cortázar”>
    <book name = “Octaedro” numpages = “125”>
        <chapter name = “Liliana llorando”/>
        <chapter name = “Los pasos en las huellas”/>
    </book>
    <book name= “Rayuela” numpages = “...”>
        ...
    </book>
</author>
<author name = “Hermann Hesse”>
    ...
</author>
```

XML

```
<author name = “Julio Cortázar”>
    <book name = “Octaedro” numpages = “125”>
        <chapter name = “Liliana llorando”/>
        <chapter name = “Los pasos en las huellas”/>
    </book>
    <book name= “Rayuela” numpages = “...”>
        ...
    </book>
</author>
<author name = “Hermann Hesse”>
    ...
</author>
```

XML

data tree



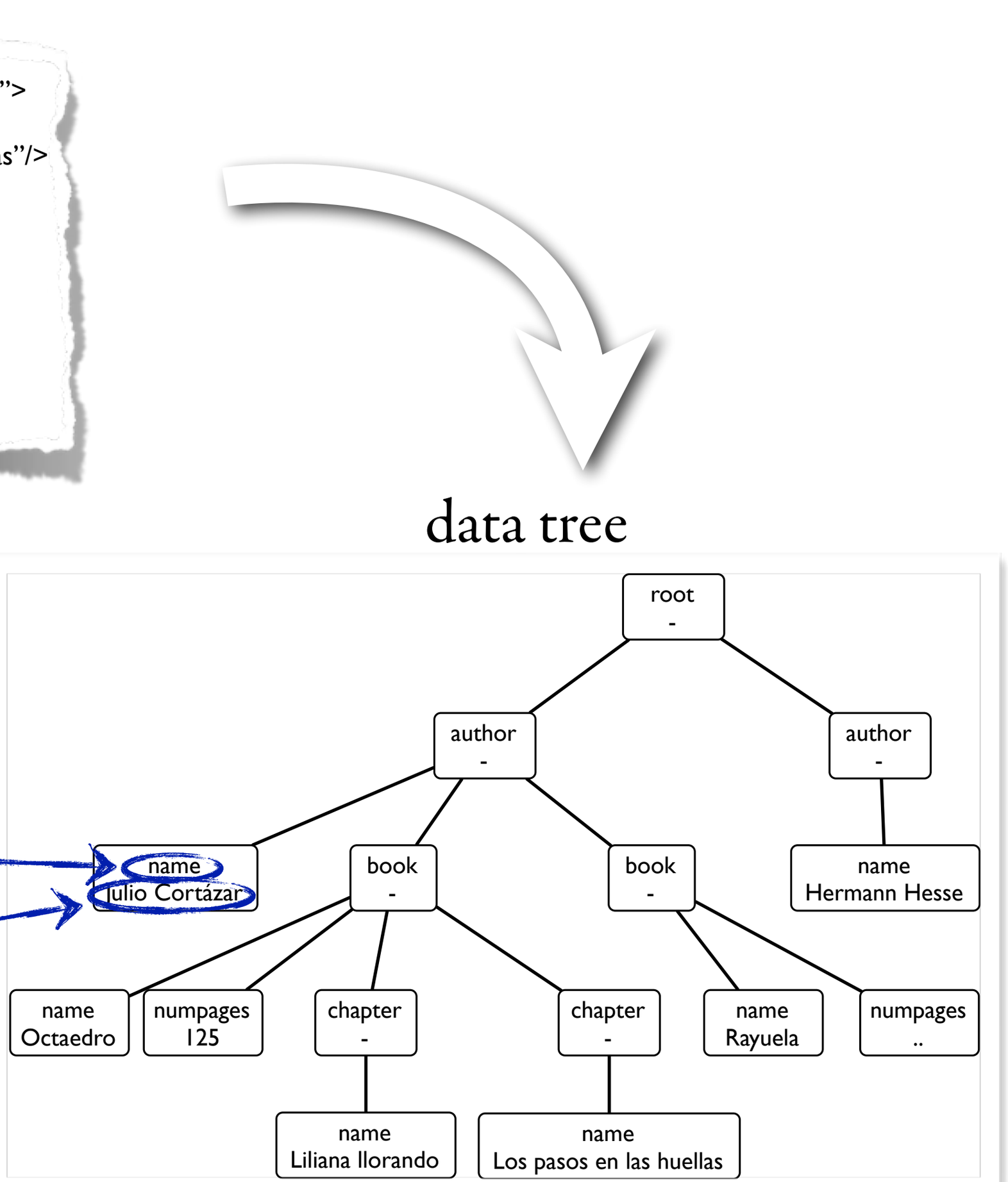
```
<author name = “Julio Cortázar”>
    <book name = “Octaedro” numpages = “125”>
        <chapter name = “Liliana llorando”/>
        <chapter name = “Los pasos en las huellas”/>
    </book>
    <book name= “Rayuela” numpages = “...”>
        ...
    </book>
</author>
<author name = “Hermann Hesse”>
    ...
</author>
```

XML

data tree

data tree

A finite, unranked tree over a finite alphabet & an infinite domain

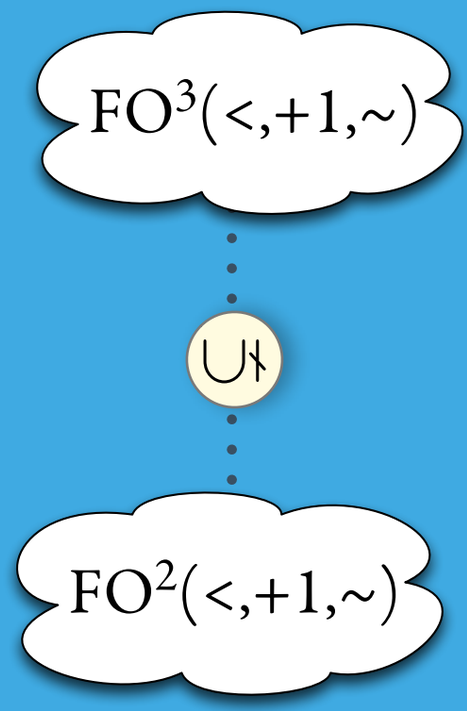


Formalisms for data trees

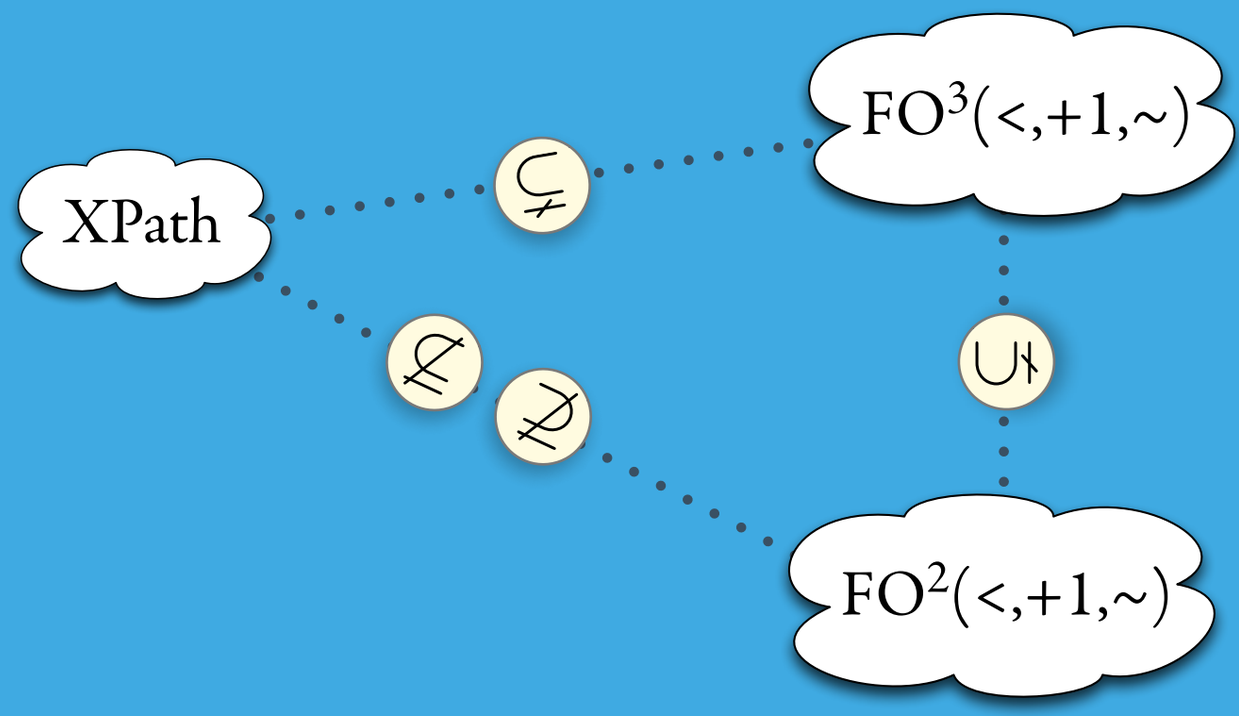
Formalisms for data trees

$FO^3(<,+1,\sim)$

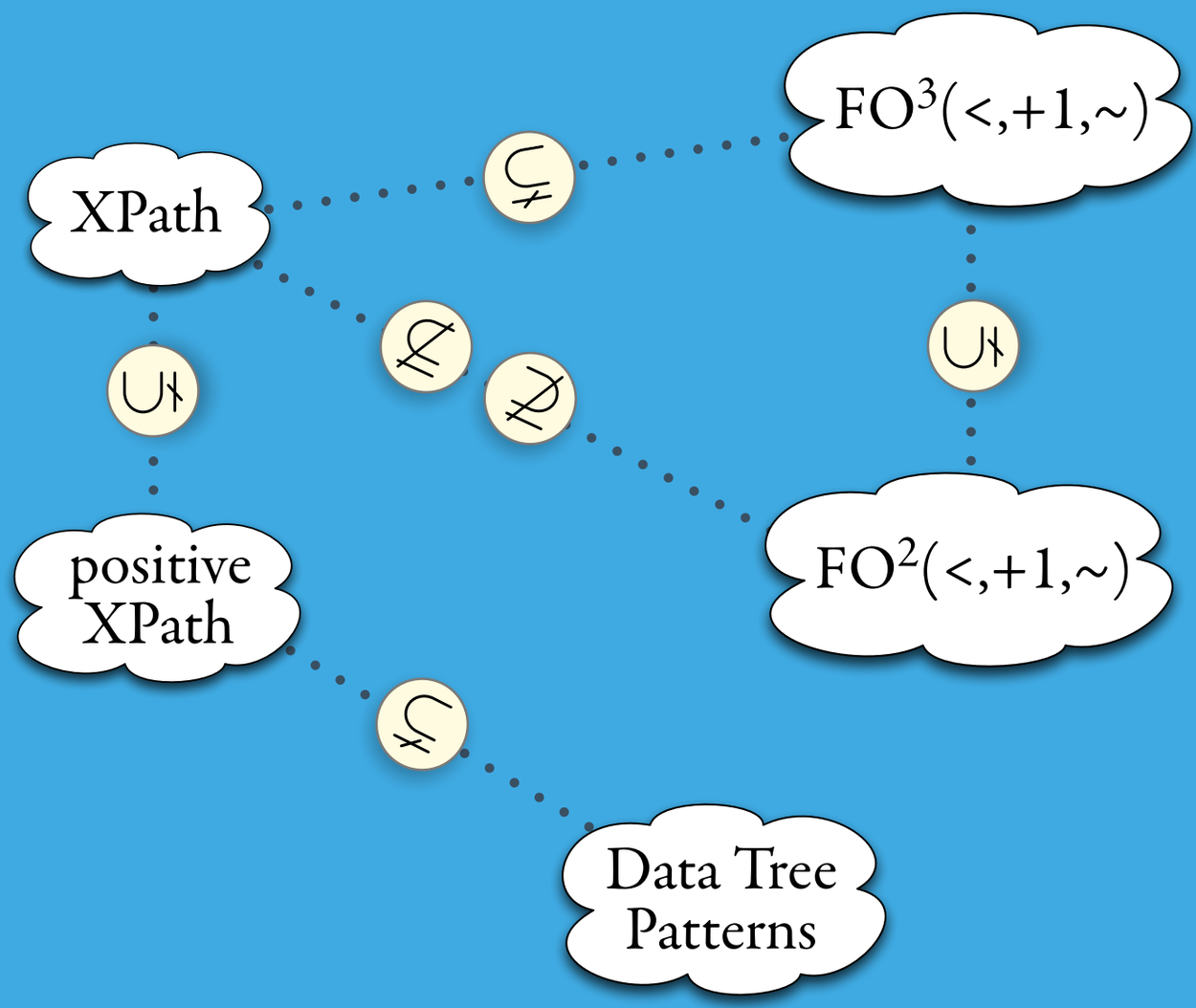
Formalisms for data trees



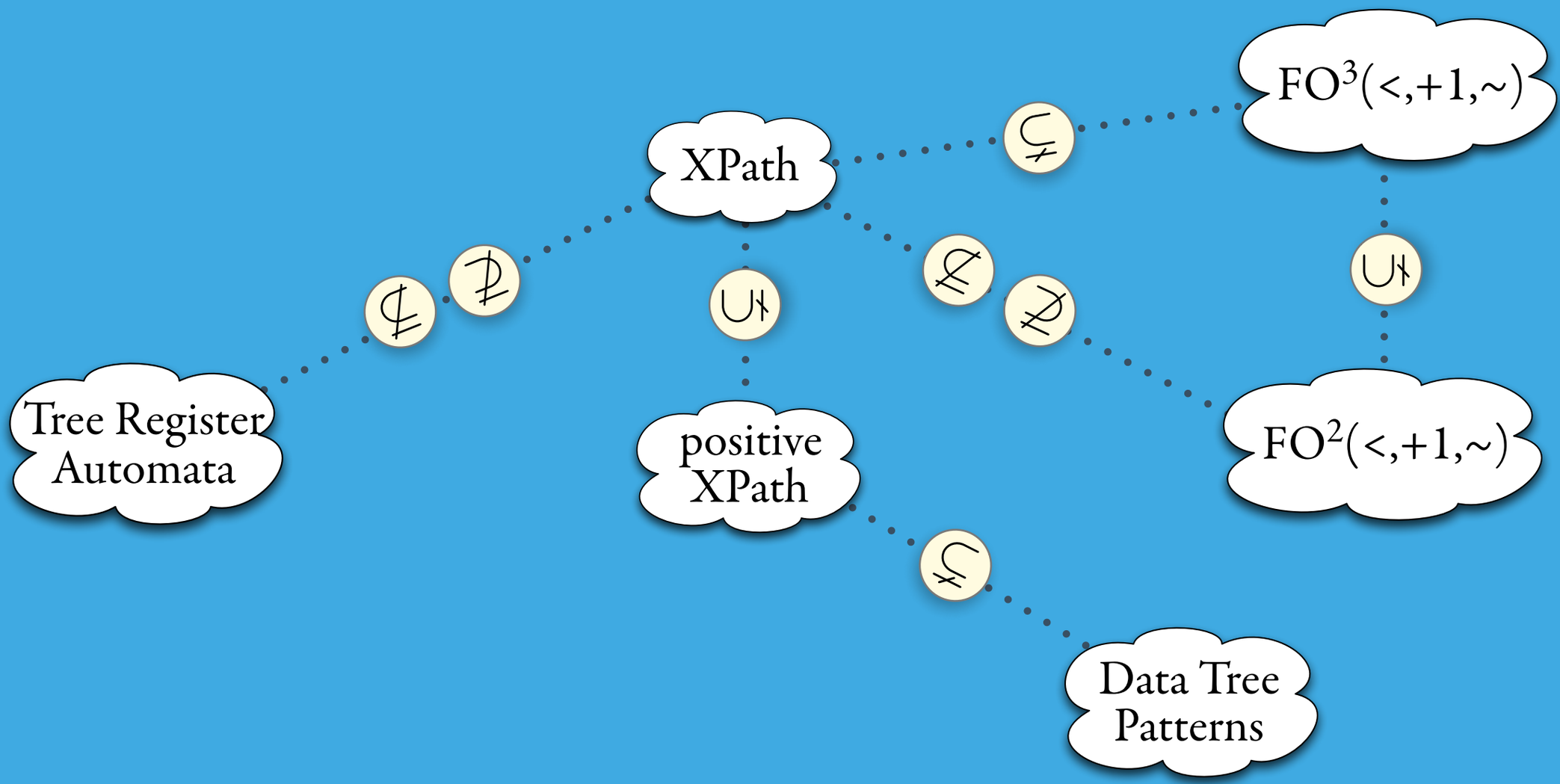
Formalisms for data trees



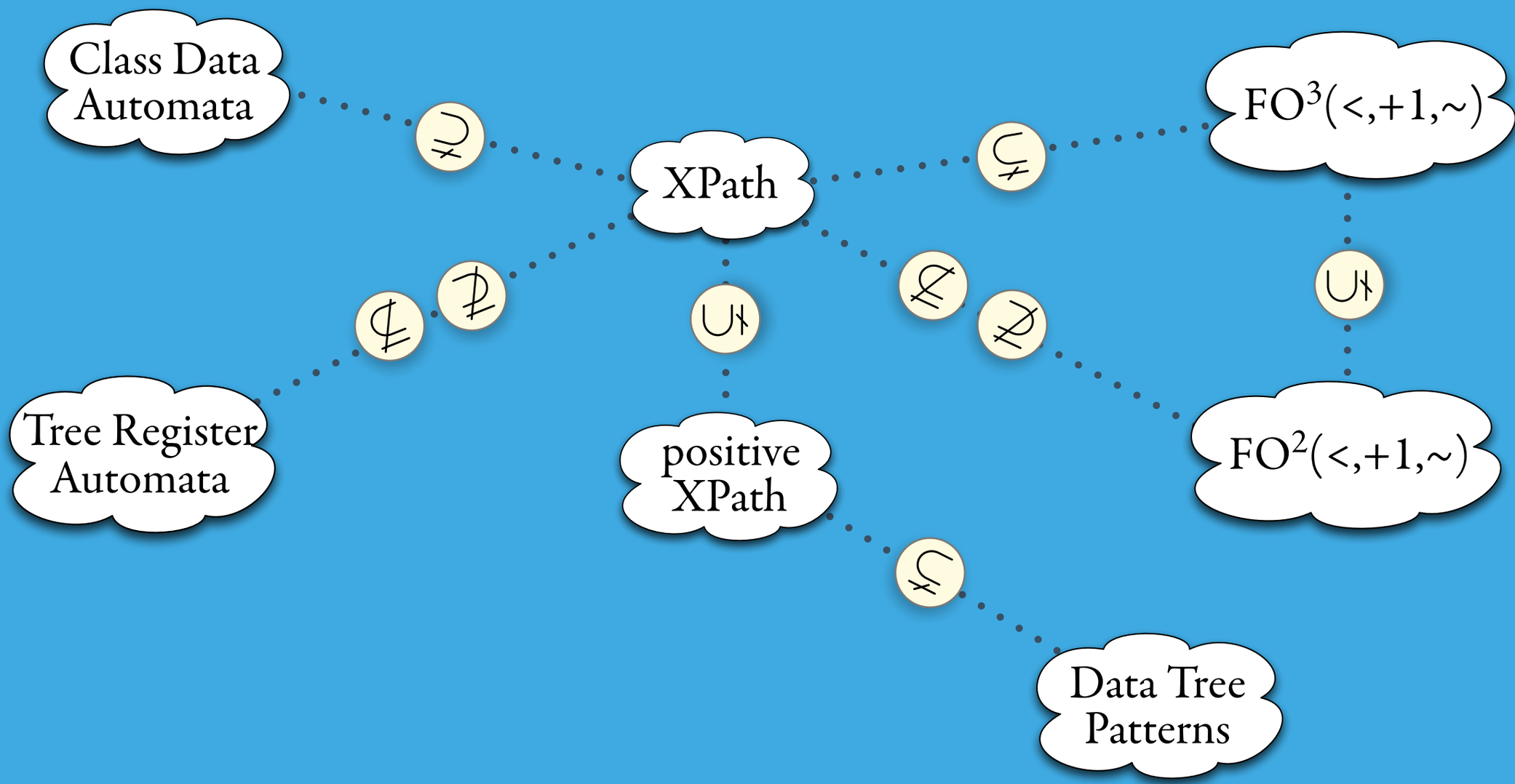
Formalisms for data trees



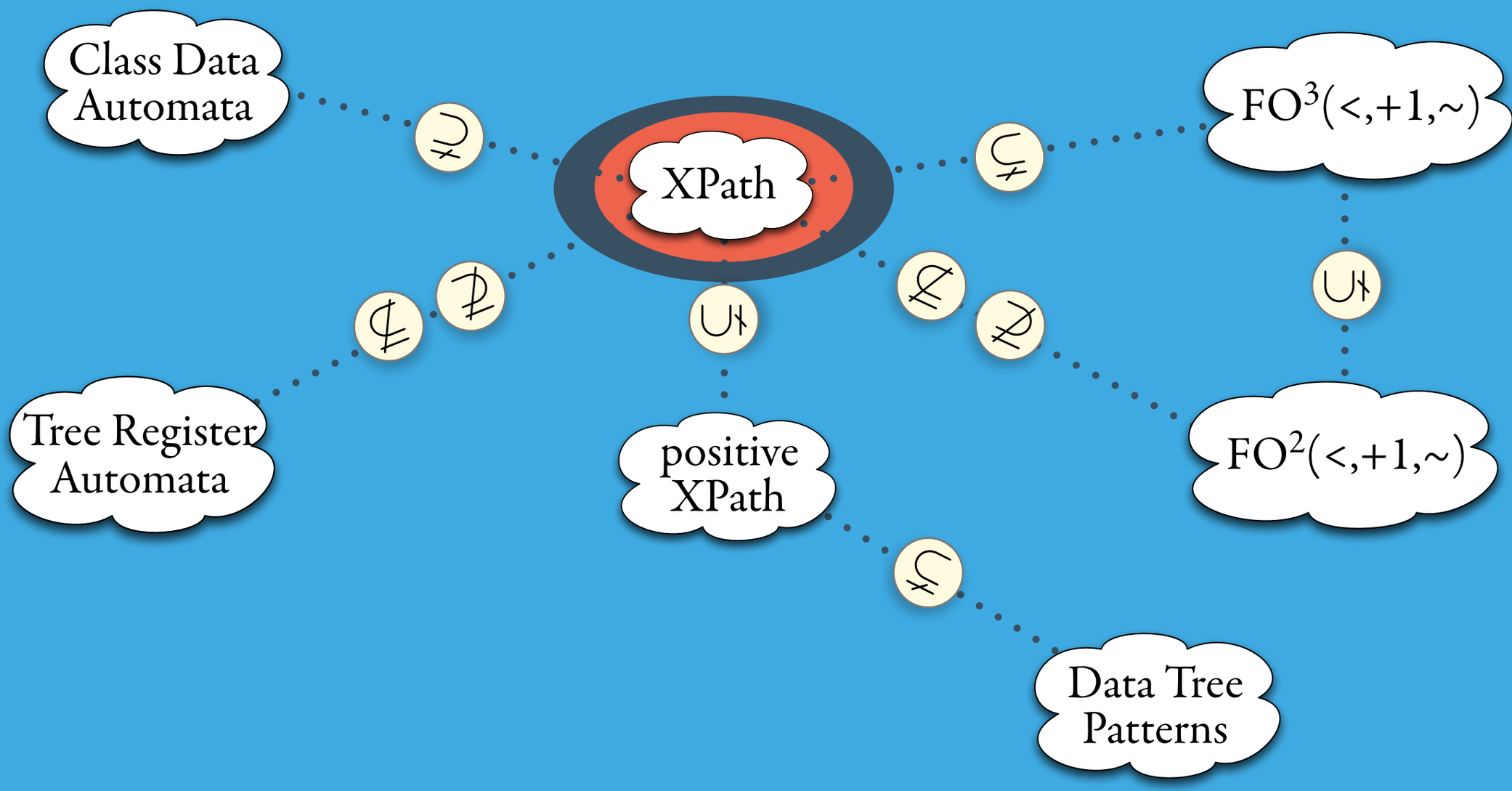
Formalisms for data trees



Formalisms for data trees

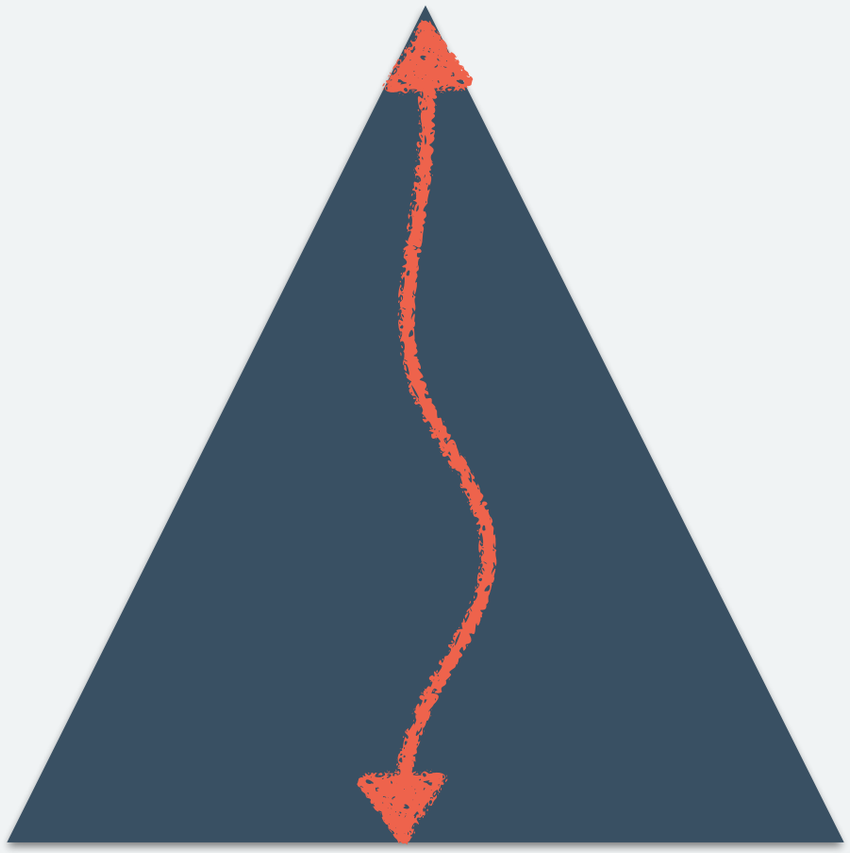


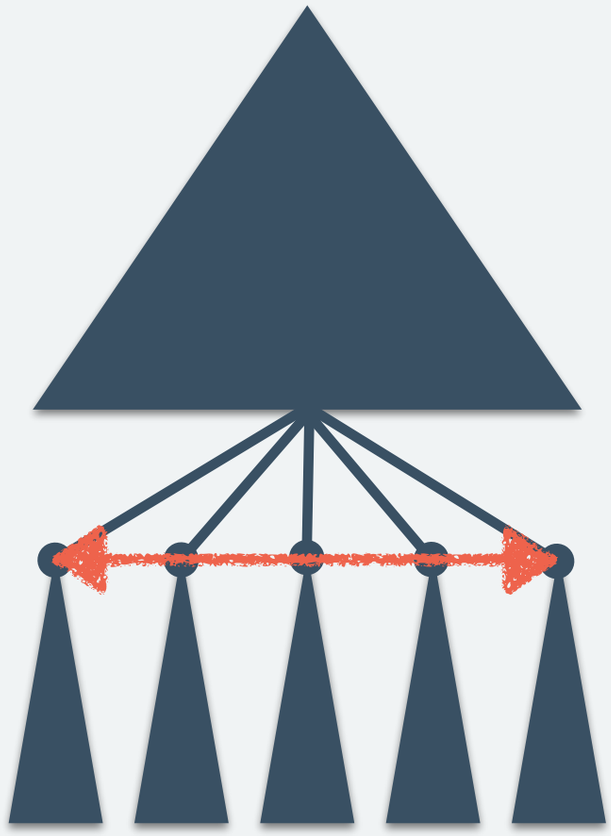
Formalisms for data trees



navigation

navigation

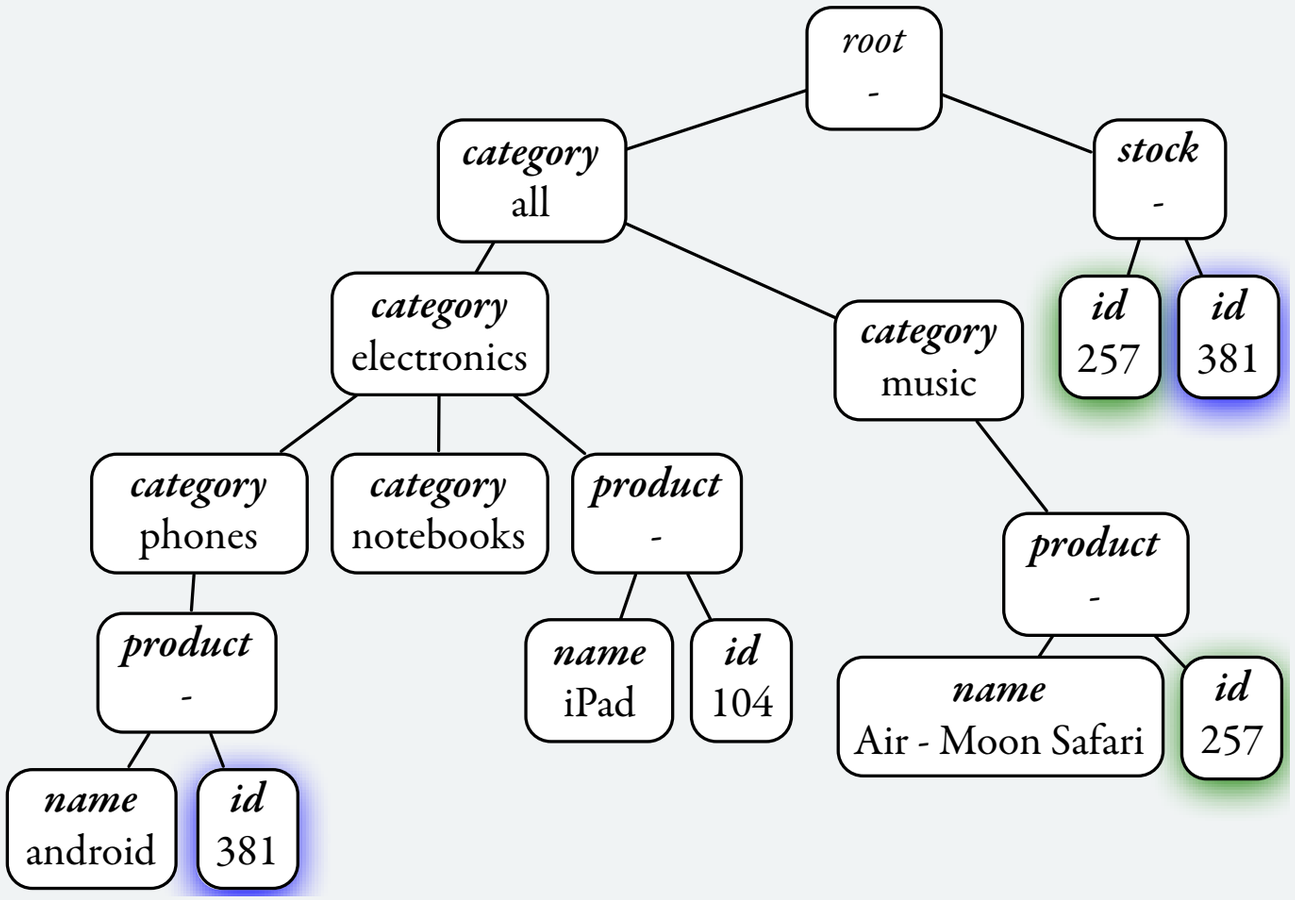




navigation

& joins

navigation



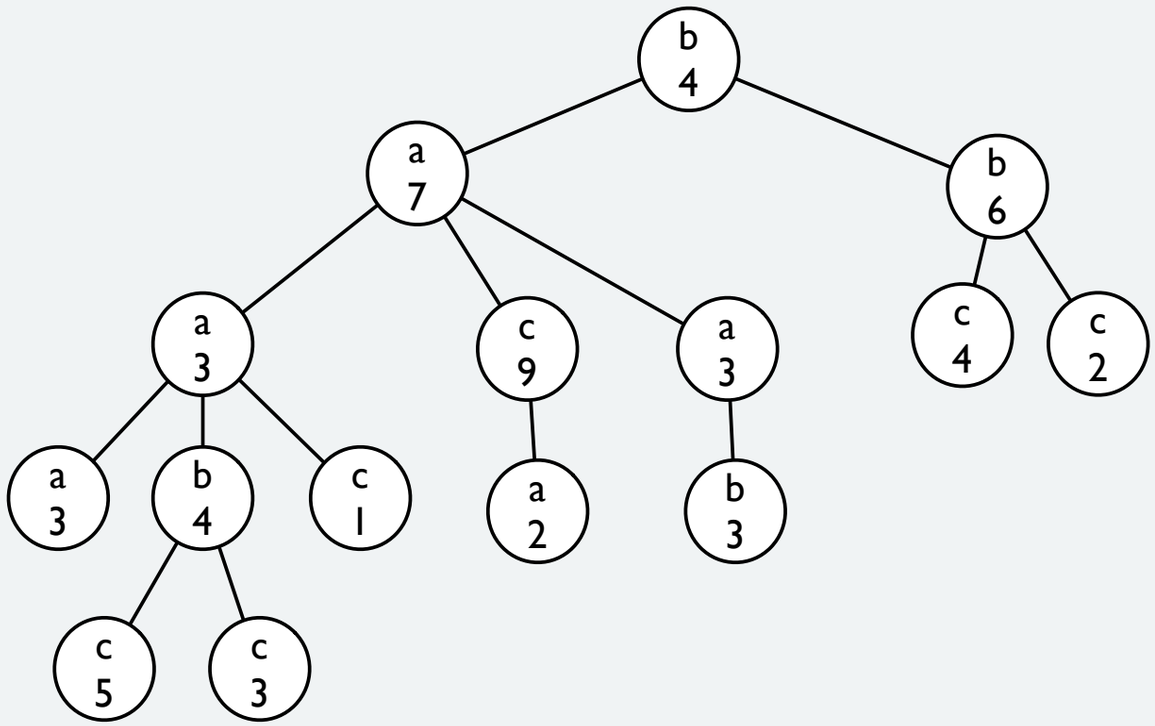
"return all products in stock"

& joins

XPath, what's that...?

two sorted language → path expressions

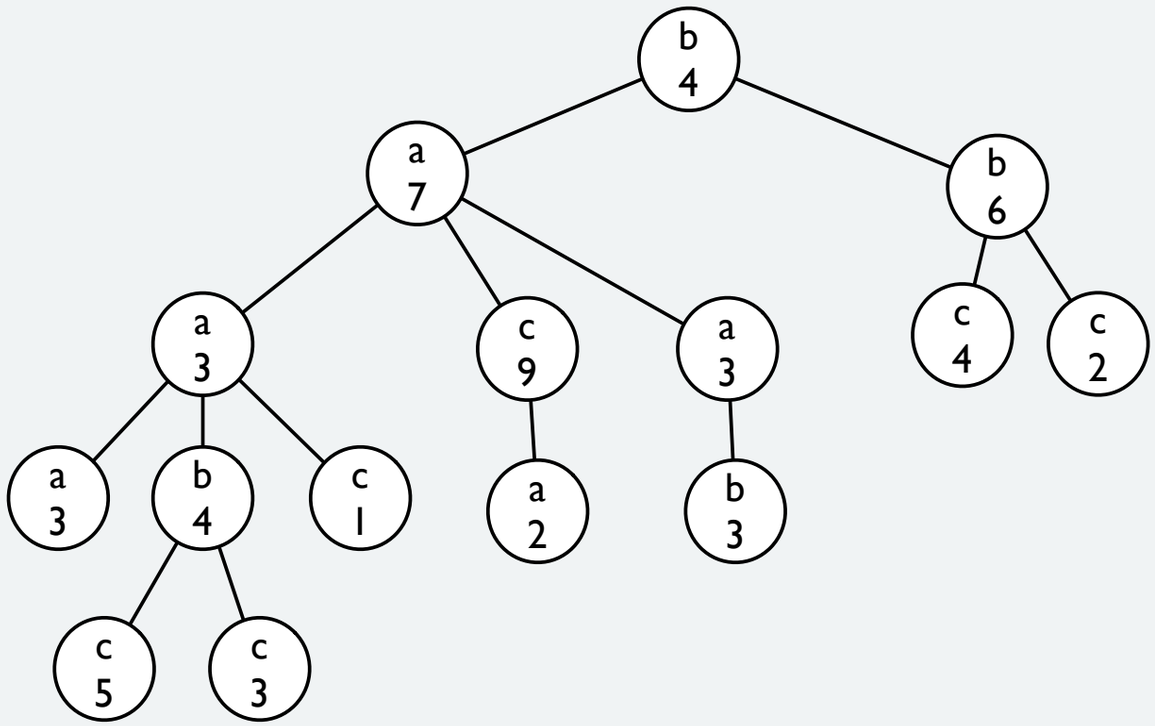
two sorted language → node expressions



XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

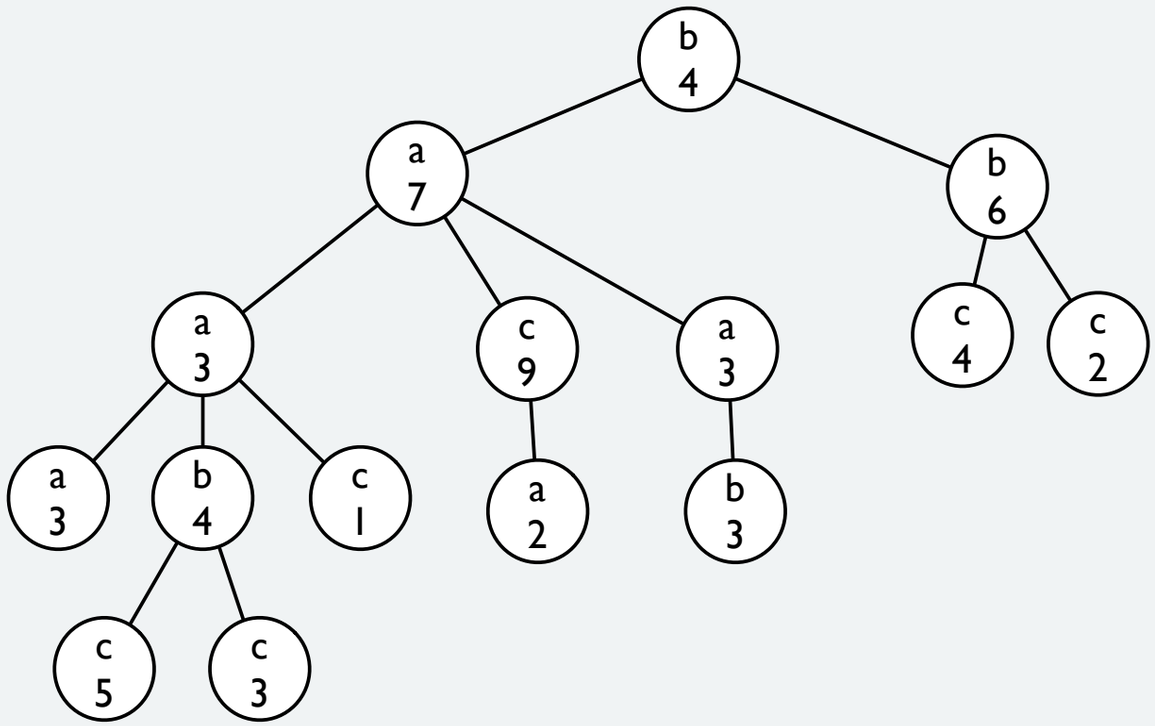
path exp



XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

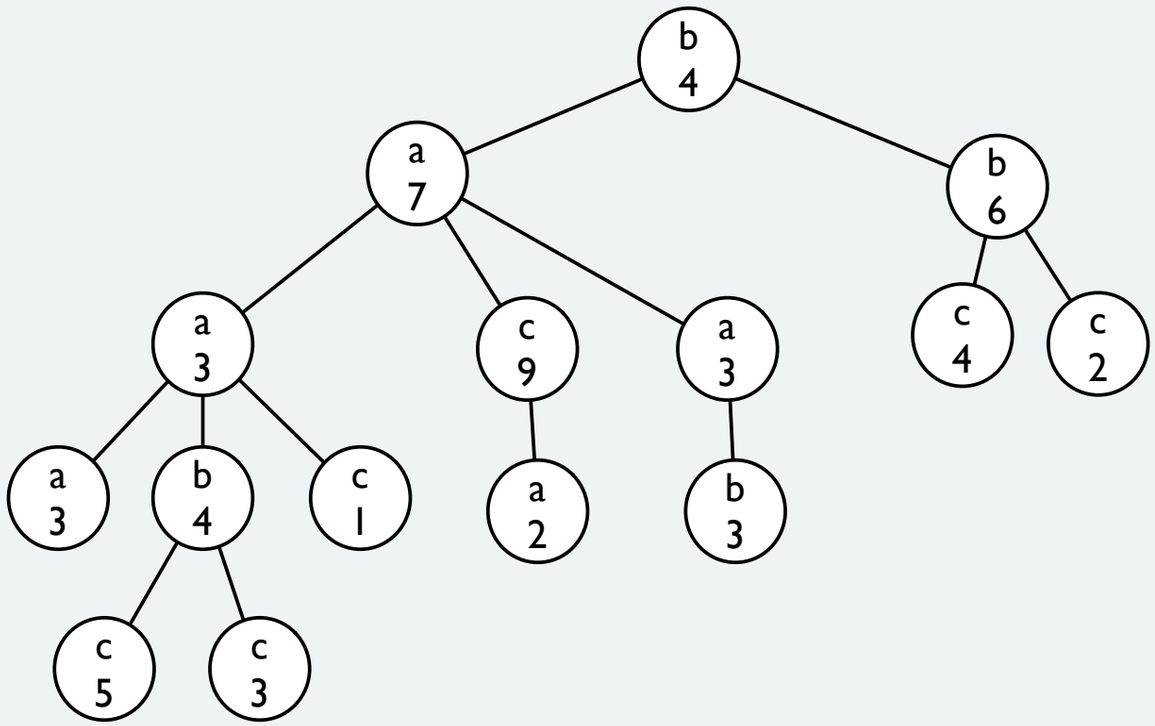
path exp	go to ancestor, go to child, go to right sibling, go to descendant



XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

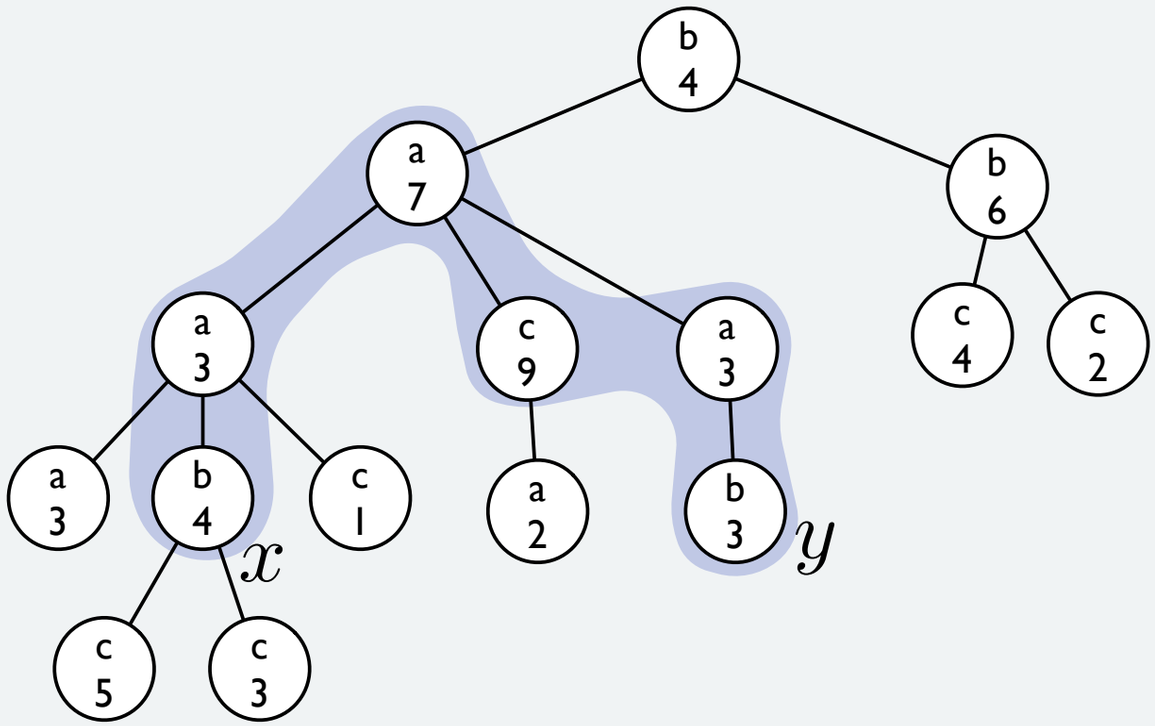
path exp $\uparrow^* \downarrow \rightarrow \downarrow_*$



XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

path exp $t, (x, y) \models \uparrow^* \downarrow \rightarrow \downarrow_*$

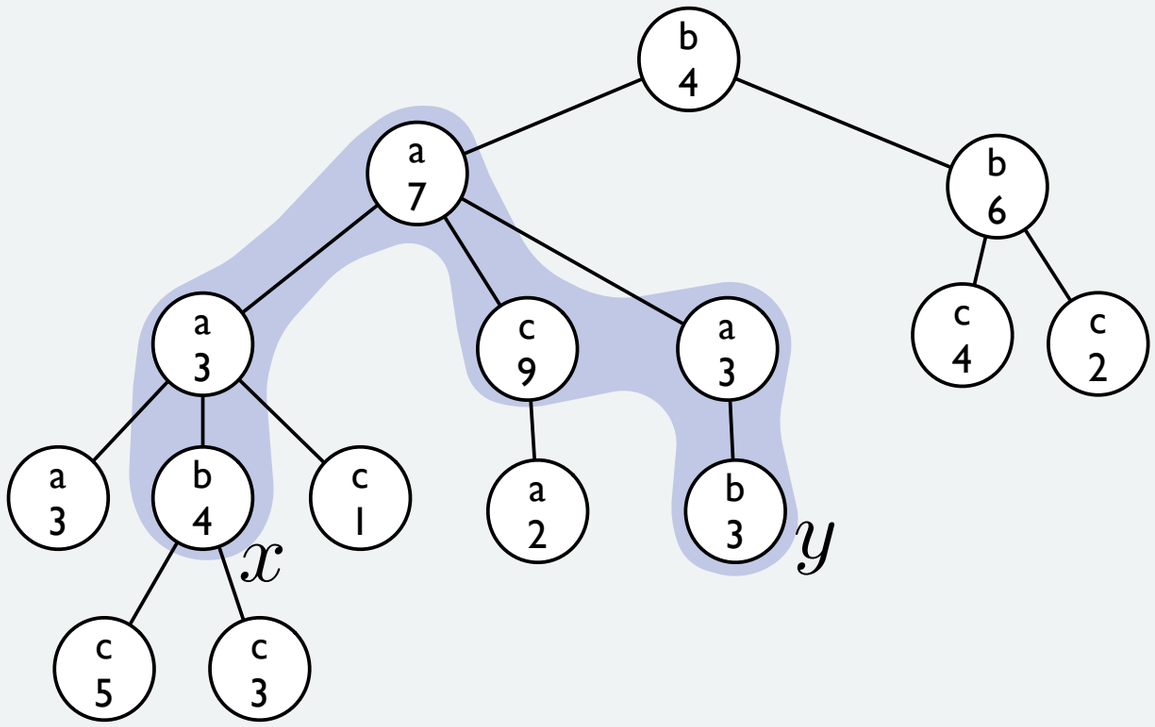


t

XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

path exp $t, (x, y) \models \uparrow^* [a] \downarrow [c] \rightarrow \downarrow_* [b]$

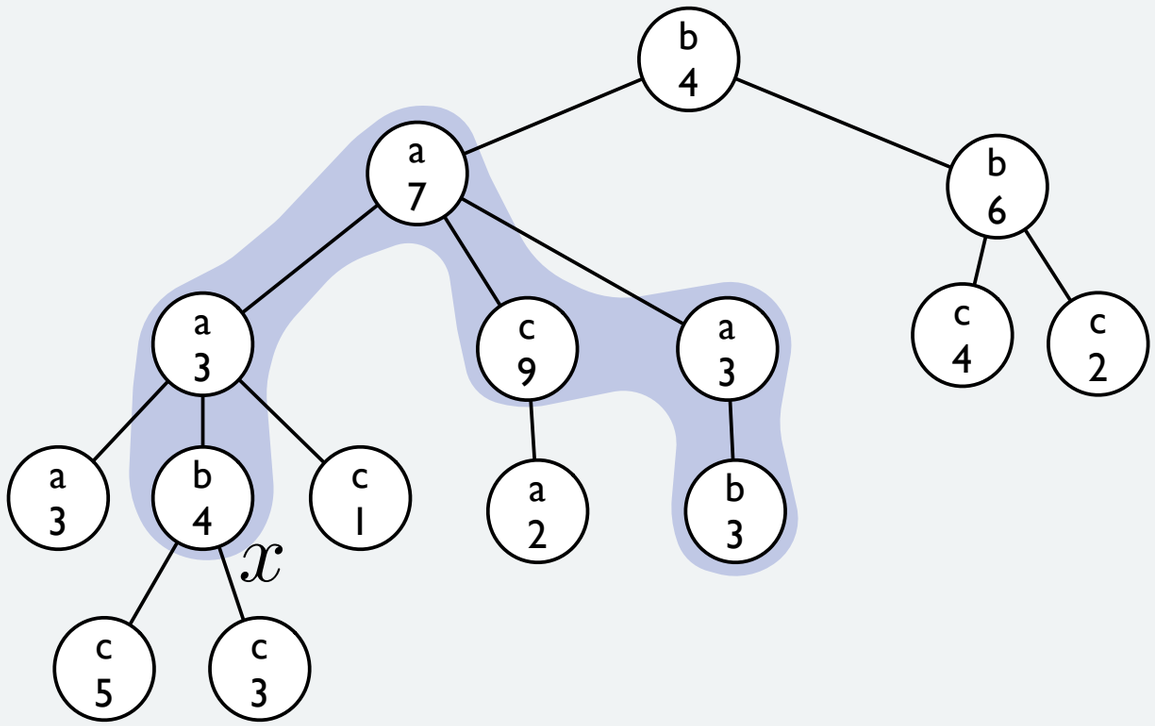


t

XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

node exp $\quad t, x \models \langle \uparrow^* [a] \downarrow [c] \rightarrow \downarrow_* [b] \rangle$

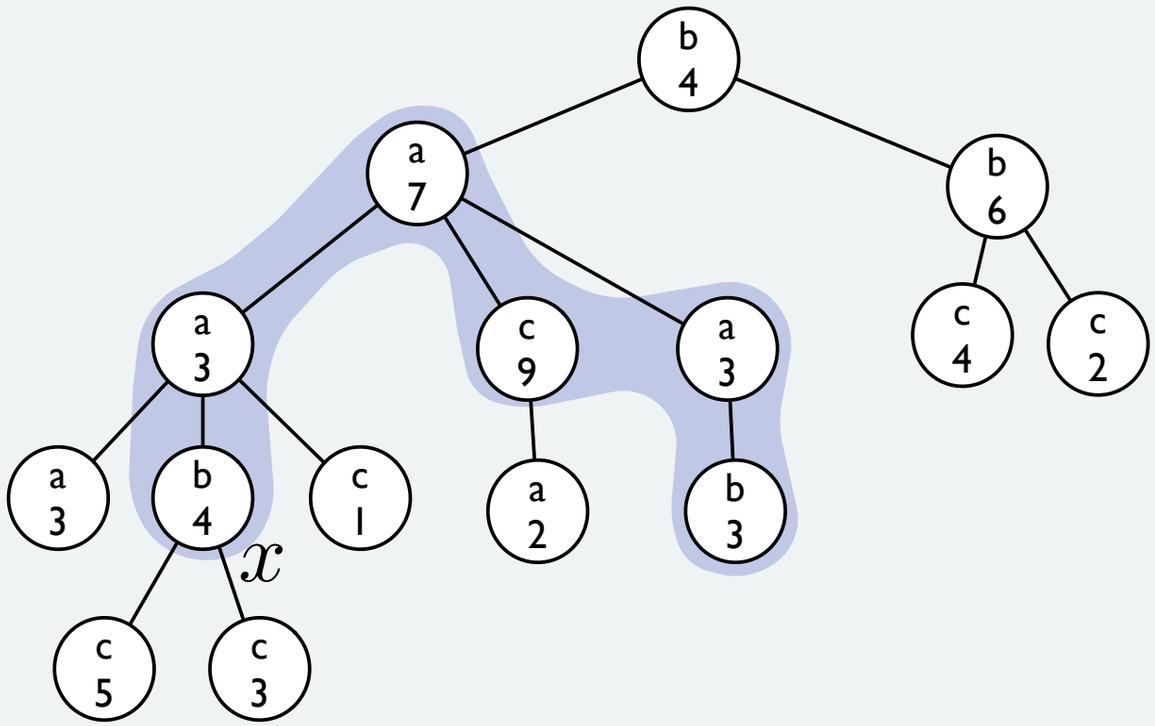


t

XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

node exp $t, x \models \langle \uparrow^* [a] \downarrow [c] \rightarrow \downarrow_* [\neg \langle \downarrow \rangle \wedge b] \rangle$

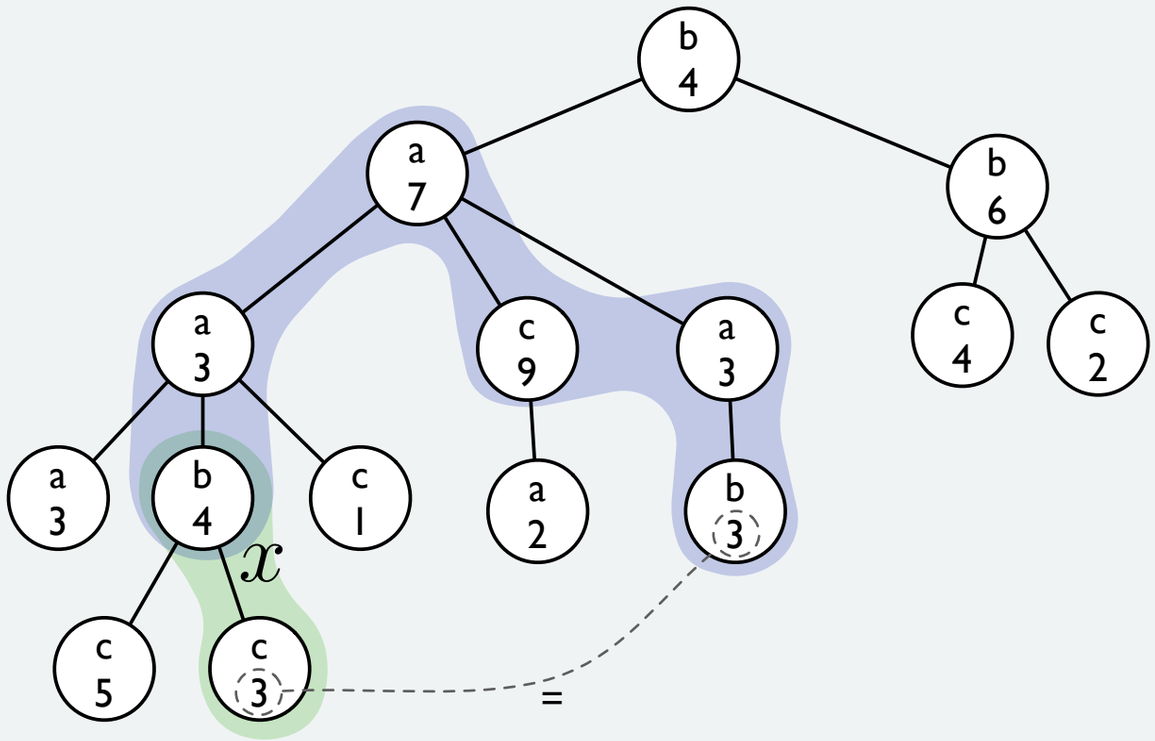


t

XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

node exp $t, x \models \langle \downarrow[c] = \uparrow^*[a]\downarrow[c] \rightarrow \downarrow_*[\neg\langle\downarrow\rangle \wedge b] \rangle$

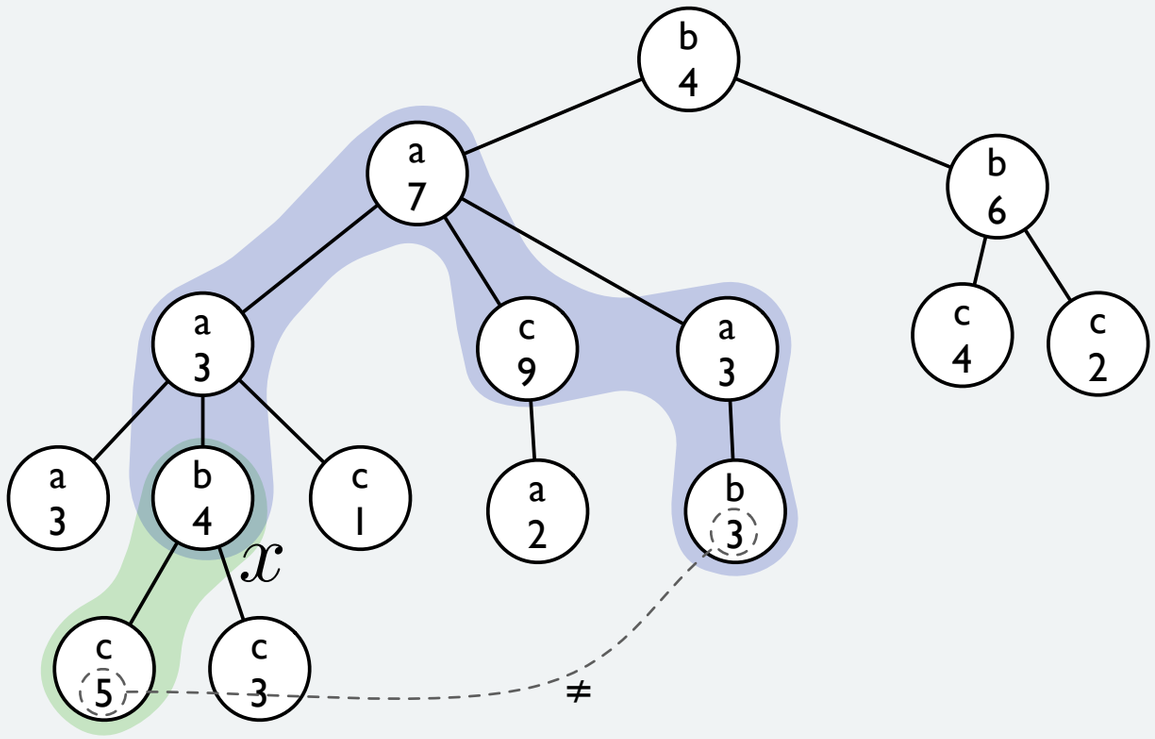


t

XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

node exp $t, x \models \langle \downarrow[c] \neq \uparrow^*[a]\downarrow[c] \rightarrow \downarrow_*[\neg\langle\downarrow\rangle \wedge b] \rangle$

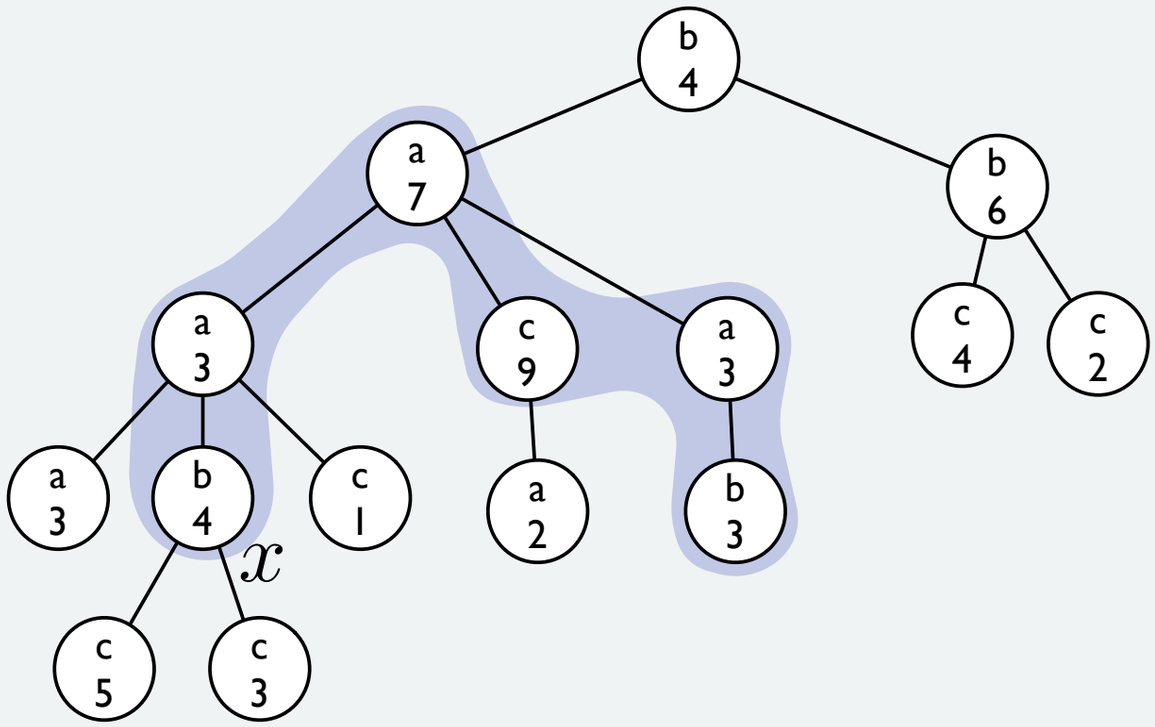


t

XPath, what's that...?

two sorted language → path expressions
two sorted language → node expressions

node exp $t, x \models \neg \langle \downarrow[c] \neq \uparrow^*[a]\downarrow[c] \rightarrow \downarrow_*[\neg\langle\downarrow\rangle \wedge b] \rangle$



t

XPath: Syntax

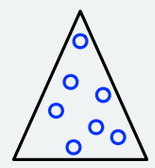
path expressions

binary relations

$$\alpha, \beta ::= \varepsilon \mid \alpha\,\beta \mid [\phi] \mid o$$

$$o \in \{\rightarrow, \rightarrow^{+}, \rightarrow^{*}, \leftarrow, {}^{+}\leftarrow, {}^{*}\leftarrow, \downarrow, \downarrow_{+}, \downarrow_{*}, \uparrow, \uparrow^{+}, \uparrow^{*}\}$$

node expressions

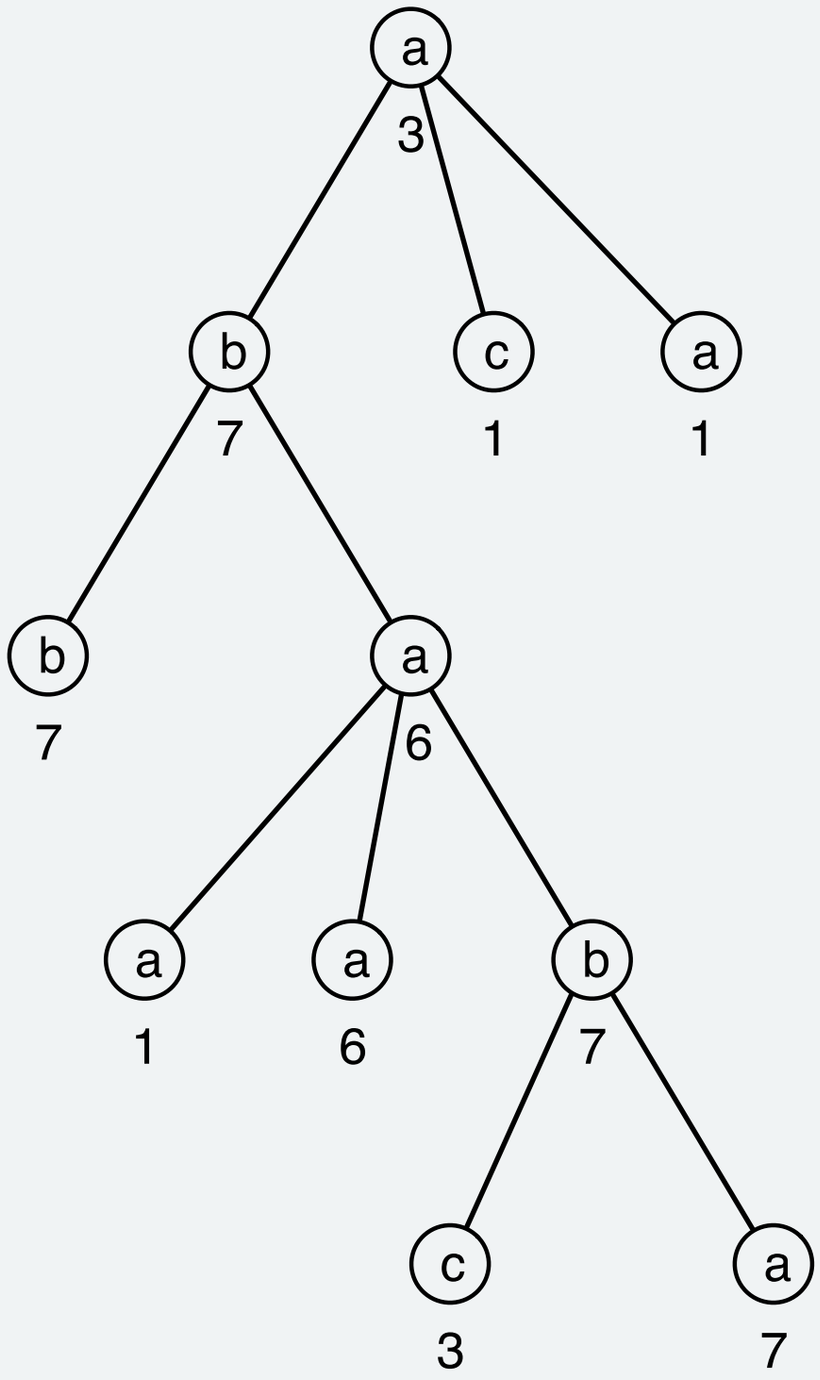


sets of nodes

$$\phi, \psi ::= a \mid \neg\,\phi \mid \phi \wedge \psi \mid \langle\,\alpha\,\rangle \mid \langle\,\alpha = \beta\,\rangle \mid \langle\,\alpha \neq \beta\,\rangle$$

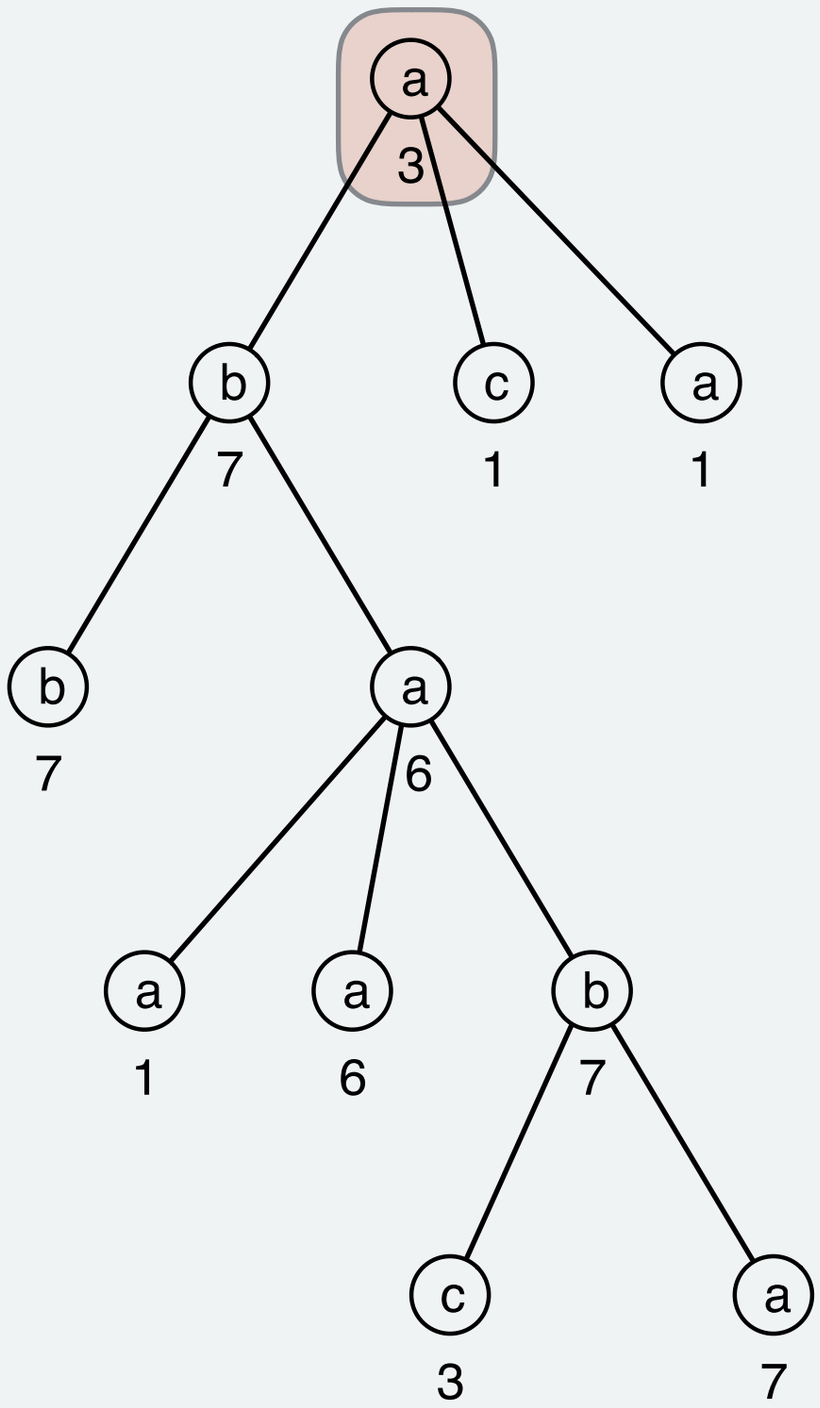
$$a \in \mathrm{A}$$

Semantics by Examples



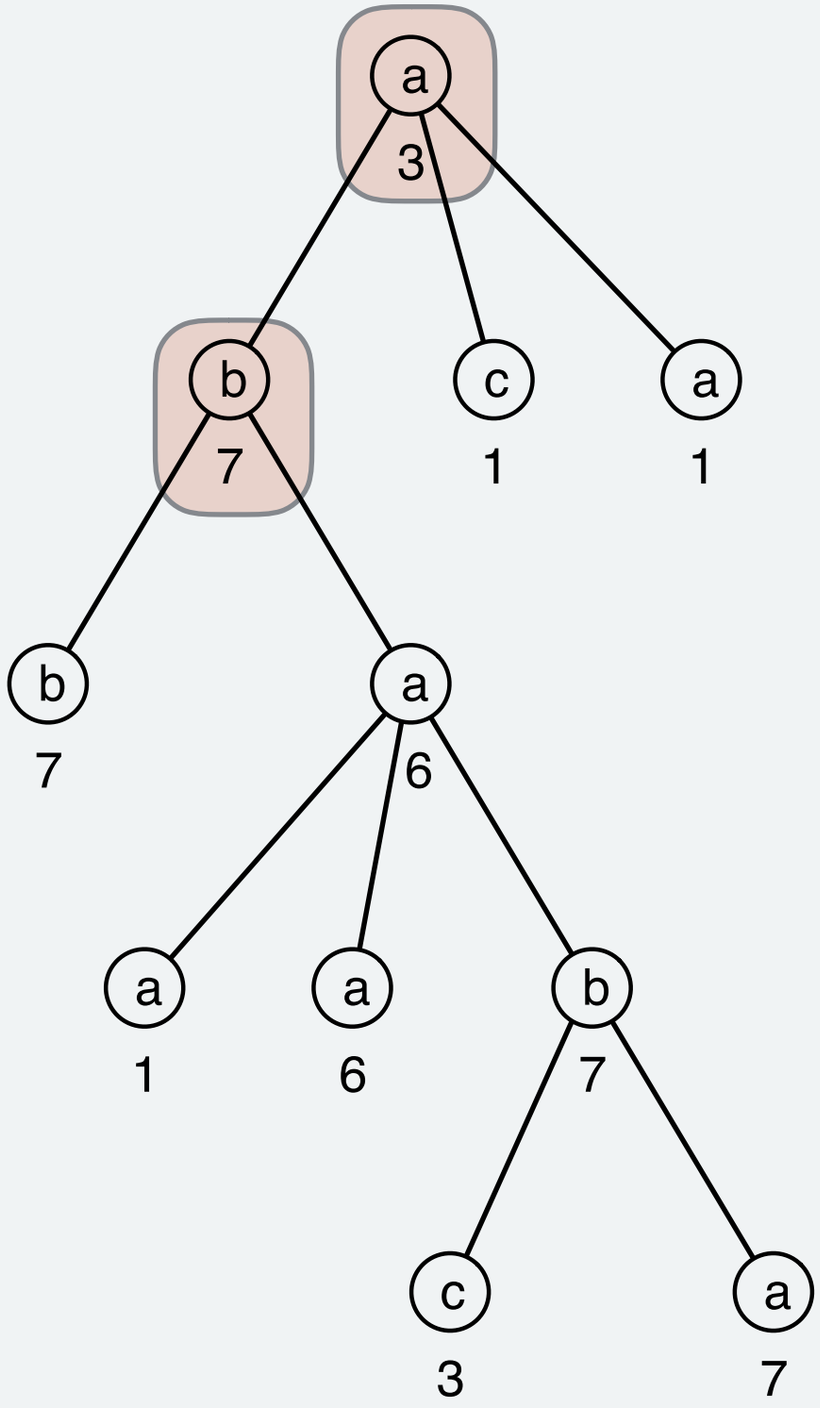
$$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$$

Semantics by Examples



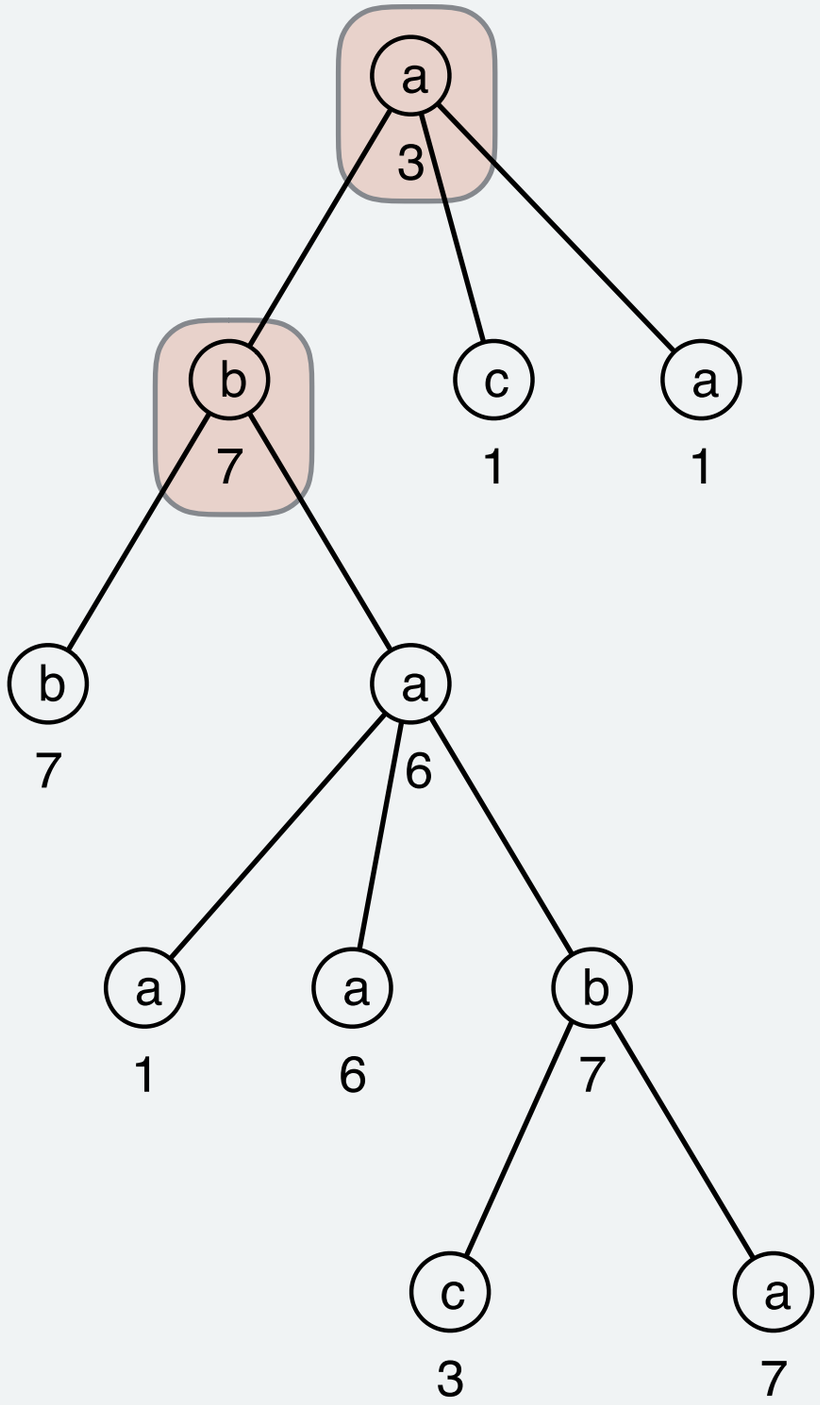
$$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$$

Semantics by Examples



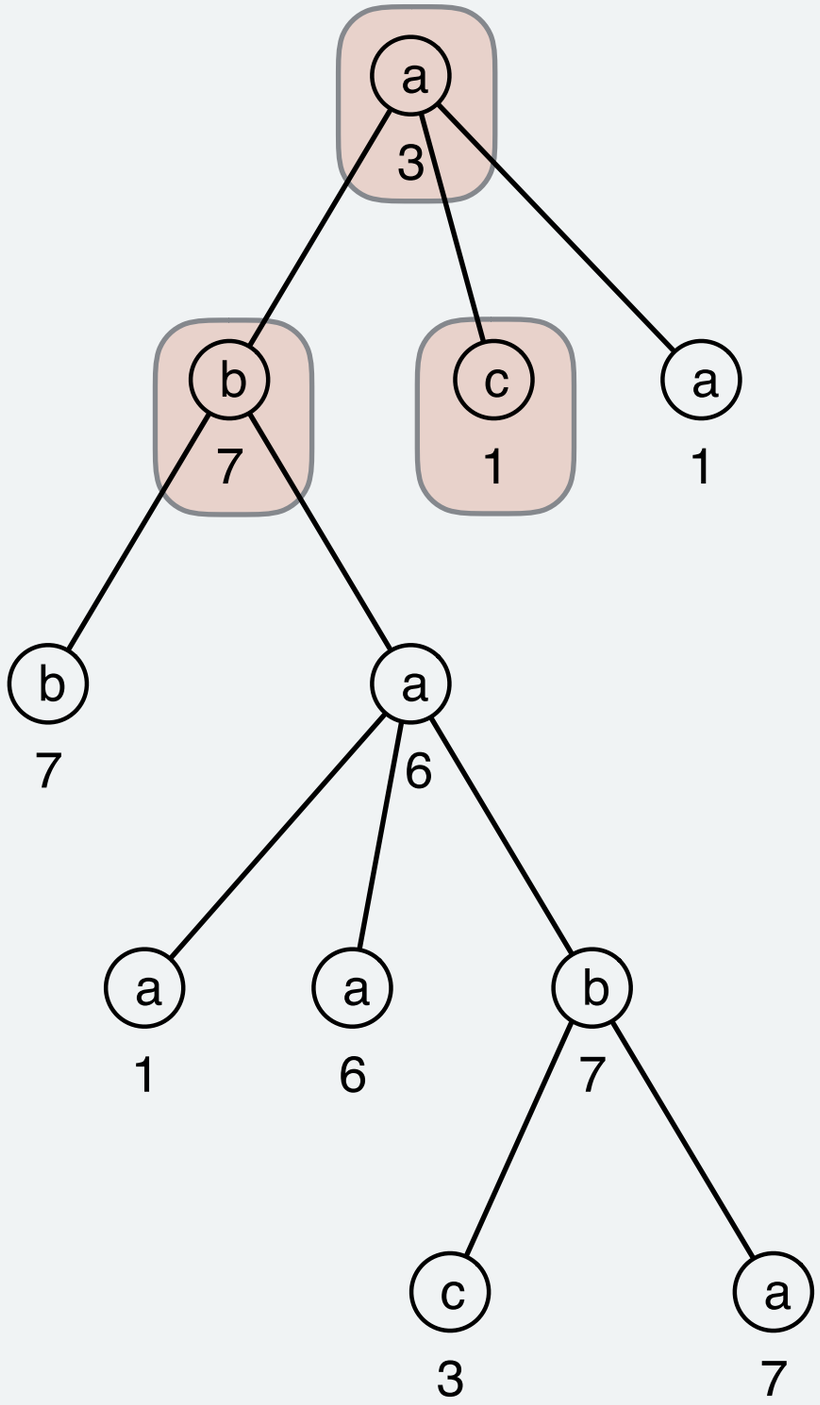
$$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$$

Semantics by Examples



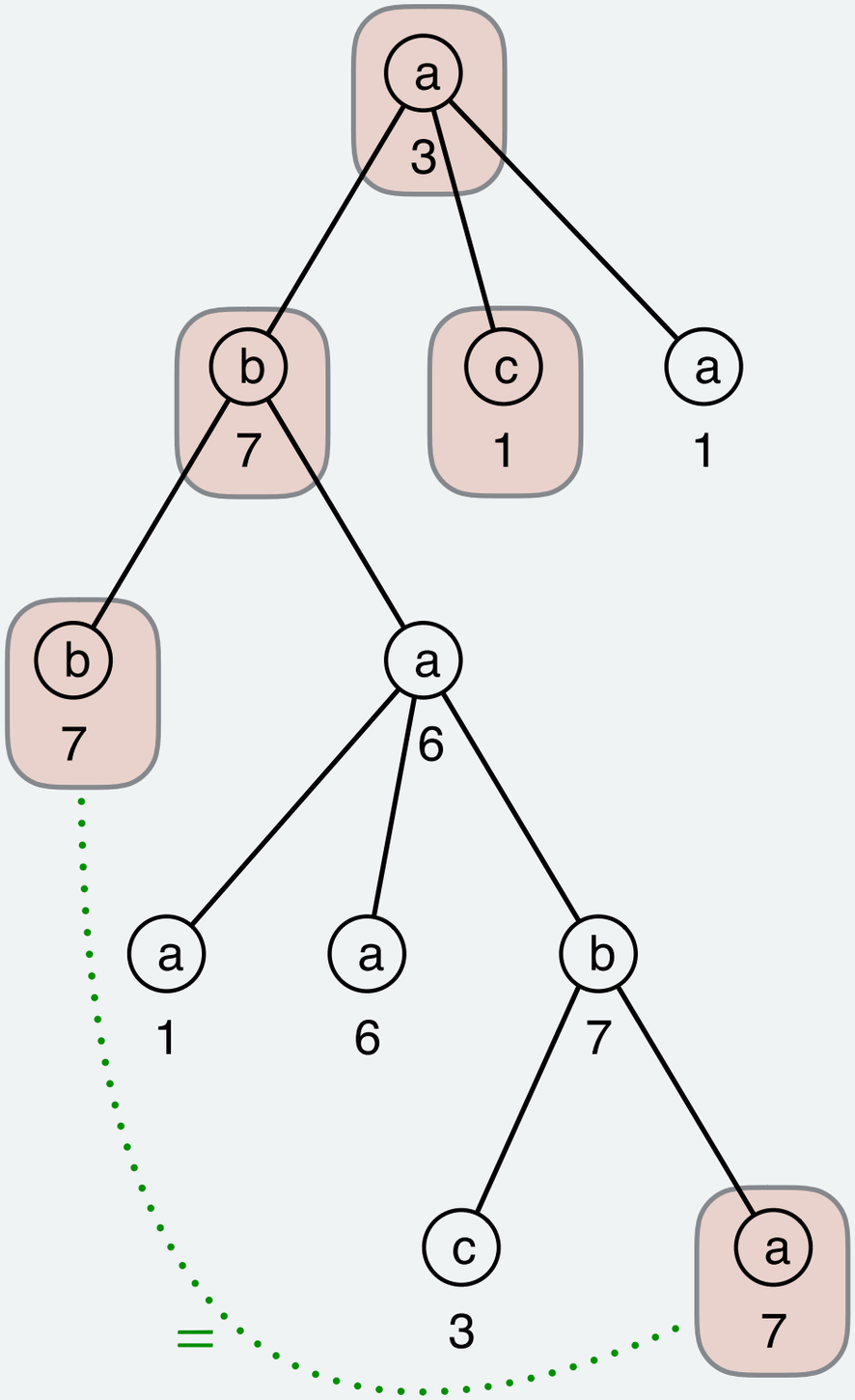
$$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$$

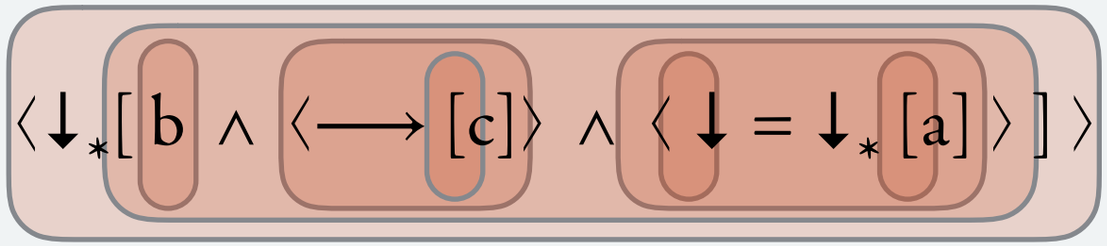
Semantics by Examples



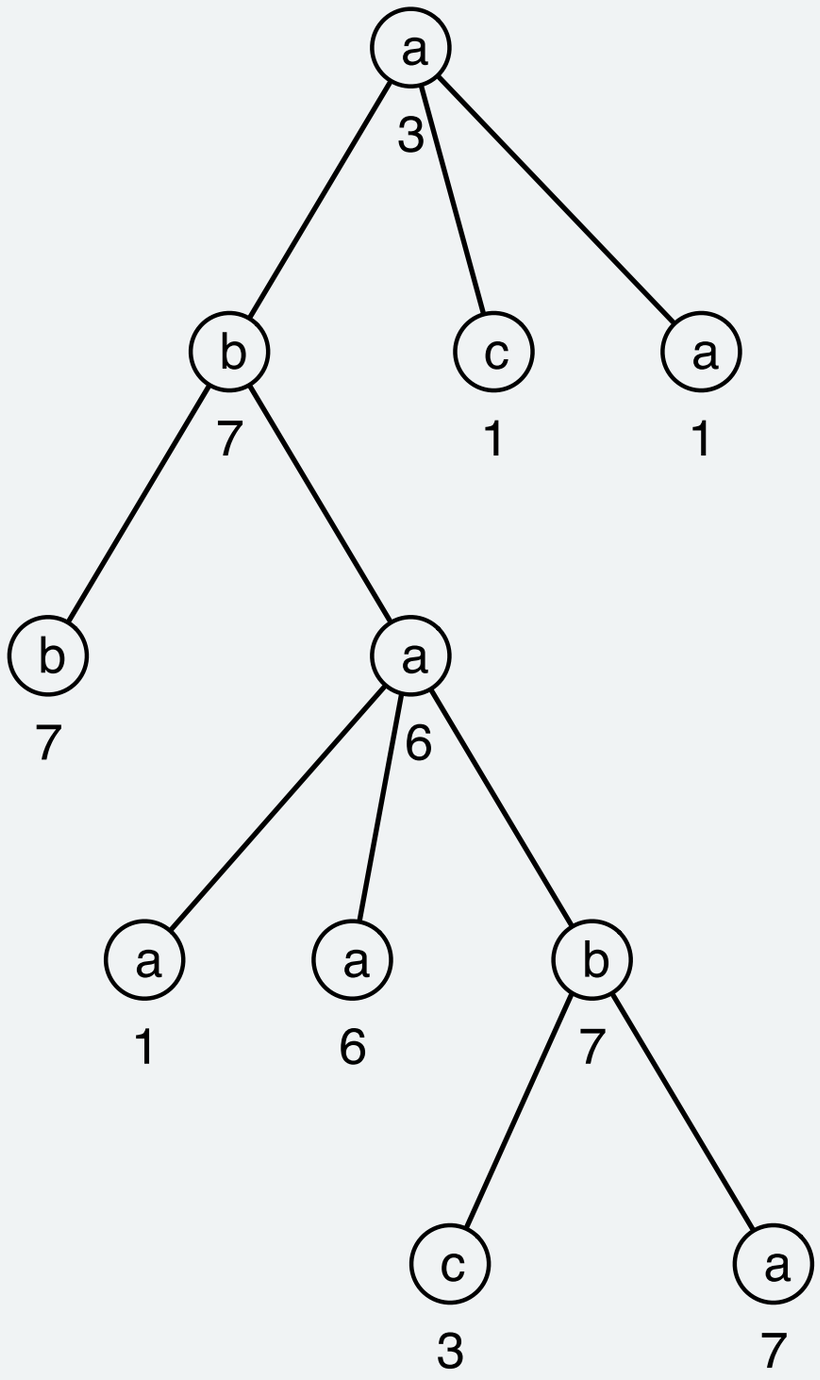
$$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$$

Semantics by Examples



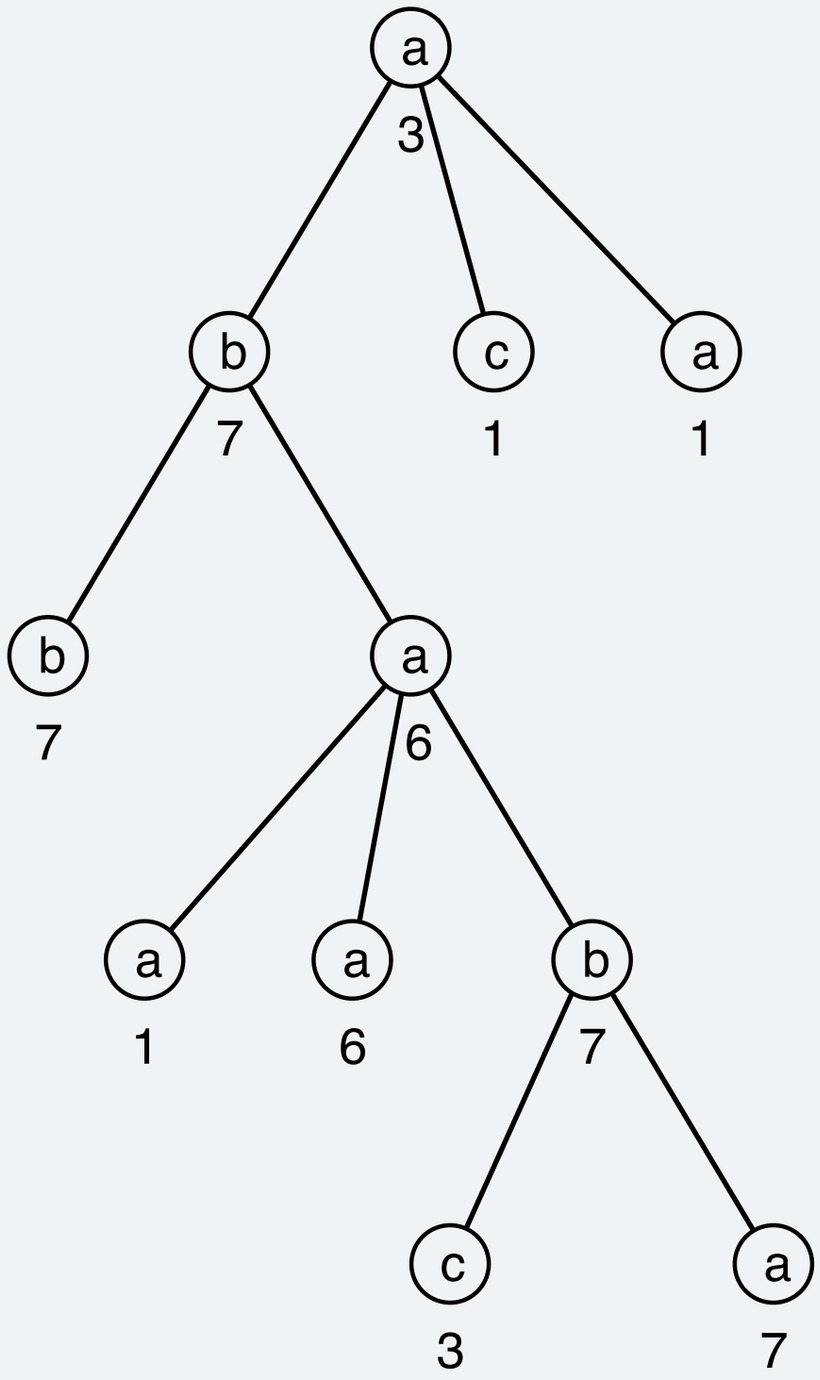


Semantics by Examples



$$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$$

Semantics by Examples

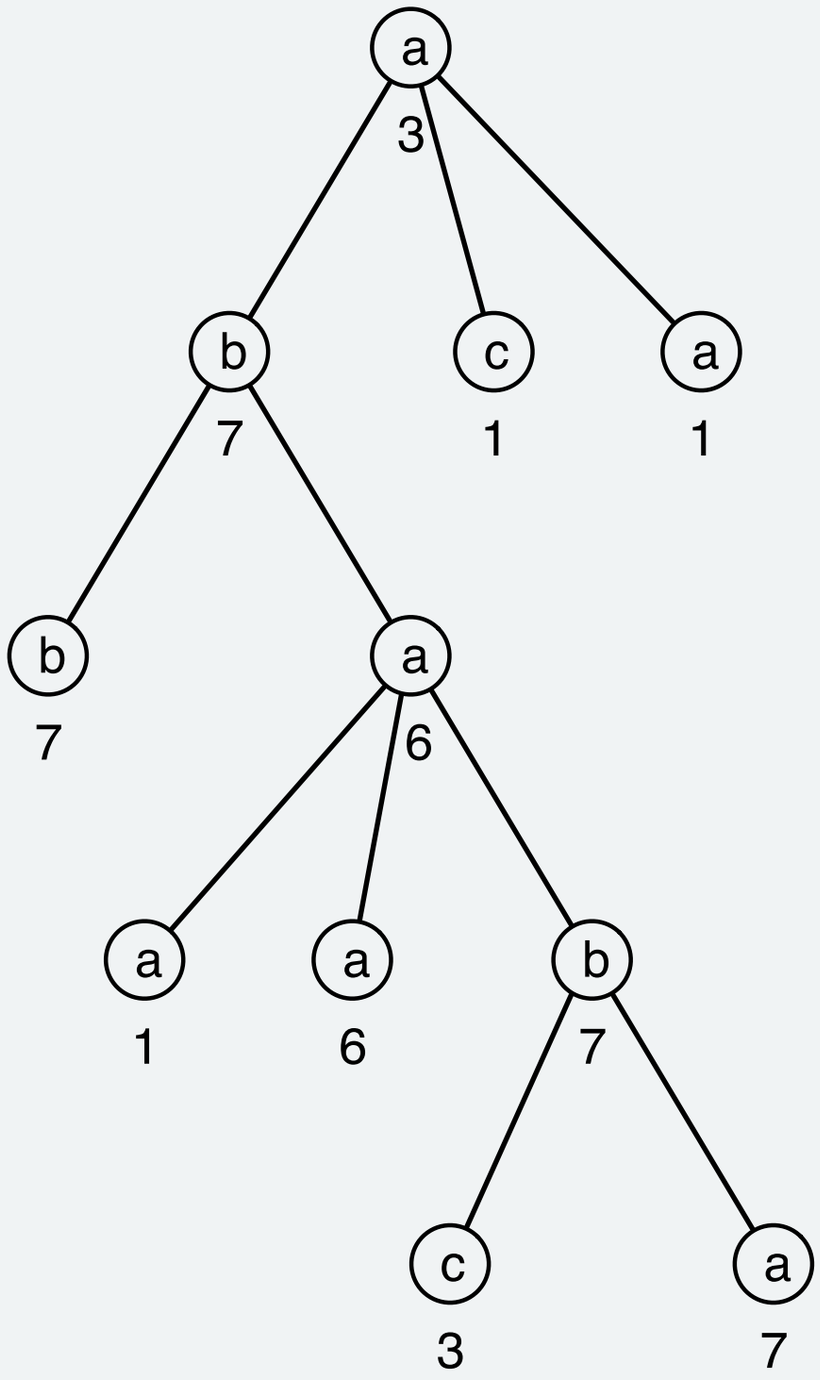


$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$

There is an **a** and a **b** nodes with the same data value.

Semantics by Examples



$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

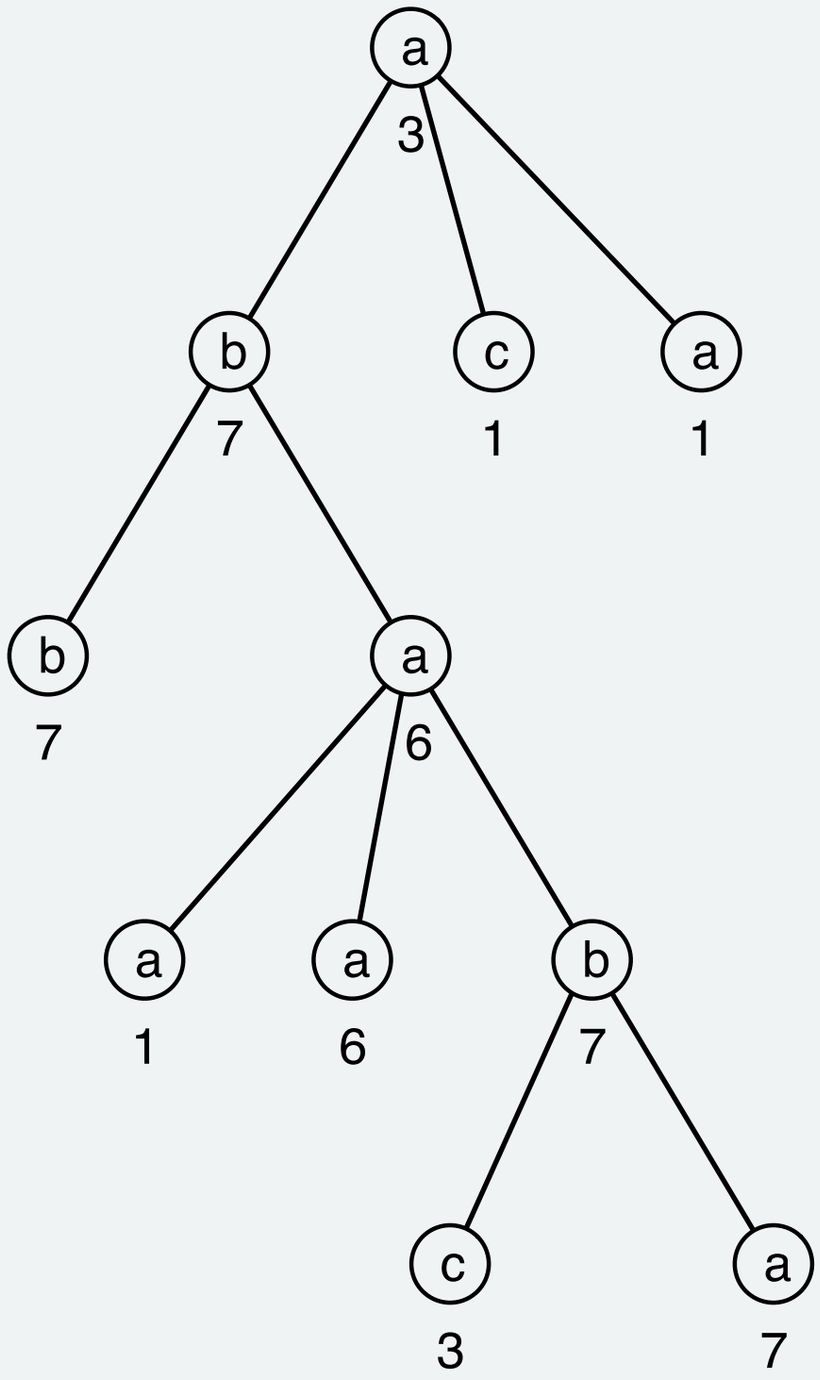
$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$

There is an **a** and a **b** nodes with the same data value.

$\langle \downarrow_* [c] \neq \downarrow_* [c] \rangle$

There are two **c**'s with different data value.

Semantics by Examples



$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$

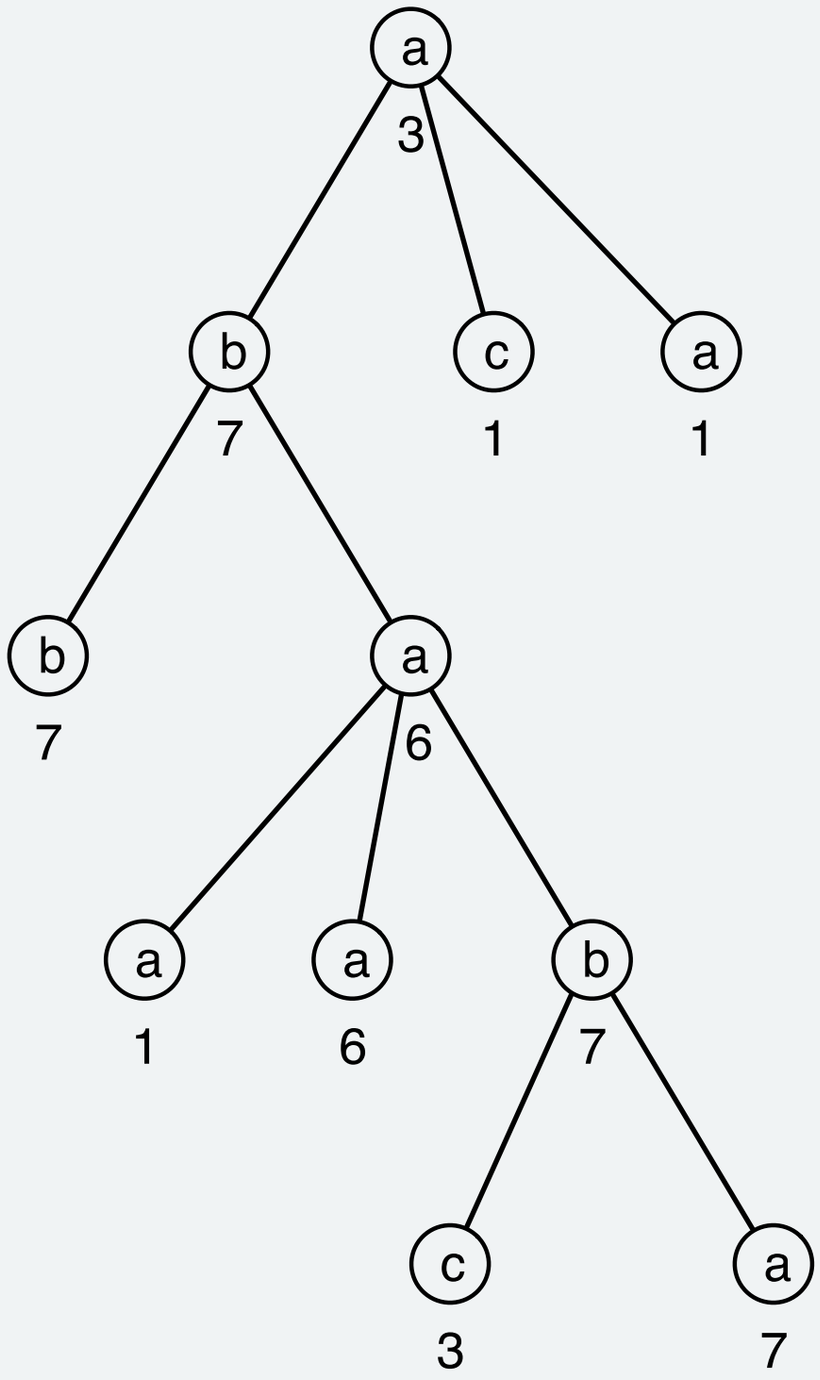
There is an **a** and a **b** nodes with the same data value.

$\langle \downarrow_* [c] \neq \downarrow_* [c] \rangle$

There are two **c**'s with different data value.

$\neg \langle \downarrow_* [b] \neq \downarrow_* [b] \rangle$

Semantics by Examples



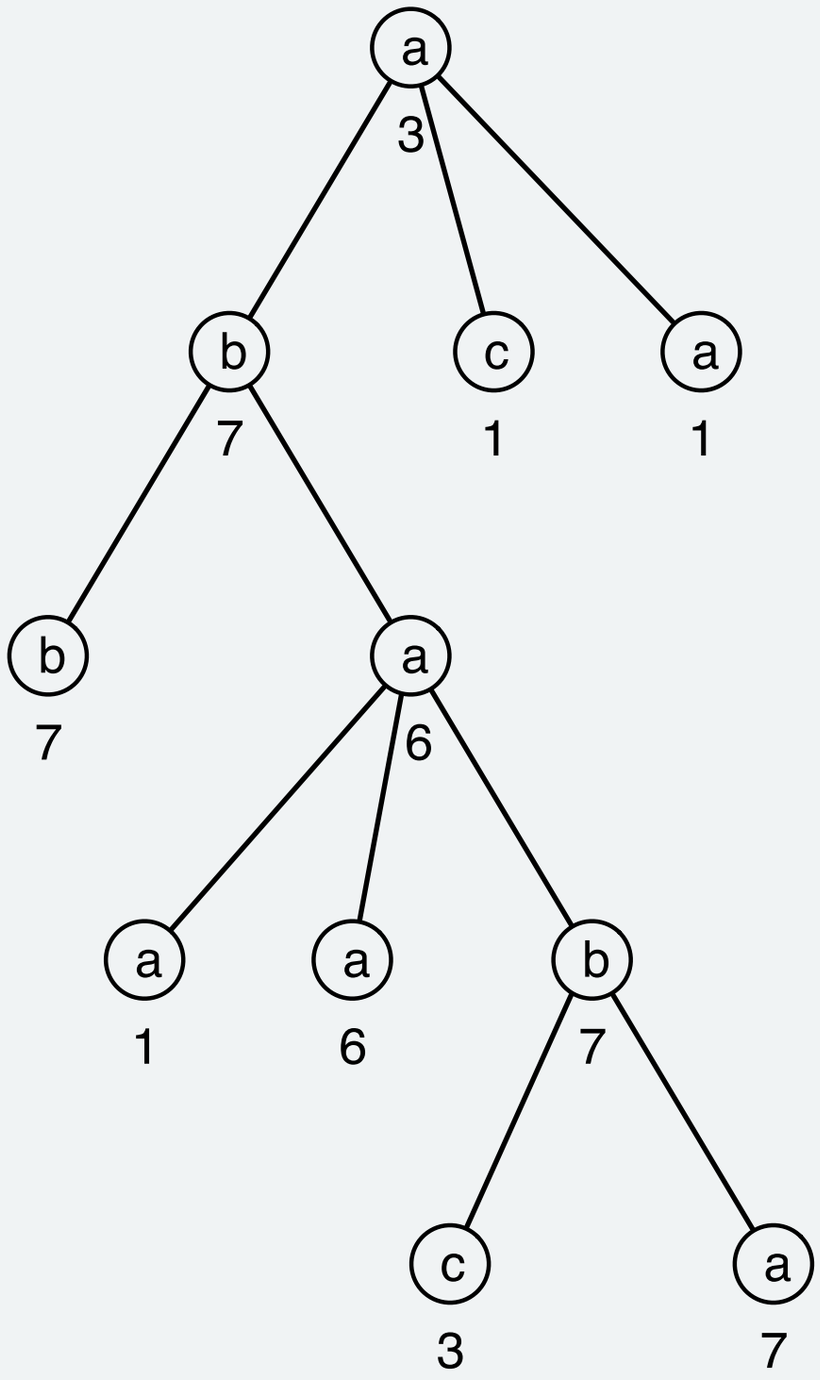
$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$ There is an **a** and a **b** nodes with the same data value.

$\langle \downarrow_* [c] \neq \downarrow_* [c] \rangle$ There are two **c**'s with different data value.

$\neg \langle \downarrow_* [b] \neq \downarrow_* [b] \rangle$ All the **b**'s have the same value.

Semantics by Examples



$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

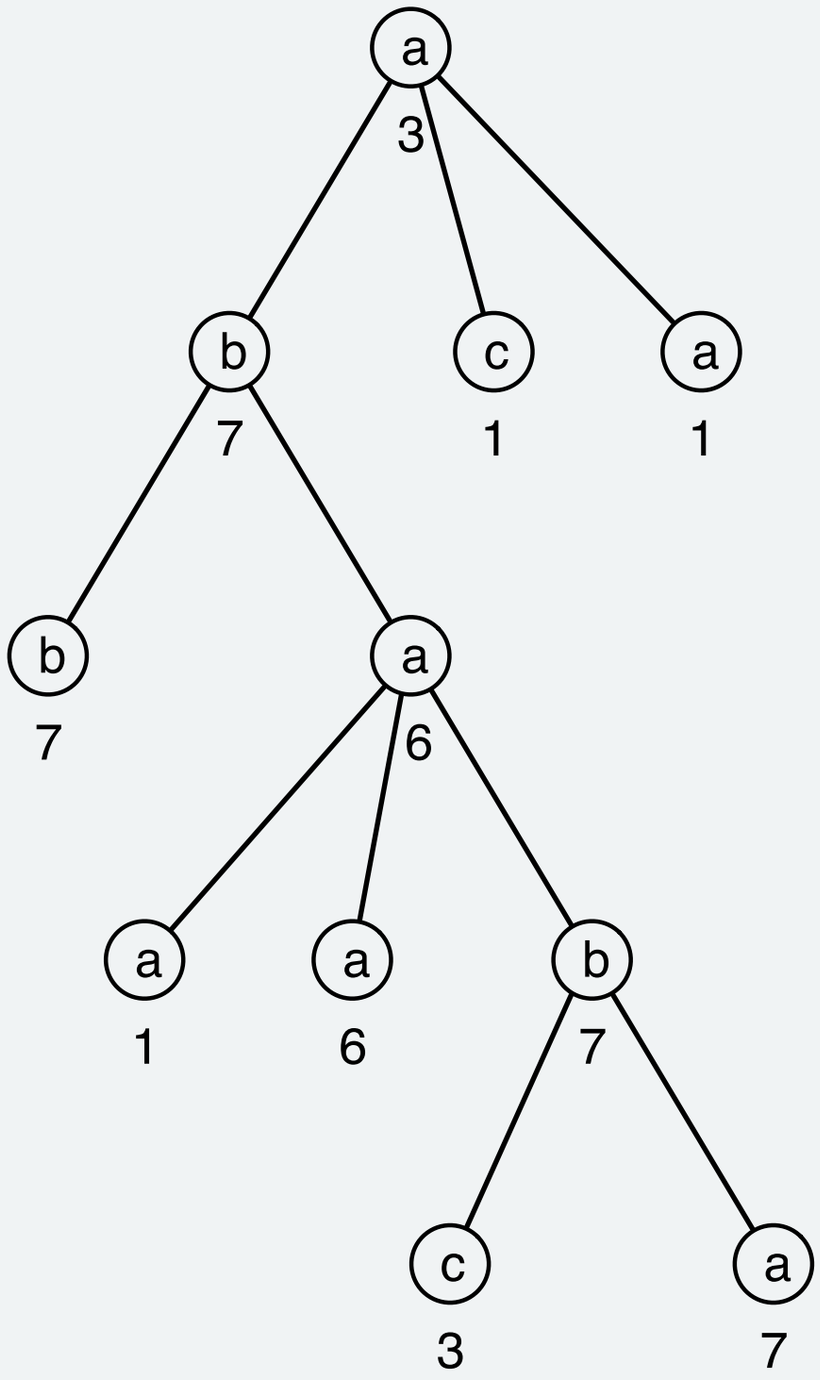
$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$ There is an **a** and a **b** nodes with the same data value.

$\langle \downarrow_* [c] \neq \downarrow_* [c] \rangle$ There are two **c**'s with different data value.

$\neg \langle \downarrow_* [b] \neq \downarrow_* [b] \rangle$ All the **b**'s have the same value.

$\neg \langle \downarrow_* [b] = \downarrow_* [c] \rangle$

Semantics by Examples



$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

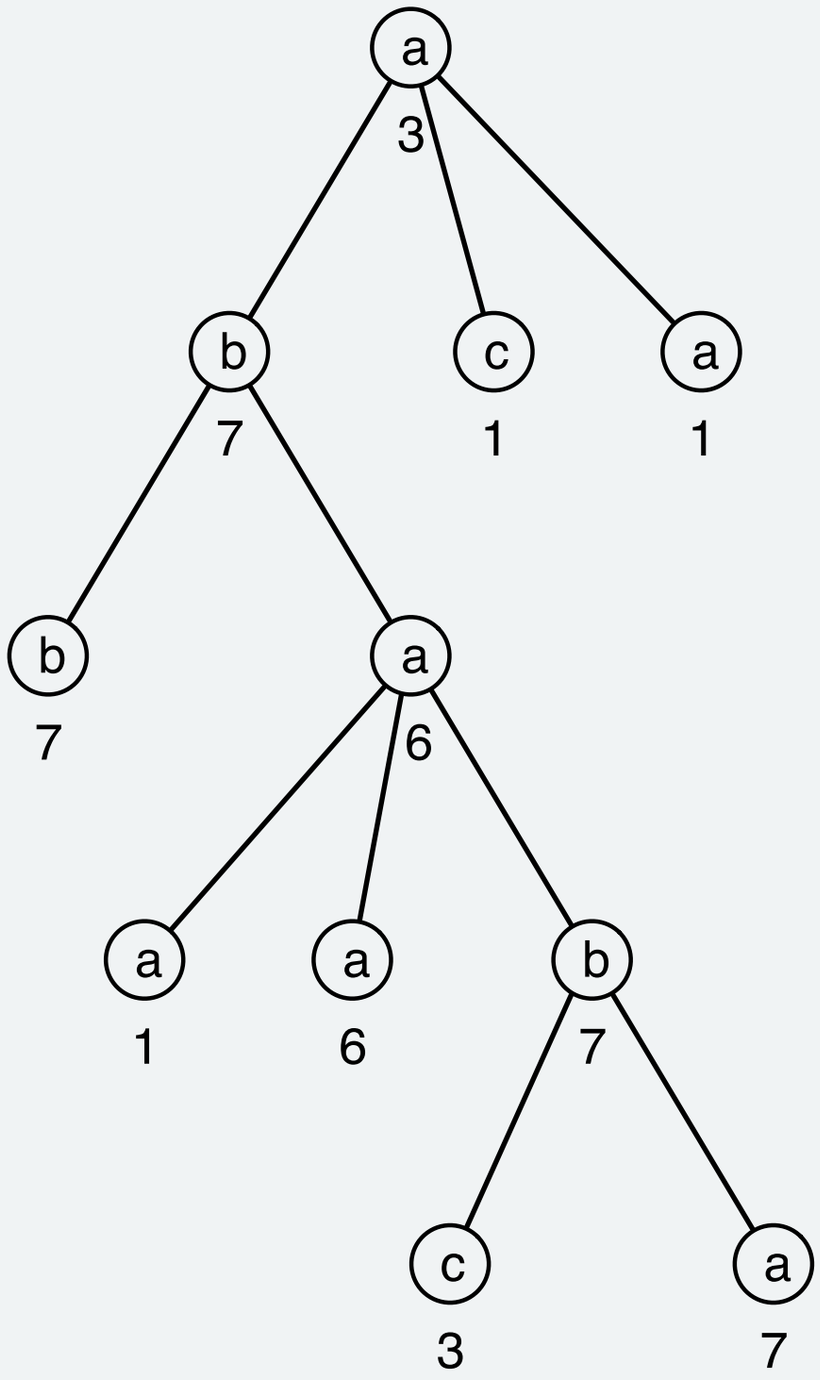
$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$ There is an **a** and a **b** nodes with the same data value.

$\langle \downarrow_* [c] \neq \downarrow_* [c] \rangle$ There are two **c**'s with different data value.

$\neg \langle \downarrow_* [b] \neq \downarrow_* [b] \rangle$ All the **b**'s have the same value.

$\neg \langle \downarrow_* [b] = \downarrow_* [c] \rangle$ There is no data value shared by a **b** and a **c**.

Semantics by Examples



$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$ There is an **a** and a **b** nodes with the same data value.

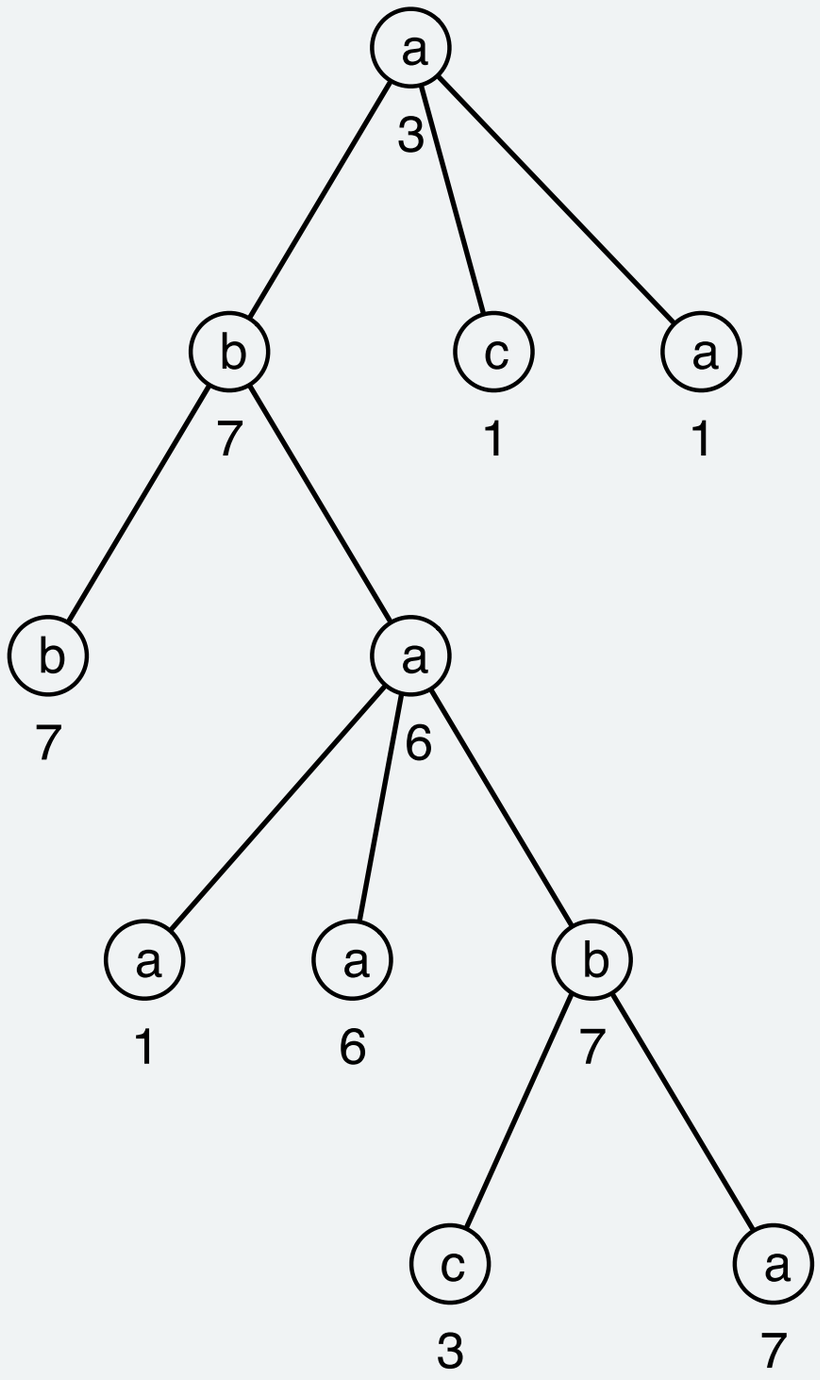
$\langle \downarrow_* [c] \neq \downarrow_* [c] \rangle$ There are two **c**'s with different data value.

$\neg \langle \downarrow_* [b] \neq \downarrow_* [b] \rangle$ All the **b**'s have the same value.

$\neg \langle \downarrow_* [b] = \downarrow_* [c] \rangle$ There is no data value shared by a **b** and a **c**.

???? **c** is a *key*

Semantics by Examples



$\langle \downarrow_* [\, b \wedge \langle \longrightarrow [c] \rangle \wedge \langle \downarrow = \downarrow_* [a] \rangle \,] \rangle$

$\langle \downarrow_* [a] = \downarrow_* [b] \rangle$ There is an **a** and a **b** nodes with the same data value.

$\langle \downarrow_* [c] \neq \downarrow_* [c] \rangle$ There are two **c**'s with different data value.

$\neg \langle \downarrow_* [b] \neq \downarrow_* [b] \rangle$ All the **b**'s have the same value.

$\neg \langle \downarrow_* [b] = \downarrow_* [c] \rangle$ There is no data value shared by a **b** and a **c**.

???? **c** is a *key*

$\neg \langle \downarrow_* [c \wedge (\langle \varepsilon = \uparrow^* \longrightarrow \longrightarrow^* \downarrow_* [c] \rangle \vee \langle \varepsilon = \uparrow \uparrow^* [c] \rangle)] \rangle$

Satisfiability of XPath

Satisfiability of XPath

Undecidable

full XPath ♠

♠ [Geerts, Fan, PODS'05]

Satisfiability of XPath

Decidable, non-PR

Forward ♦

$\mathsf{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$

Vertical ♣

$\mathsf{XPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*)$

Undecidable

full XPath ♠

♦ [F. ICDT'10]

♣ [F., Segoufin, STACS'10]

♠ [Geerts, Fan, PODS'05]

Satisfiability of XPath

ExpTime

Decidable, non-PR

Undecidable

Downward ♥
XPath($\downarrow, \downarrow_*$)

XPath without data tests ♠

Forward ♦
XPath($\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$)

Vertical ♣
XPath($\downarrow, \downarrow_*, \uparrow, \uparrow^*$)

full XPath ♠

♥ [F. PODS'09]
♦ [F. ICDT'10]
♣ [F., Segoufin, STACS'10]
♠ [Geerts, Fan, PODS'05]

Satisfiability of XPath

NP

Positive-XPath ♠

ExpTime

Downward ♥
XPath($\downarrow, \downarrow_*$)

XPath without data tests ♠

Decidable, non-PR

Forward ♦
XPath($\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$)

Vertical ♣
XPath($\downarrow, \downarrow_*, \uparrow, \uparrow^*$)

Undecidable

full XPath ♠

♥ [F. PODS'09]
♦ [F. ICDT'10]
♣ [F., Segoufin, STACS'10]
♠ [Geerts, Fan, PODS'05]

Satisfiability of XPath

XPath: undecidable♣
XPath without data tests: ExpTime-complete♣
Positive-XPath: NP-complete♣

♣ [Geerts, Fan, DBPL 2005]

Satisfiability of XPath

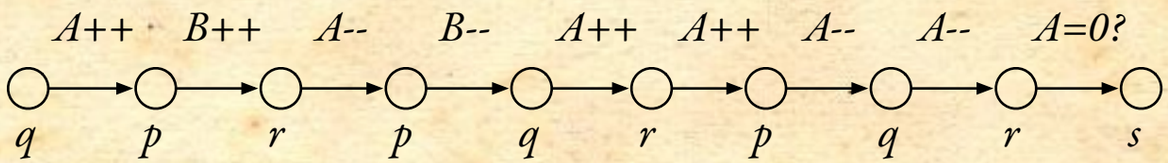
XPath: undecidable♣

XPath without data tests: ExpTime-complete♣

Positive-XPath: NP-complete♣

By reduction from the 2-counter Minsky automaton.

Any accepting run of the automaton,



♣ [Geerts, Fan, DBPL 2005]

Satisfiability of XPath

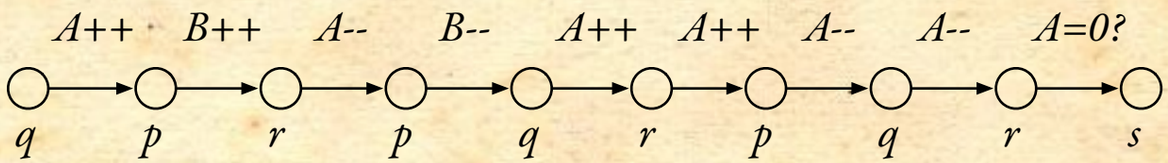
XPath: undecidable ♣

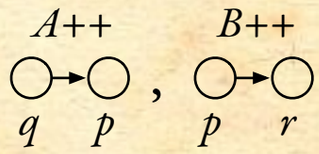
XPath without data tests: ExpTime-complete ♣

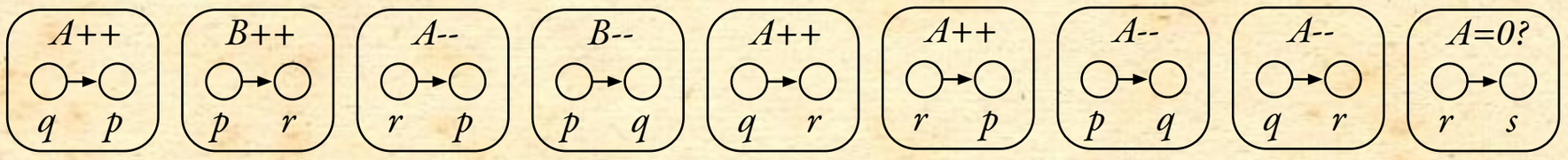
Positive-XPath: NP-complete ♣

By reduction from the 2-counter Minsky automaton.

Any accepting run of the automaton,



Has associated a tree over the finite alphabet $\Sigma = \{$  $, \ldots\}$



This is a forest, but the proof would be equivalent adding a root. And vice-versa.

♣ [Geerts, Fan, DBPL 2005]

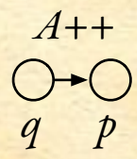
Satisfiability of XPath

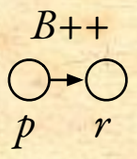
XPath: undecidable ♣

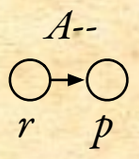
XPath without data tests: ExpTime-complete ♣

Positive-XPath: NP-complete ♣

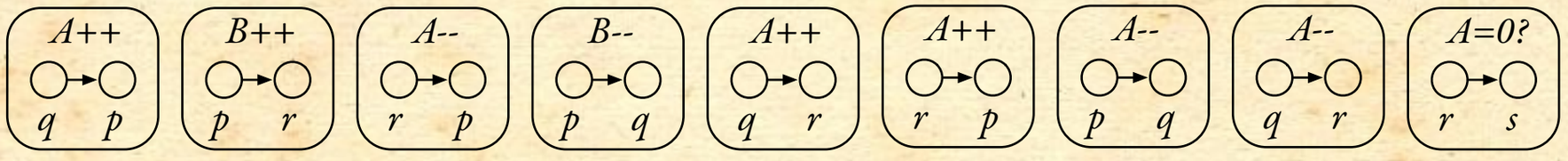
Main Idea: to use data values to *synchronize* **increments** and **decrements**.

Has associated a tree over the finite alphabet $\Sigma = \{$ 

, 

, 

, . . . $\}$



This is a forest, but the proof would be equivalent adding a root. And vice-versa.

♣ [Geerts, Fan, DBPL 2005]

Satisfiability of XPath

XPath: undecidable ♣

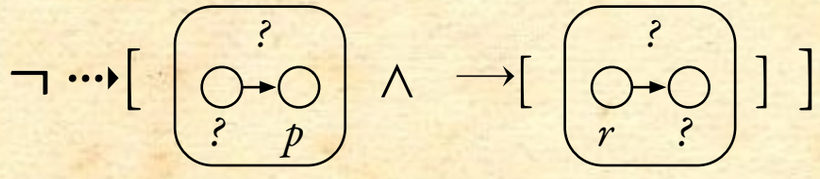
XPath without data tests: ExpTime-complete ♣

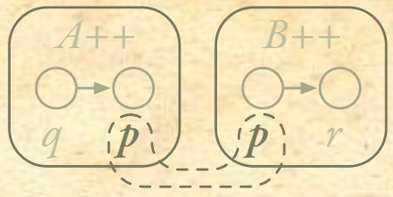
Positive-XPath: NP-complete ♣

Main Idea: to use data values to *synchronize* **increments** and **decrements**.

Conditions to test:

- Any pair of successive elements are in the 'chain' relation





...for any pair of states $p \neq r$.

- The run starts with the initial state and ends with a final state.

♣ [Geerts, Fan, DBPL 2005]

Satisfiability of XPath

XPath: undecidable♣

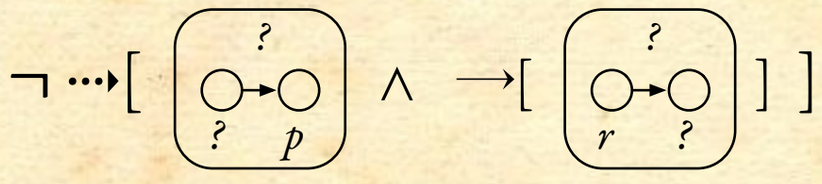
XPath without data tests: ExpTime-complete♣

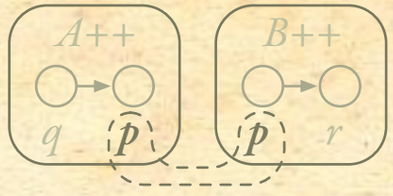
Positive-XPath: NP-complete♣

There are ***two*** kinds of illegal transitions wrt the counters:
- to decrement a counter with value 0,
- to perform a "C=0?" transition with a non-zero counter C.

Conditions to test:

❇ Any pair of successive elements are in the 'chain' relation





...for any pair of states $p \neq r$.

❇ The run starts with the initial state and ends with a final state.

♣ [Geerts, Fan, DBPL 2005]

Satisfiability of XPath

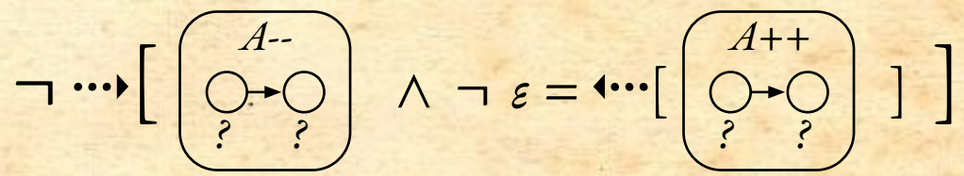
XPath: undecidable ♣

XPath without data tests: ExpTime-complete ♣

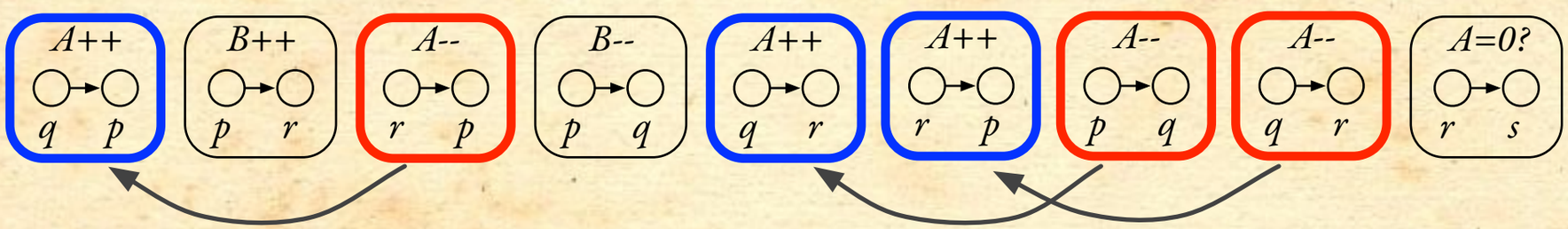
Positive-XPath: NP-complete ♣

Data values are used to **pair incrementing** and **decrementing** instructions.

Every "A--" instruction must be preceeded by a "A++" with the **same data value.**



Hence, "A--" cannot be performed if the counter value is 0.



♣ [Geerts, Fan, DBPL 2005]

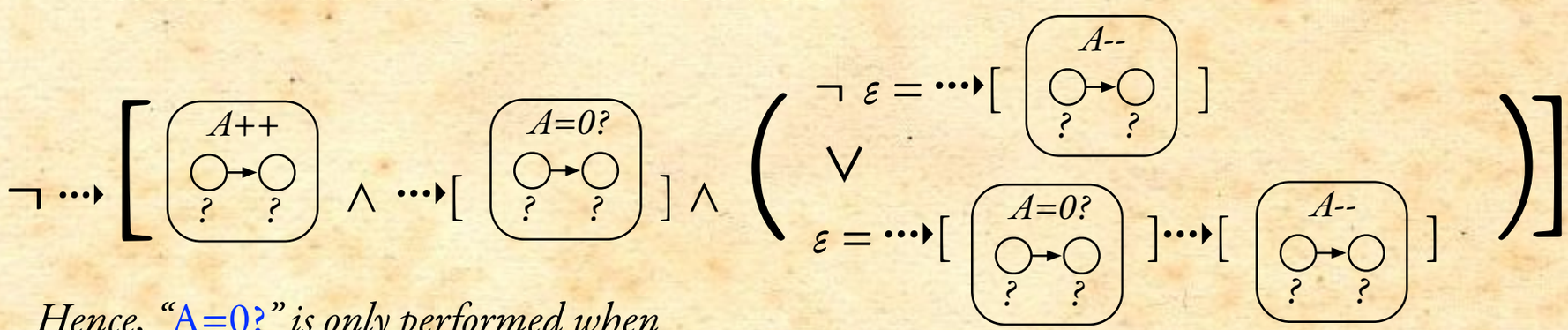
Satisfiability of XPath

XPath: undecidable♣

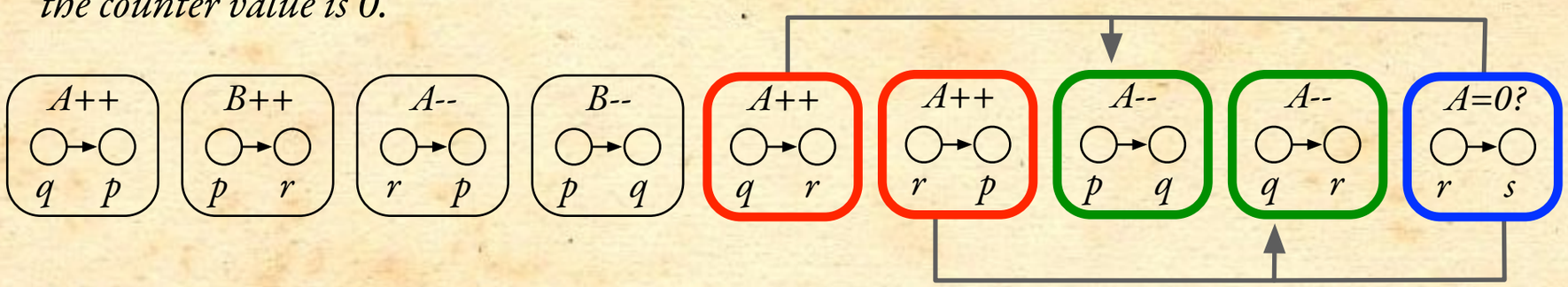
XPath without data tests: ExpTime-complete♣

Positive-XPath: NP-complete♣

❇ Every "A++" instruction that is to the left of a "A=0?" instruction, has a "A--" instruction with the same datum in between.



Hence, "A=0?" is only performed when the counter value is 0.



♣ [Geerts, Fan, DBPL 2005]

Satisfiability of XPath

NP

Positive-XPath ♠

ExpTime

Downward ♥
XPath($\downarrow, \downarrow_*$)

XPath without data tests ♠

Decidable, non-PR

Forward ♦
XPath($\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$)

Vertical ♣
XPath($\downarrow, \downarrow_*, \uparrow, \uparrow^*$)

Undecidable

full XPath ♠

♥ [F. PODS'09]
♦ [F. ICDT'10]
♣ [F., Segoufin, STACS'10]
♠ [Geerts, Fan, PODS'05]

Downward XPath

XPath(↓)	PSpace-complete ♣ ♦
XPath(↓*)	ExpTime-complete ♦
XPath(↓, ↓*)	ExpTime-complete ♦

♣ [Geerts, Fan, DBPL 2005]

♦ [Figueira, PODS 2009]

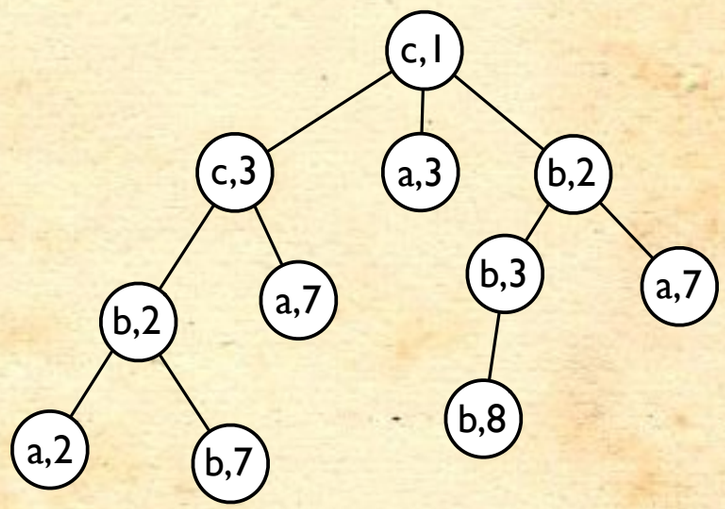
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula



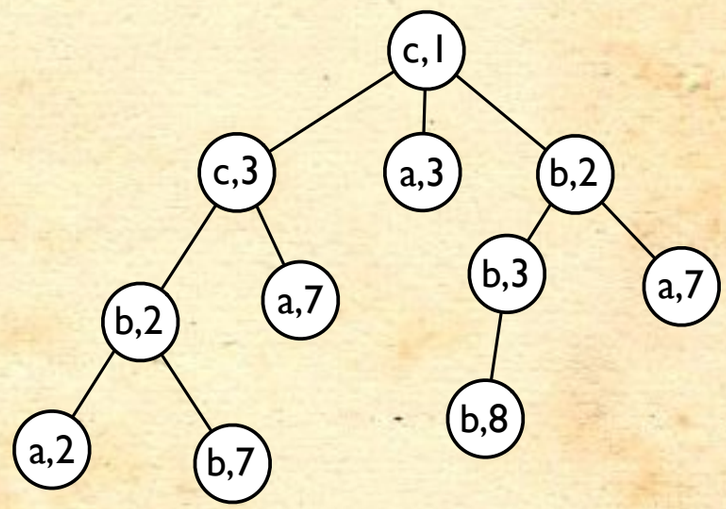
XPath($\downarrow, \downarrow_*$) ExpTime-complete

$$(1)\ \langle \alpha = \beta \rangle \quad (2)\ \langle \alpha \neq \beta \rangle \quad (3)\ \neg\langle \alpha \neq \beta \rangle \quad (4)\ \neg\langle \alpha = \beta \rangle$$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



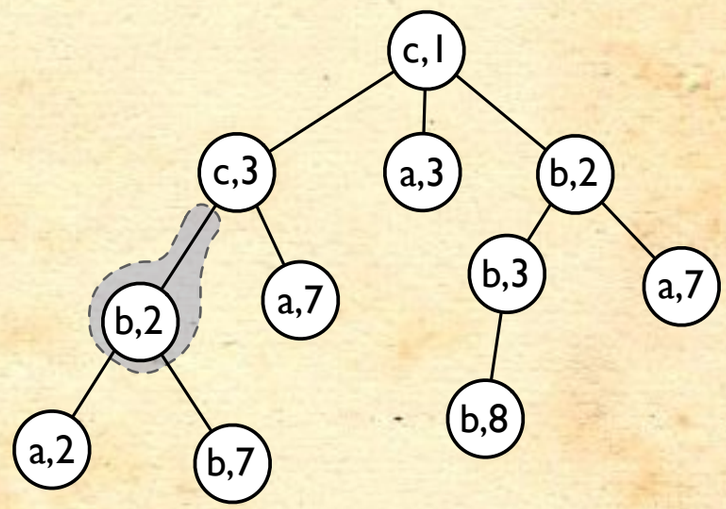
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



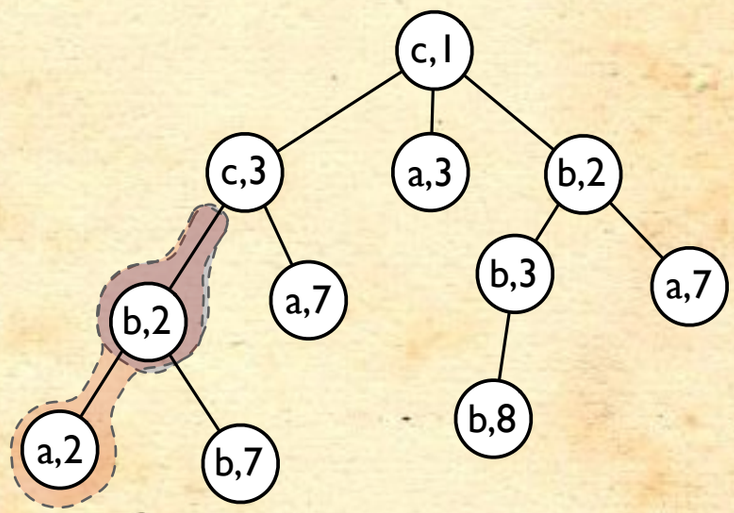
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



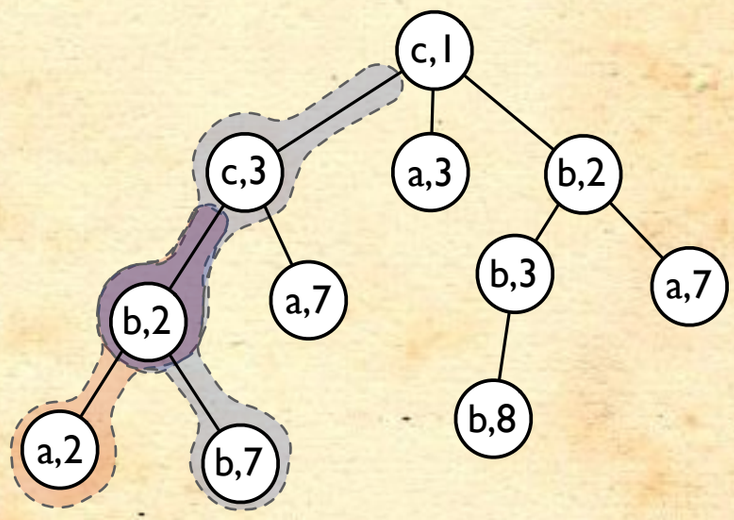
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



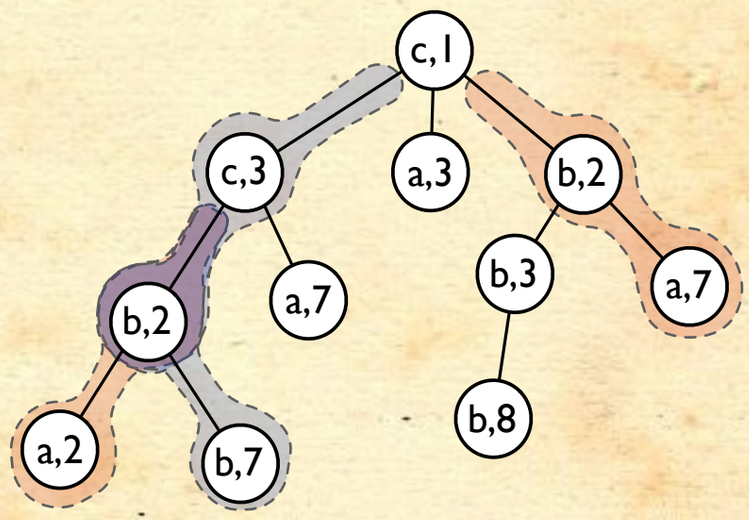
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



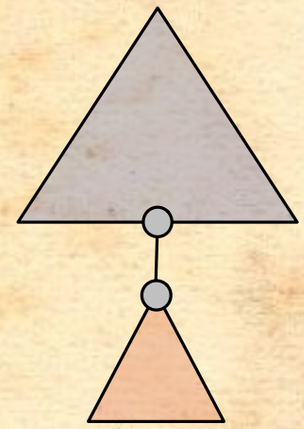
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

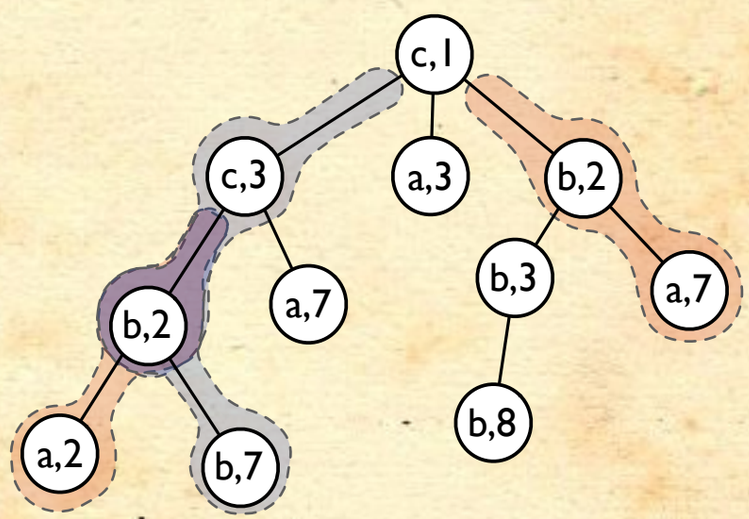
Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



closed under
duplication of subtrees



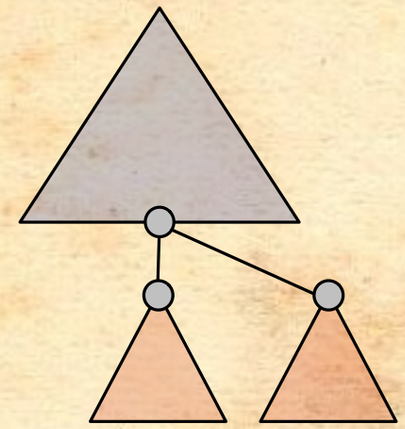
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

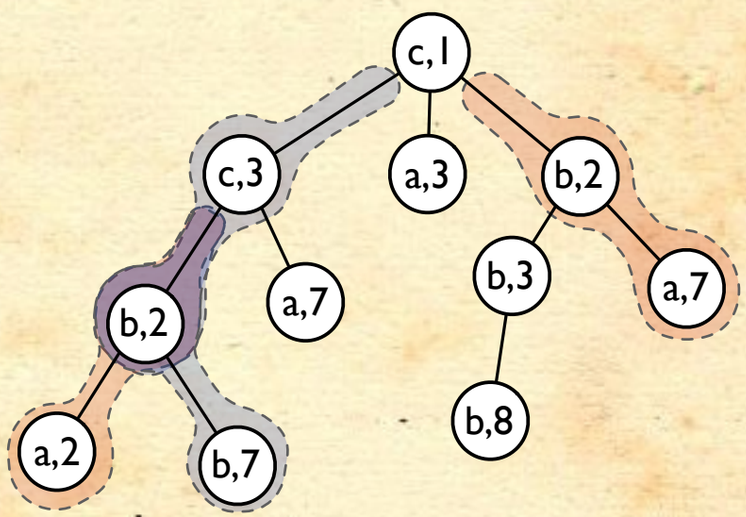
Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



closed under
duplication of subtrees



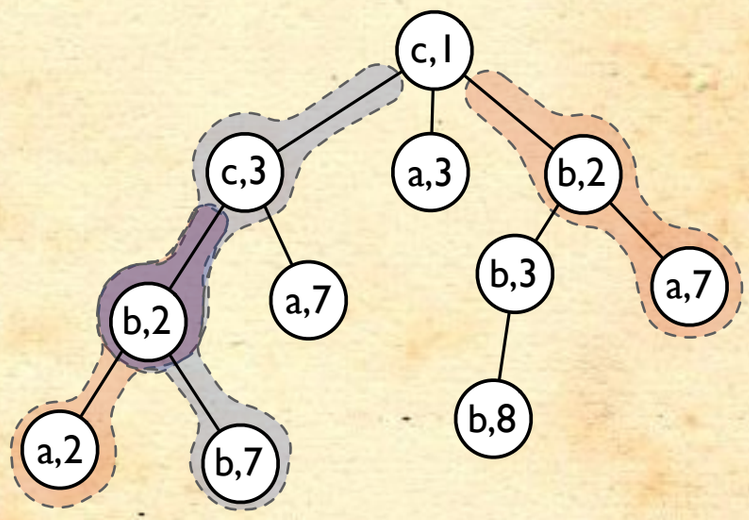
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



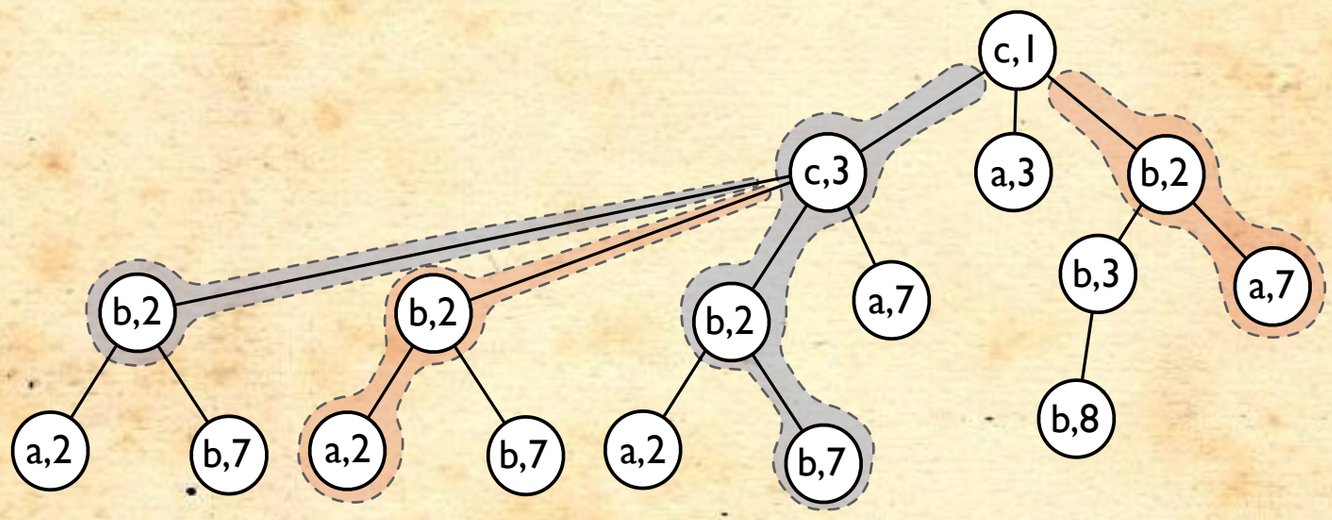
XPath($\downarrow, \downarrow_*$) ExpTime-complete

$$(1)\ \langle \alpha = \beta \rangle \quad (2)\ \langle \alpha \neq \beta \rangle \quad (3)\ \neg\langle \alpha \neq \beta \rangle \quad (4)\ \neg\langle \alpha = \beta \rangle$$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

ϕ = for every c, $\langle \downarrow_*[b] = \downarrow[b]\downarrow_*[a] \rangle$



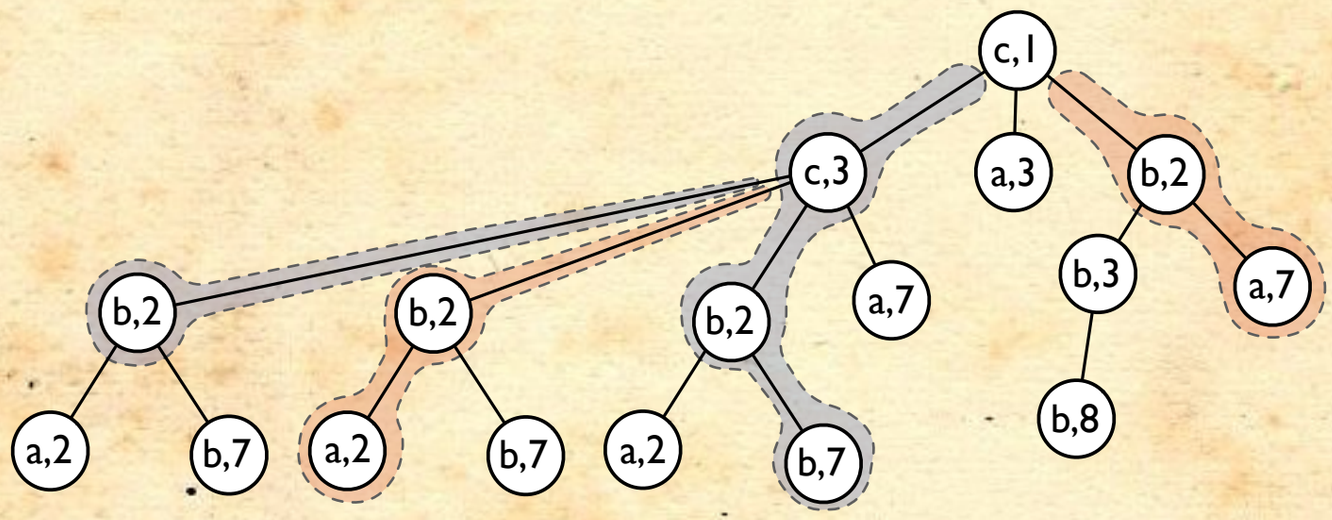
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

II) We can delete subtrees with unmarked root.



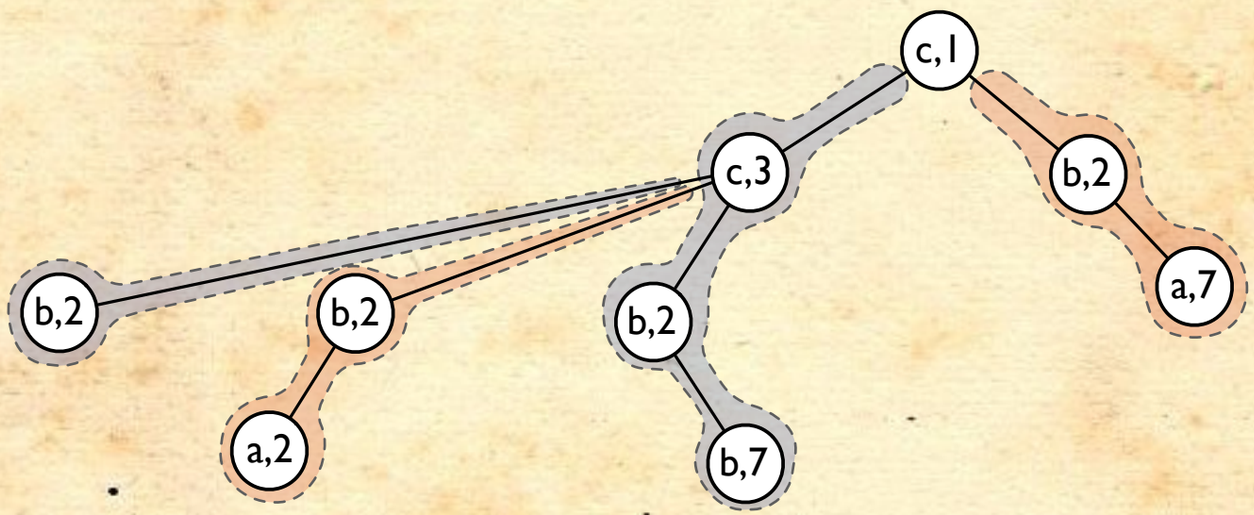
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

II) We can delete subtrees with unmarked root.



XPath($\downarrow, \downarrow_*$) ExpTime-complete

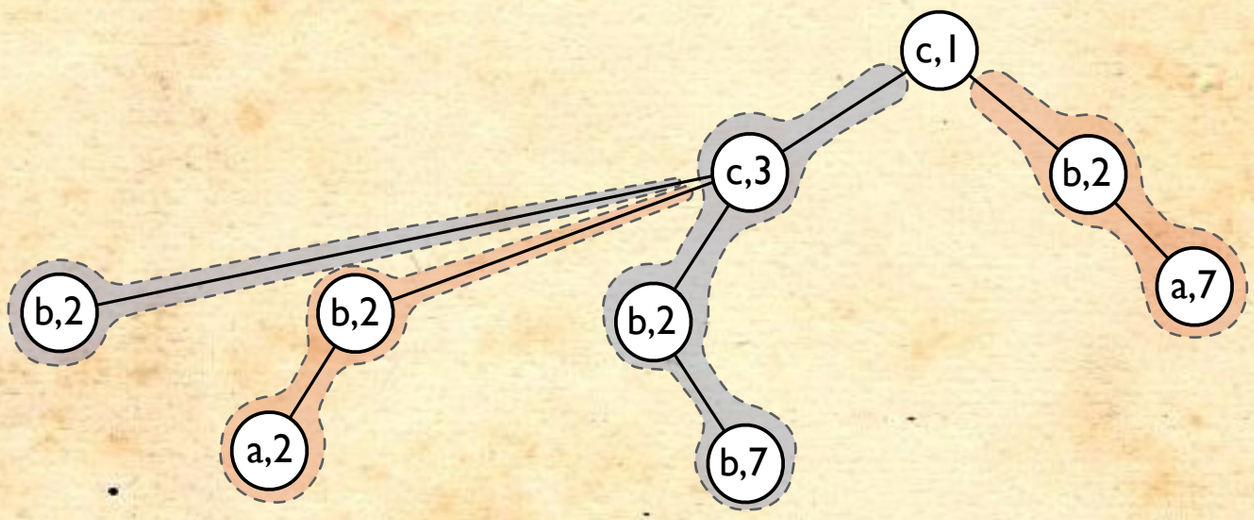
(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

II) We can delete subtrees with unmarked root.

Markings always disjoint.



XPath($\downarrow, \downarrow_*$) ExpTime-complete

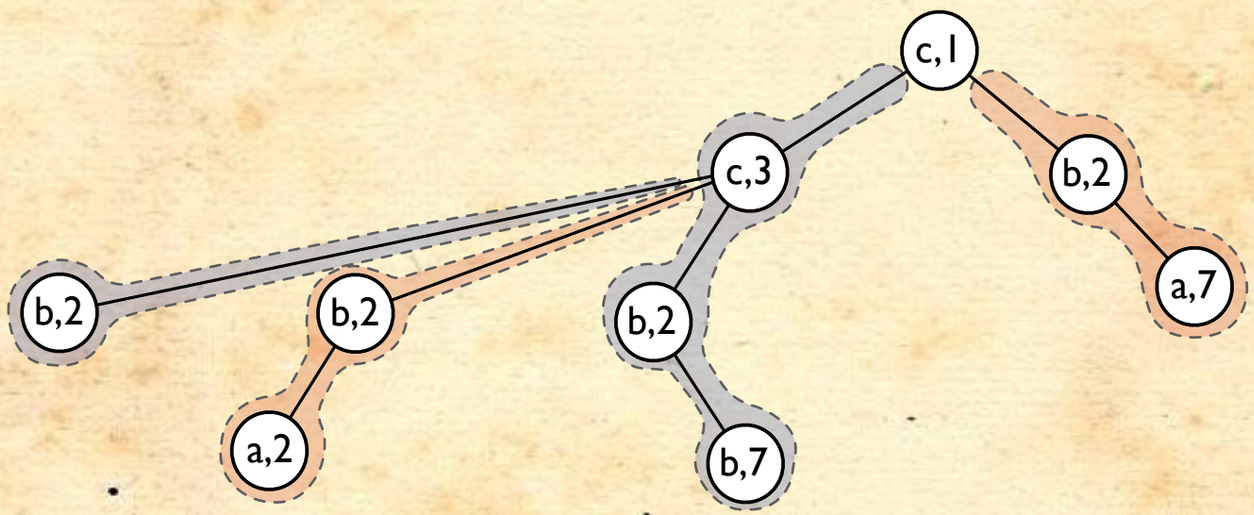
$$(1)\ \langle \alpha = \beta \rangle \quad (2)\ \langle \alpha \neq \beta \rangle \quad (3)\ \neg\langle \alpha \neq \beta \rangle \quad (4)\ \neg\langle \alpha = \beta \rangle$$

Take a tree satisfying a formula

I) For (1) and (2) we can find witnessing paths that we mark.

II) We can delete subtrees with unmarked root.

Markings always disjoint. Poly-width model property.



XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Markings always disjoint.
Poly-width model property.

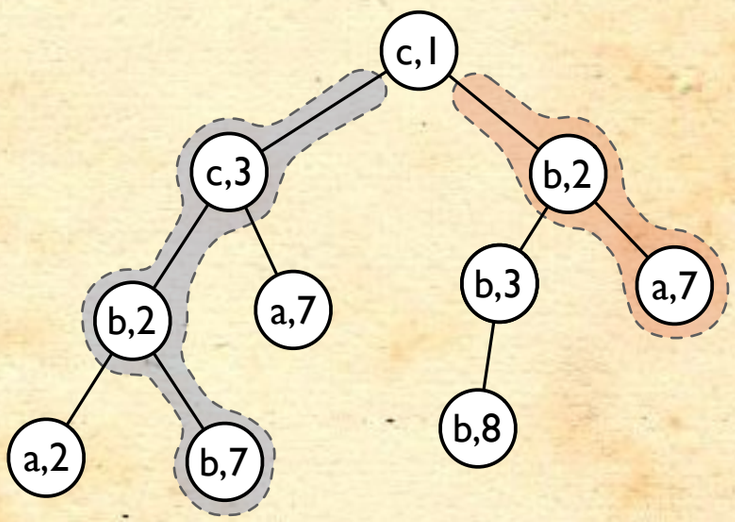
XPath($\downarrow, \downarrow_*$) ExpTime-complete

$$(1)\ \langle \alpha = \beta \rangle \quad (2)\ \langle \alpha \neq \beta \rangle \quad (3)\ \neg\langle \alpha \neq \beta \rangle \quad (4)\ \neg\langle \alpha = \beta \rangle$$

Markings always disjoint.
Poly-width model property.

III) For every node: at most one witnessing data value for each type (3) formula.
(Polynomially many.)

$\neg\langle \downarrow[c]\downarrow[b] \neq \downarrow[b] \rangle$



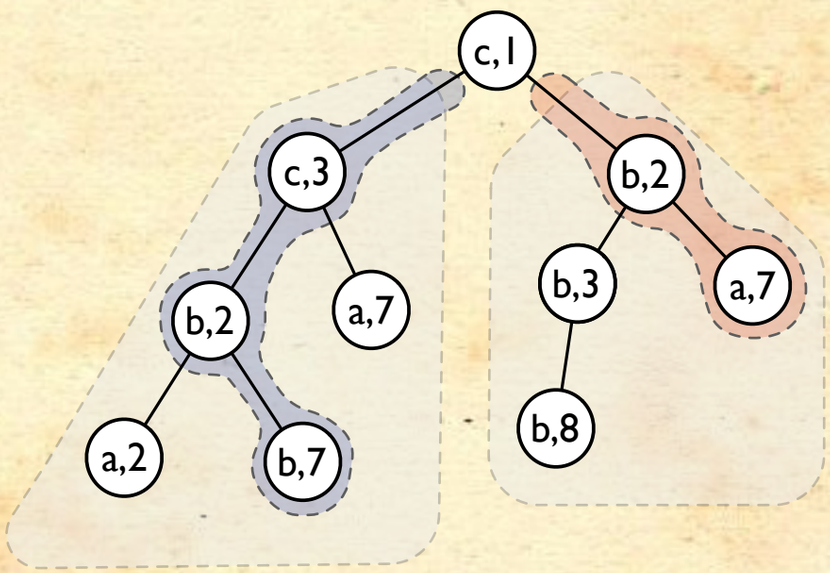
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Markings always disjoint.
Poly-width model property.

III) For every node: at most one witnessing data value for each type (3) formula.
(Polynomially many.)

$\neg\langle \downarrow[c]\downarrow[b] \neq \downarrow[b] \rangle$



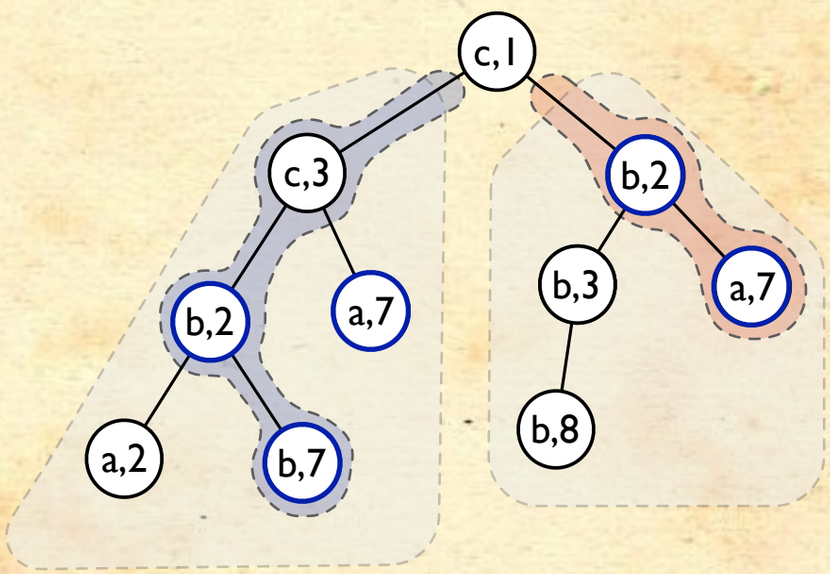
XPath($\downarrow, \downarrow_*$) ExpTime-complete

$$(1)\ \langle \alpha = \beta \rangle \quad (2)\ \langle \alpha \neq \beta \rangle \quad (3)\ \neg\langle \alpha \neq \beta \rangle \quad (4)\ \neg\langle \alpha = \beta \rangle$$

Markings always disjoint.
Poly-width model property.

III) For every node: at most one witnessing data value for each type (3) formula.
(Polynomially many.)

$\neg\langle \downarrow[c]\downarrow[b] \neq \downarrow[b] \rangle$



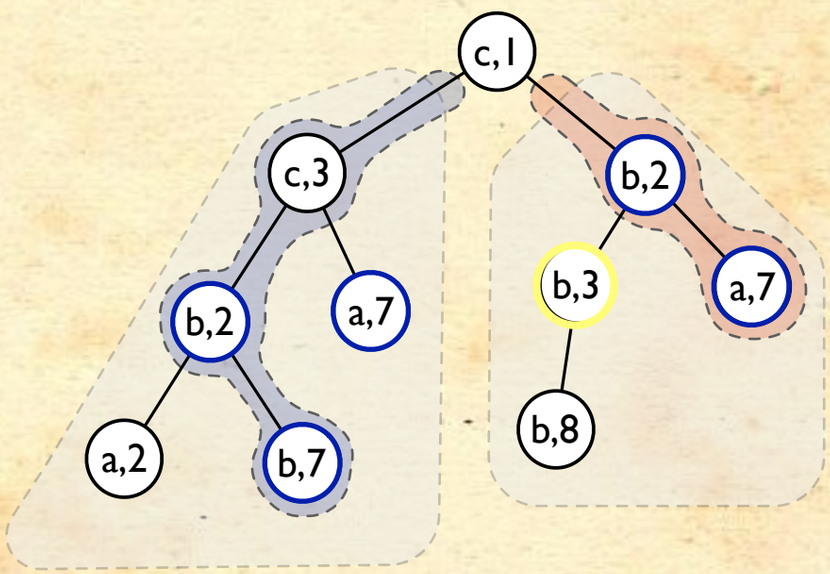
XPath($\downarrow, \downarrow_*$) ExpTime-complete

$$(1)\ \langle \alpha = \beta \rangle \quad (2)\ \langle \alpha \neq \beta \rangle \quad (3)\ \neg\langle \alpha \neq \beta \rangle \quad (4)\ \neg\langle \alpha = \beta \rangle$$

Markings always disjoint.
Poly-width model property.

III) For every node: at most one witnessing data value for each type (3) formula.
(Polynomially many.)

$\neg\langle \downarrow[c]\downarrow[b] \neq \downarrow[b] \rangle$



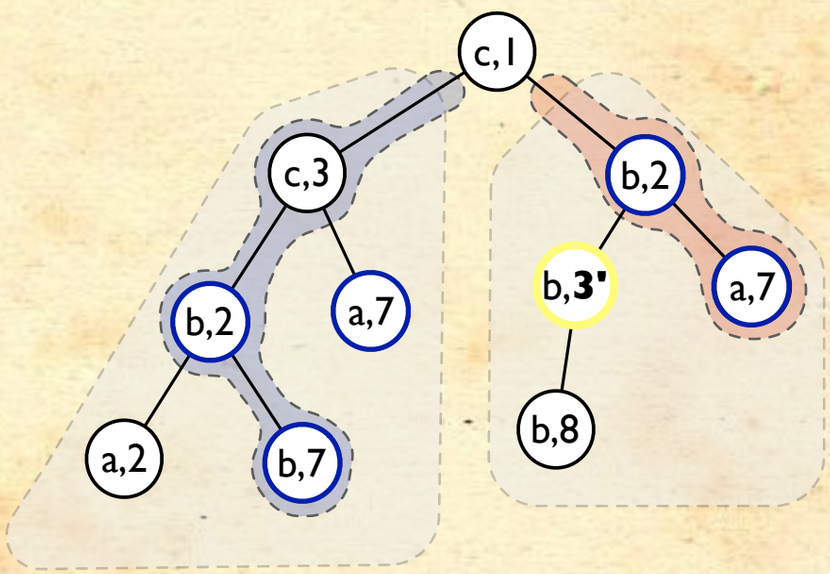
XPath($\downarrow, \downarrow_*$) ExpTime-complete

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Markings always disjoint.
Poly-width model property.

III) For every node: at most one witnessing data value for each type (3) formula.
(Polynomially many.)

$\neg\langle \downarrow[c]\downarrow[b] \neq \downarrow[b] \rangle$



XPath($\downarrow, \downarrow_*$) ExpTime-complete

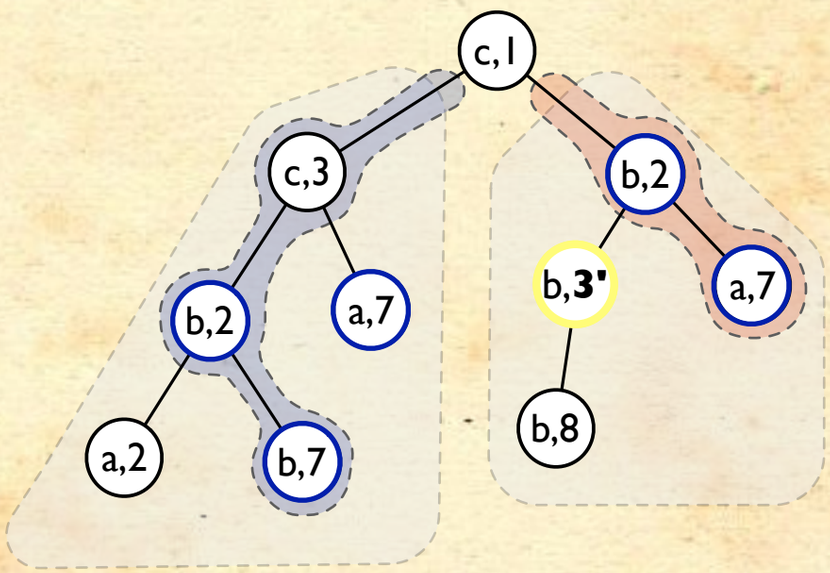
$$(1)\ \langle \alpha = \beta \rangle \quad (2)\ \langle \alpha \neq \beta \rangle \quad (3)\ \neg\langle \alpha \neq \beta \rangle \quad (4)\ \neg\langle \alpha = \beta \rangle$$

Markings always disjoint.
Poly-width model property.

III) For every node: at most one witnessing data value for each type (3) formula.
(Polynomially many.)

IV) Every two sibling subtrees share only a polynomial number of dv's.

$\neg\langle \downarrow[c]\downarrow[b] \neq \downarrow[b] \rangle$



XPath($\downarrow, \downarrow_*$) ExpTime-complete

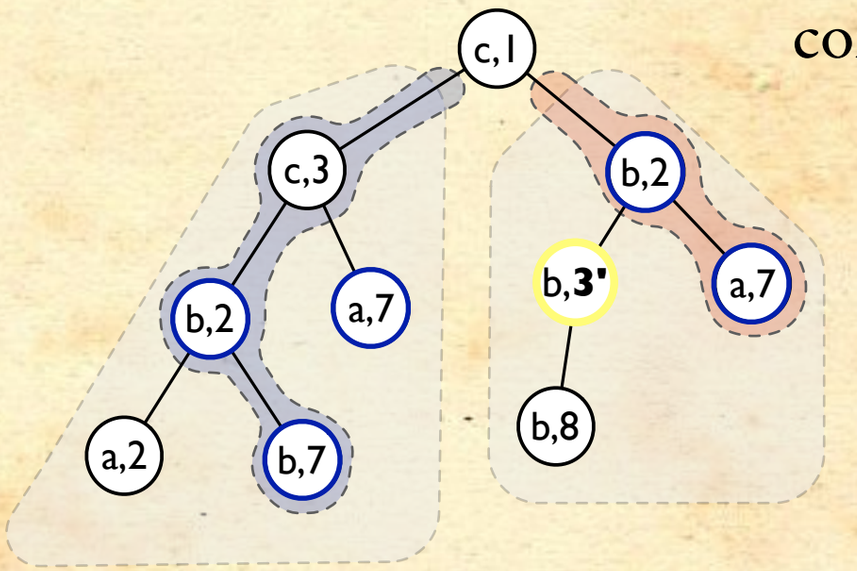
(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha \neq \beta \rangle$ (4) $\neg\langle \alpha = \beta \rangle$

Markings always disjoint.
Poly-width model property.

III) For every node: at most one witnessing data value for each type (3) formula.
(Polynomially many.)

IV) Every two sibling subtrees share only a polynomial number of dv's.

$\neg\langle \downarrow[c]\downarrow[b] \neq \downarrow[b] \rangle$



V) Using this, we devise a bottom-up algorithm to compute all possible types.

Downward XPath

Key properties:

- closure of duplication of subtrees
- absence of horizontal navigation
- unranked trees

Satisfiability of XPath

NP

Positive-XPath ♠

ExpTime

Downward ♥
XPath($\downarrow, \downarrow_*$)

XPath without data tests ♠

Decidable, non-PR

Forward ♦
XPath($\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$)

Vertical ♣
XPath($\downarrow, \downarrow_*, \uparrow, \uparrow^*$)

Undecidable

full XPath ♠

♥ [F. PODS'09]
♦ [F. ICDT'10]
♣ [F., Segoufin, STACS'10]
♠ [Geerts, Fan, PODS'05]

Forward XPath

$\mathsf{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ is decidable, non-primitive recursive♣ ♦

♦ [Jurdziński, Lazić, LICS 2007]

♣ [F, ICDT 2010]

Forward XPath

$\mathsf{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ is decidable, non-primitive recursive ♣ ♦

+ unary key constraints : decidable
+ regular languages / DTDs : decidable
+ unary foreign key constraints : undecidable

♦ [Jurdziński, Lazić, LICS 2007]

♣ [F, ICDT 2010]

Forward XPath

$\mathsf{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ is a

$\neg \langle \downarrow_* [c \wedge \langle \varepsilon = \downarrow\downarrow_*[c] \rangle] \rangle$

$\wedge \neg \langle \downarrow [\langle \downarrow_*[c] = \longrightarrow\longrightarrow^*\downarrow_*[c] \rangle] \rangle$

+ unary key constraints : decidable
+ regular languages / DTDs : decidable
+ unary foreign key constraints : undecidable

♦ [Jurdziński, Lazić, LICS 2007]

♣ [F, ICDT 2010]

Forward XPath

$\mathsf{XPath}(\downarrow, \downarrow_*, \rightarrow, \rightarrow^*)$ is decidable, non-primitive recursive ♣ ♦

+ unary key constraints : decidable
+ regular languages / DTDs : decidable
+ unary foreign key constraints : undecidable

♦ [Jurdziński, Lazić, LICS 2007]

♣ [F, ICDT 2010]

ATRA: Alternating Tree Register Automata

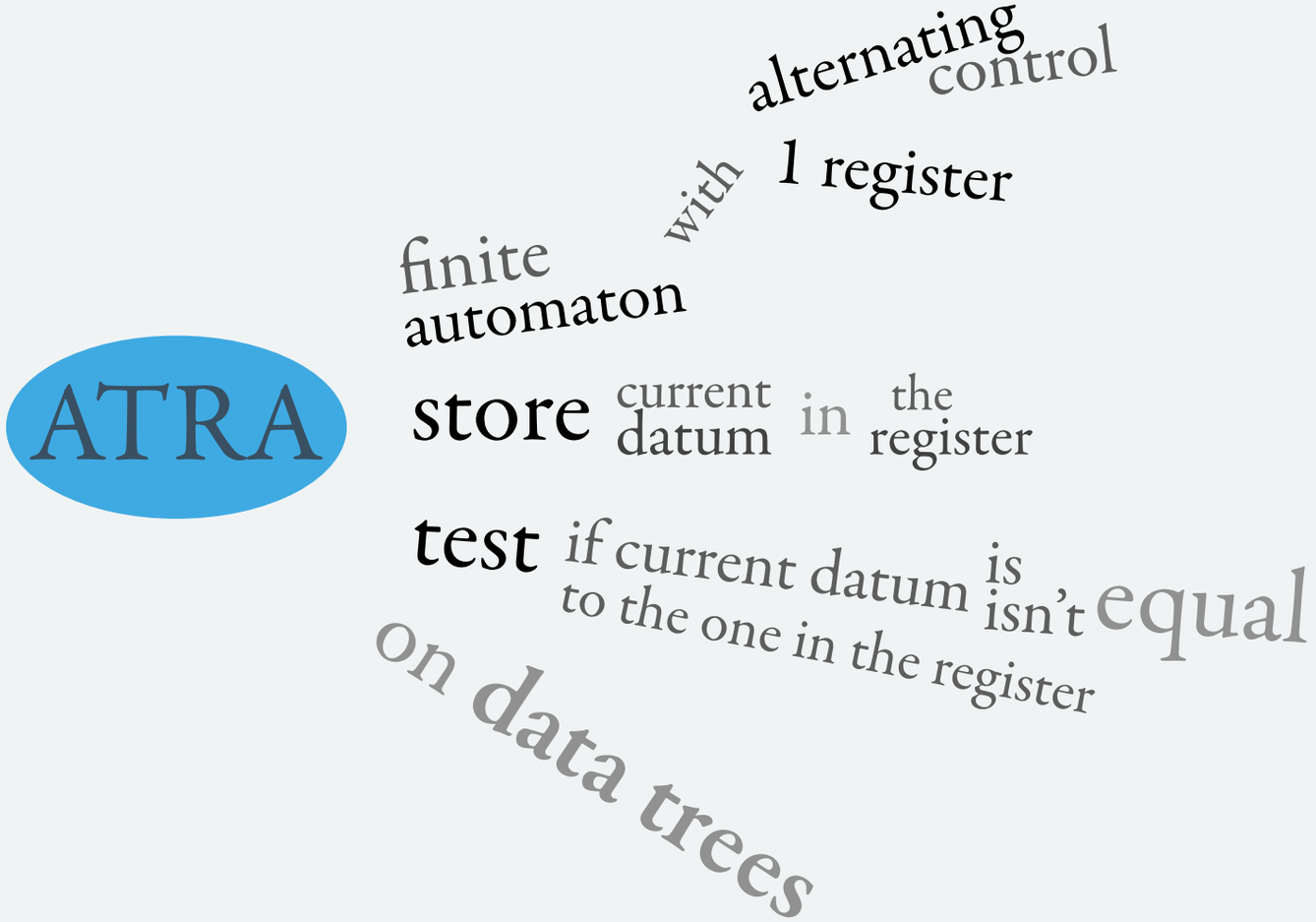
XPath ⇝ Alternating automata with one"register"

ATRA: Alternating Tree Register Automata

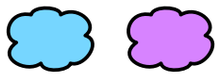
XPath ⇝ Alternating automata with one"register"

ATRA: Alternating Tree Register Automata

XPath ⇝ Alternating automata with one"register"



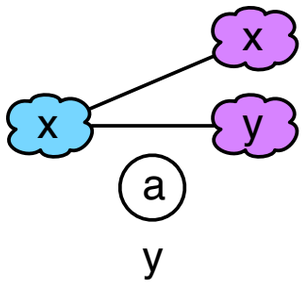
ARA

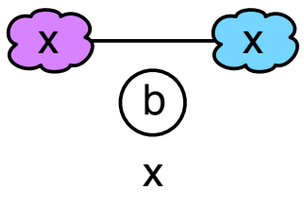
states 

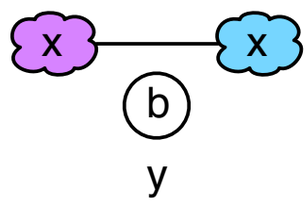
initial state 

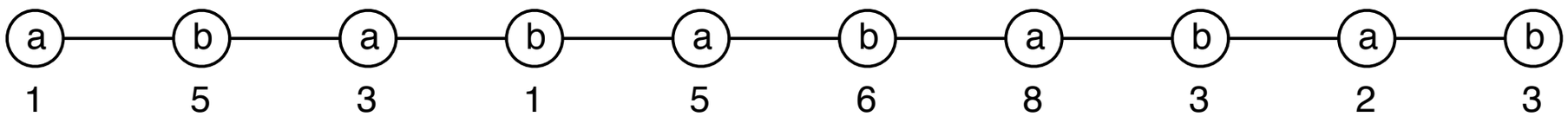
final states 

transitions

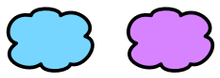








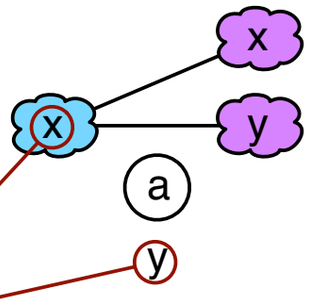
ARA

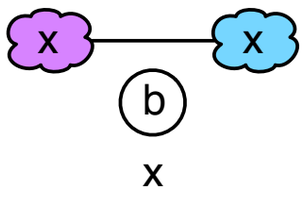
states 

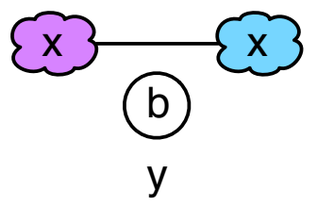
initial state 

final states 

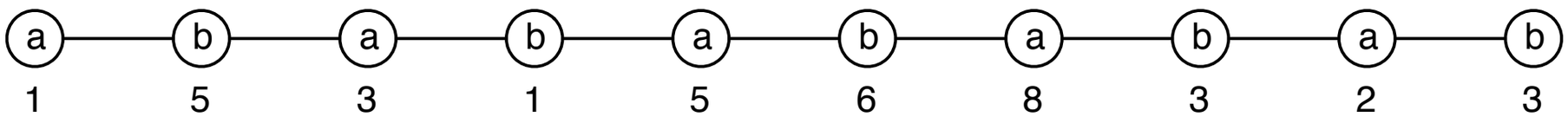
transitions



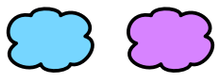




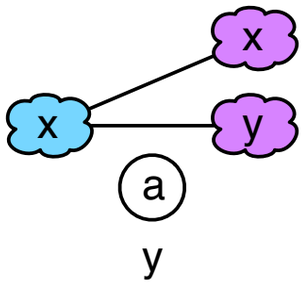
data values

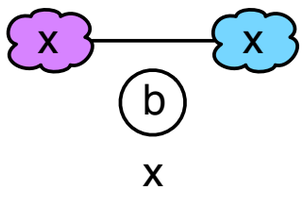


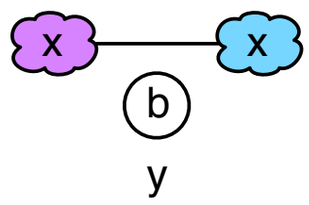
ARA

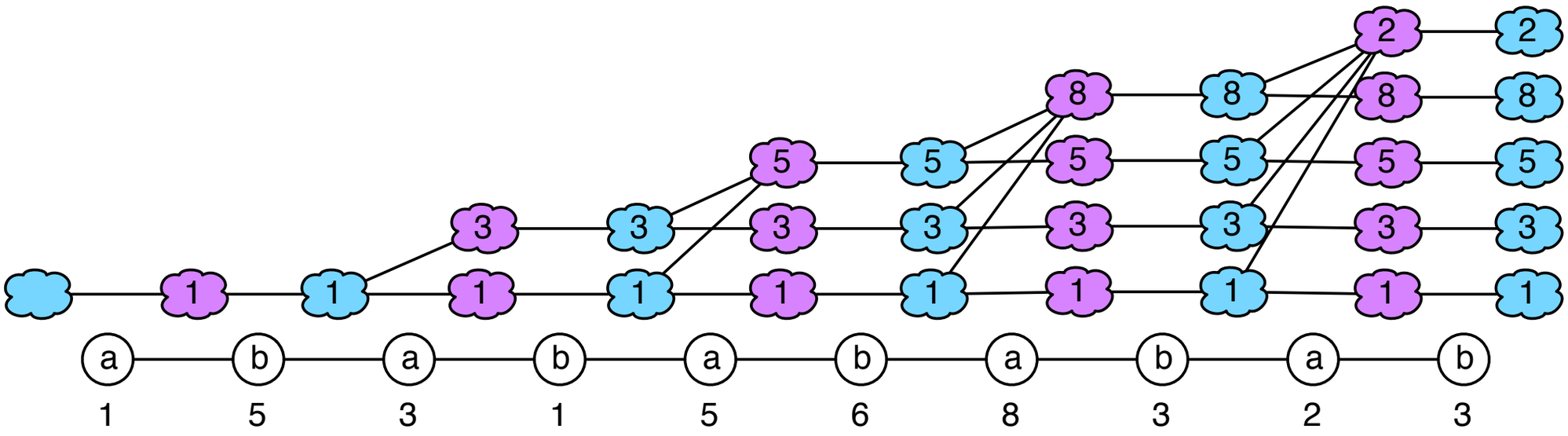
states  initial state  final states 

transitions



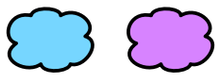




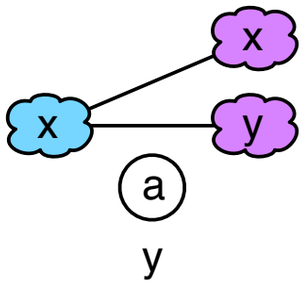


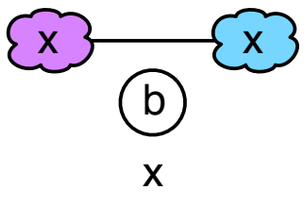
the string is *(ab)**, and all the *a*'s have different data values

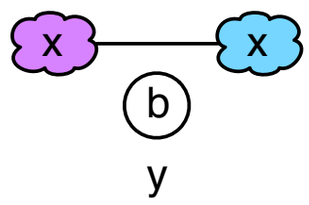
ARA

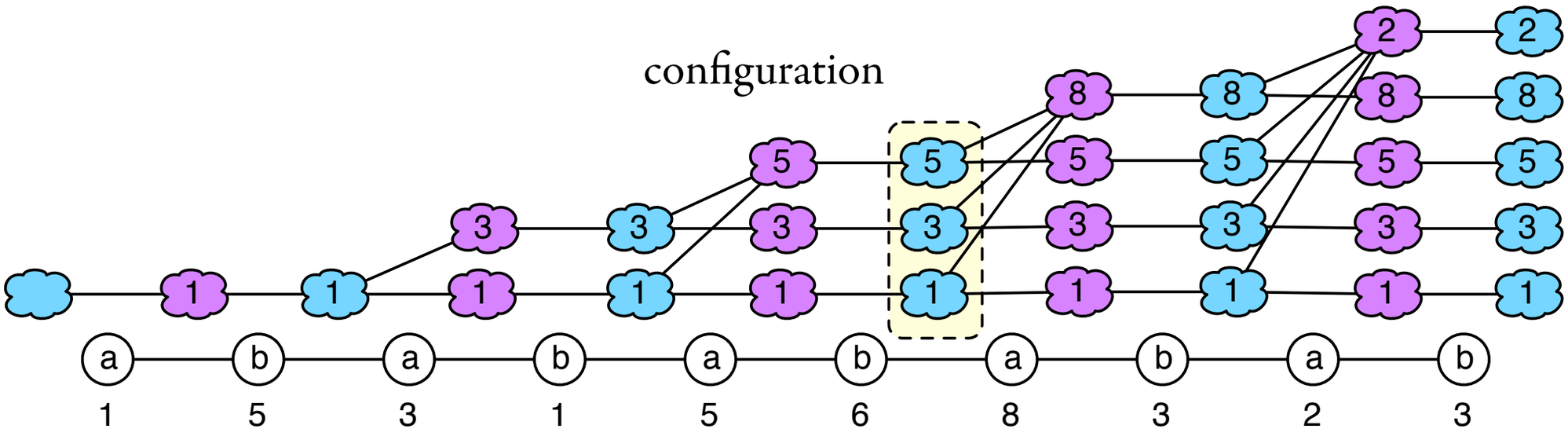
states  initial state  final states 

transitions



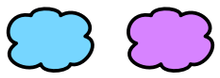






the string is *(ab)**, and all the *a*'s have different data values

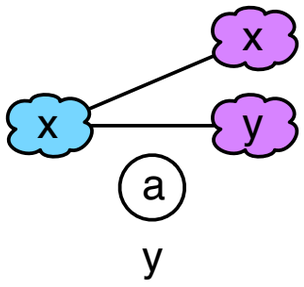
ARA

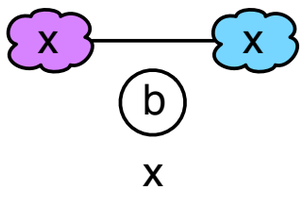
states 

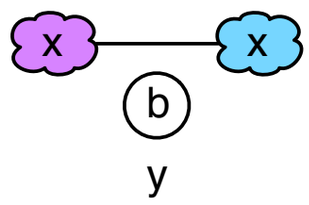
initial state 

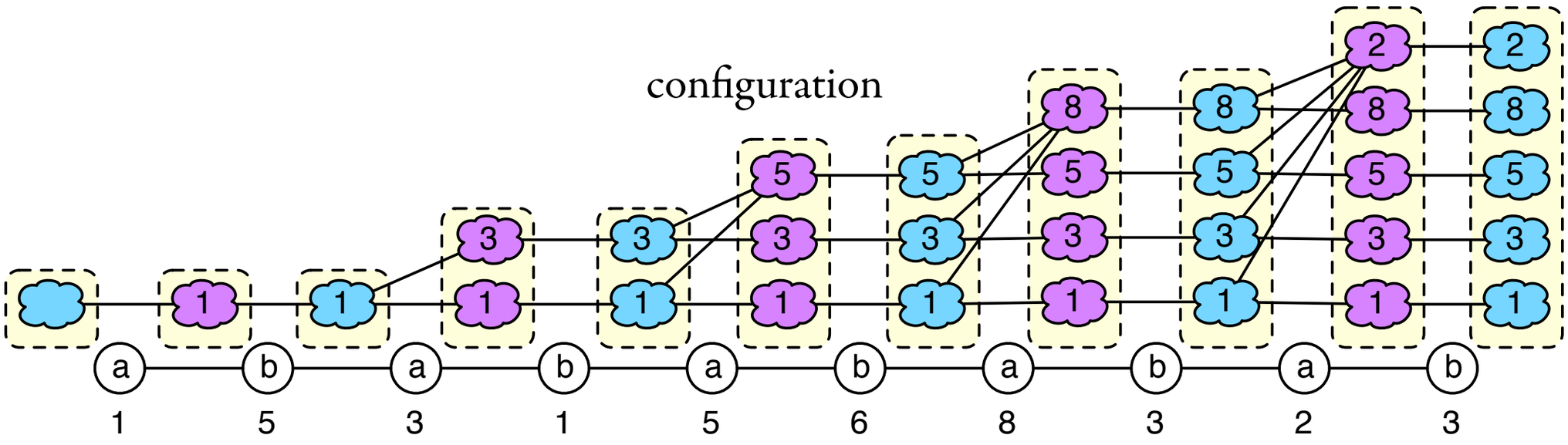
final states 

transitions



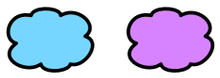






the string is *(ab)**, and all the *a*'s have different data values

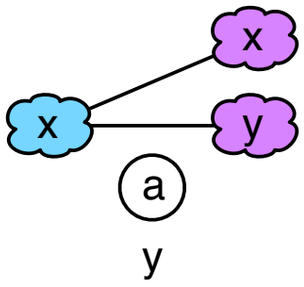
ARA

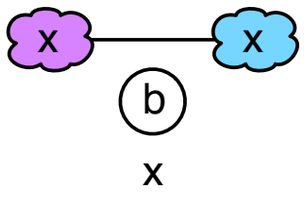
states 

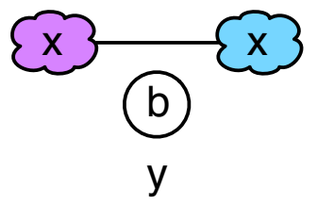
initial state 

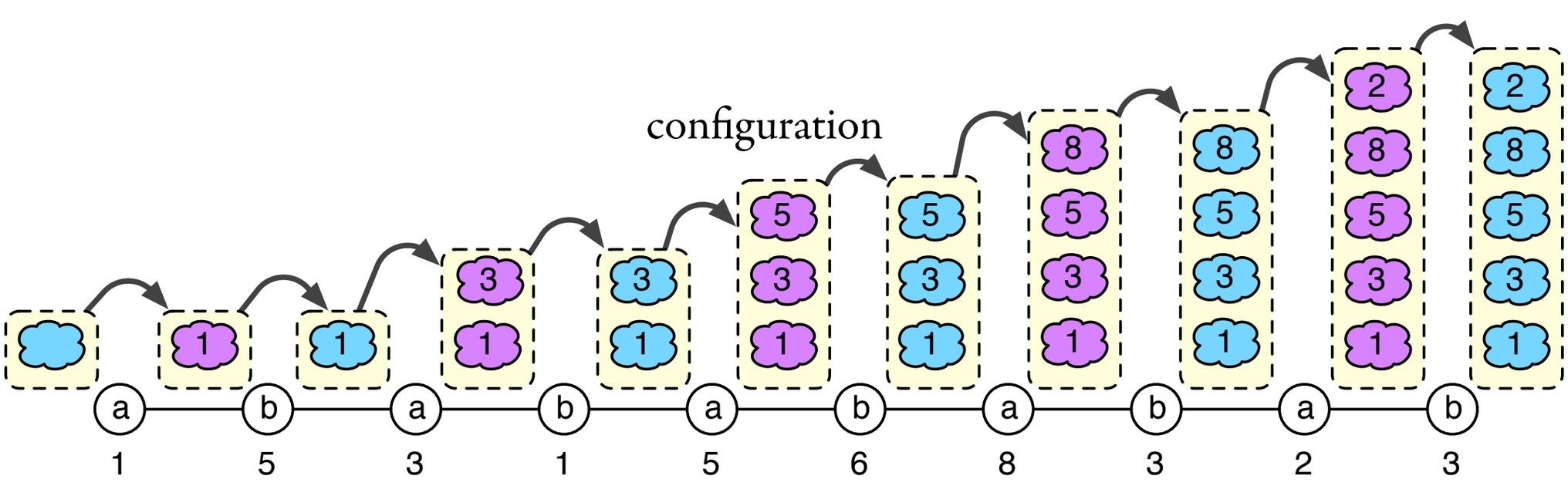
final states 

transitions









the string is *(ab)**, and all the *a*'s have different data values

ARA

☺ Decidable emptiness problem

☹ With non-primitive recursive complexity

☺ Closed under complementation, intersection, union

Proof of decidability for ARA

via theory of well quasi-orderings

quasi-order between configurations ≤

a ≤ **b** : "**b** is *more difficult* to verify than **a**"

b leads to a final configuration ⇨ **a** leads to a final configuration

Proof of decidability for ARA

via theory of well quasi-orderings

quasi-order between configurations ≤

a ≤ **b** : "**b** is *more difficult* to verify than **a**"

b leads to a final configuration ⇨ **a** leads to a final configuration

nice properties of ≤ (wqo)

○ no
○ infinite
○ decreasing
○ sequences
⋮

no infinite antichains
○○○○○ · · · ·

Proof of decidability for ARA

via theory of well quasi-orderings

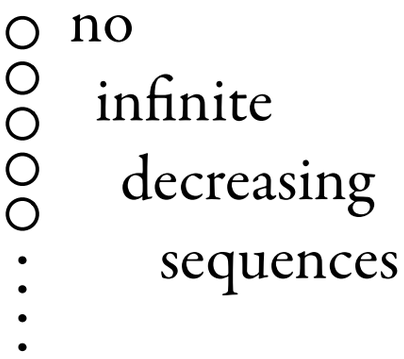
quasi-order between configurations ≤

a ≤ **b** : "**b** is *more difficult* to verify than **a**"

b leads to a final configuration ⇨ **a** leads to a final configuration

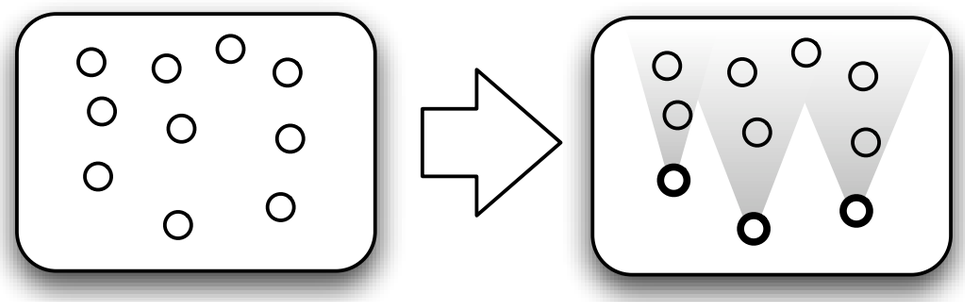
nice properties of ≤ (wqo)

no infinite decreasing sequences



no infinite antichains

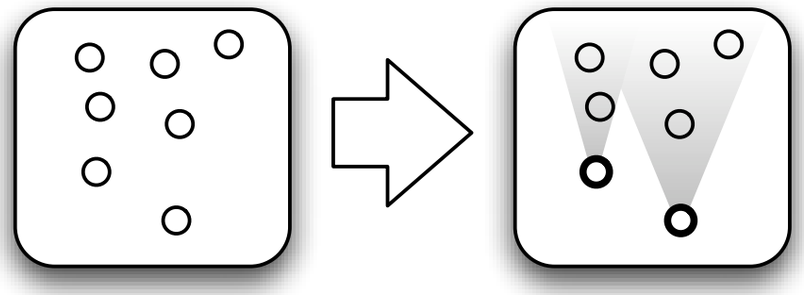
only a **finite** number of minimally 'easy' configurations



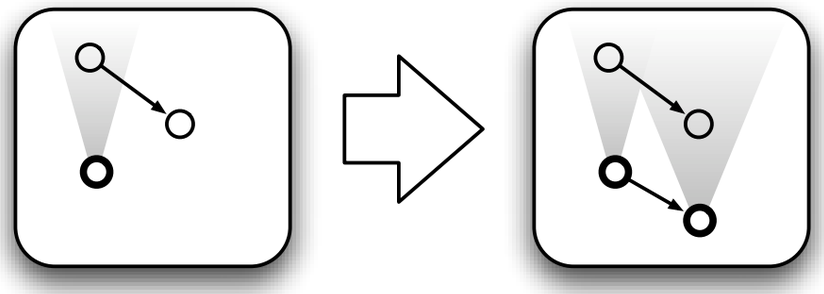
Recipe

The ingredients

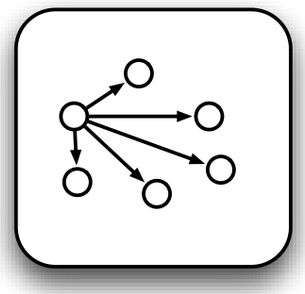
well quasi-order



well-structured



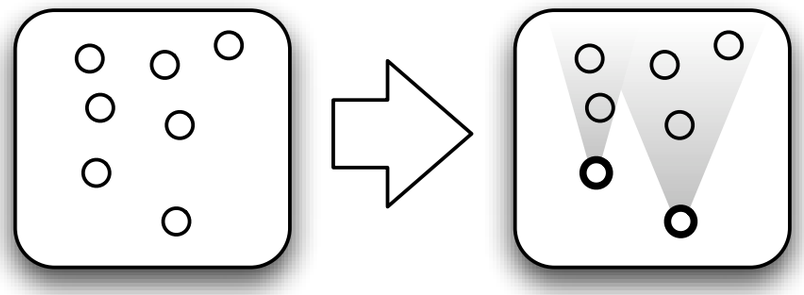
recursive set of successors



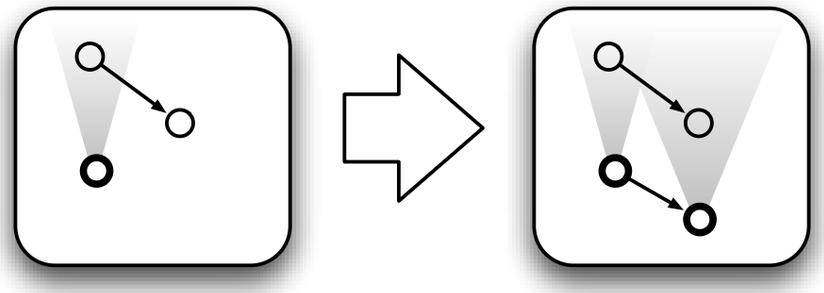
Recipe

The ingredients

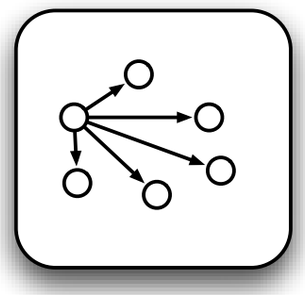
well quasi-order



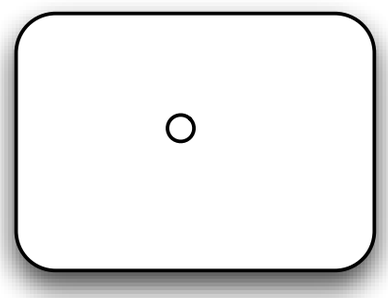
well-structured



recursive set of successors



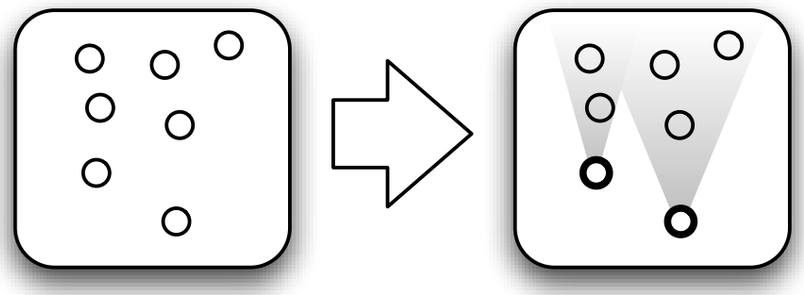
The algoritm



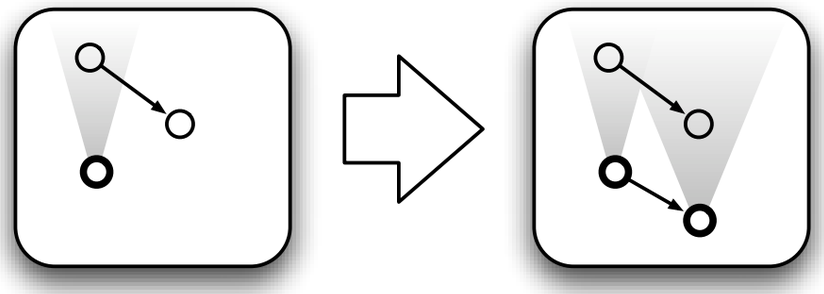
Recipe

The ingredients

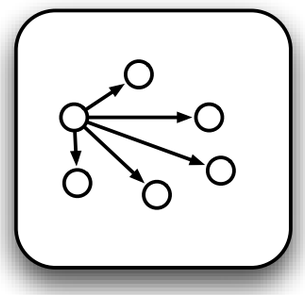
well quasi-order



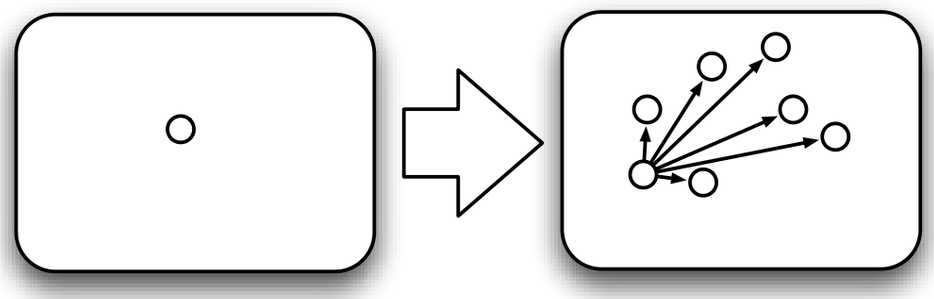
well-structured



recursive set of successors



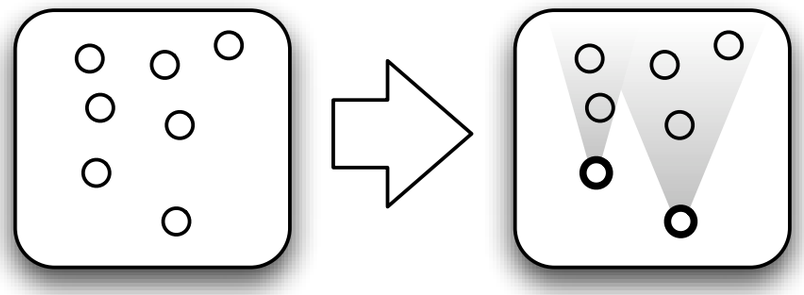
The algoritm



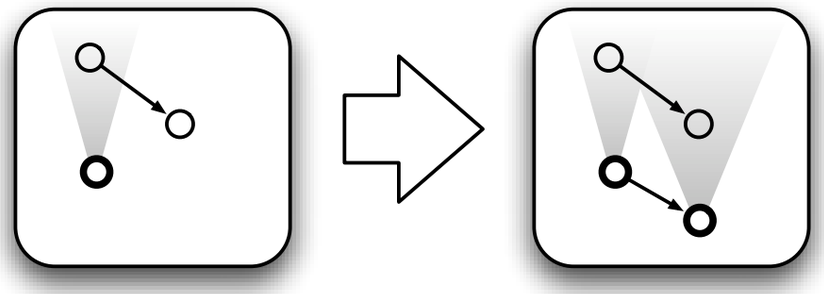
Recipe

The ingredients

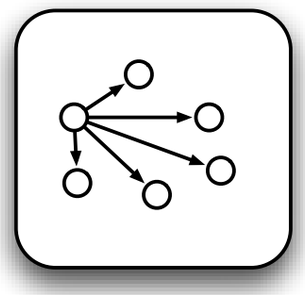
well quasi-order



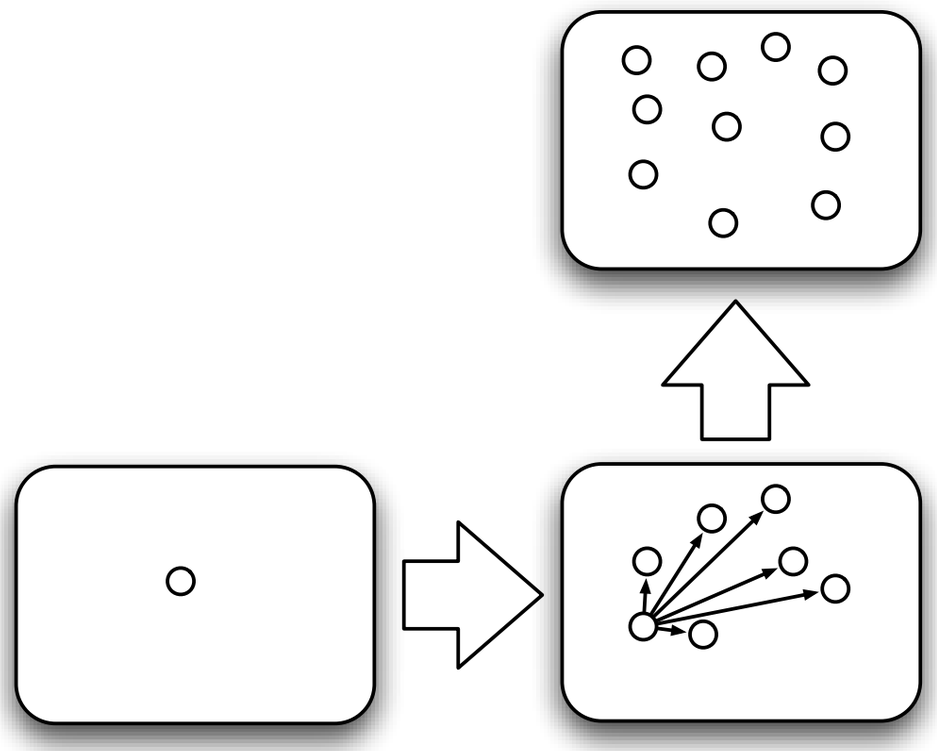
well-structured



recursive set of successors



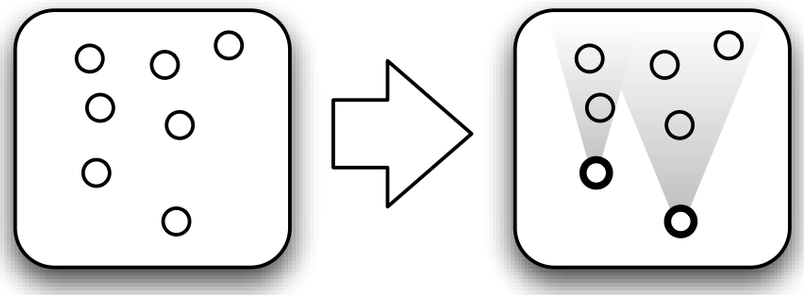
The algoritm



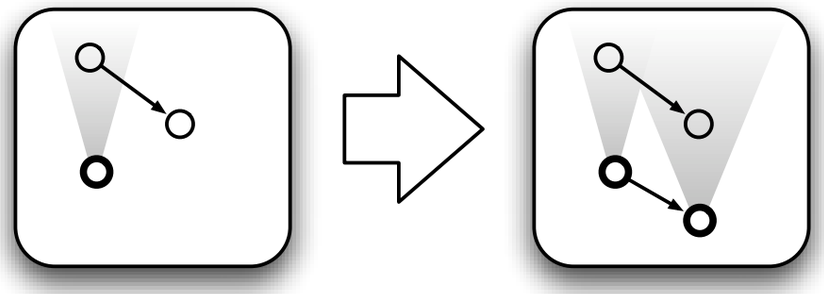
Recipe

The ingredients

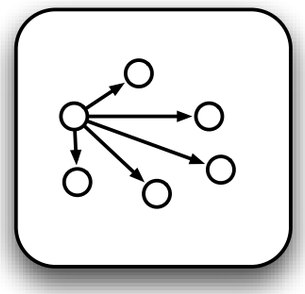
well quasi-order



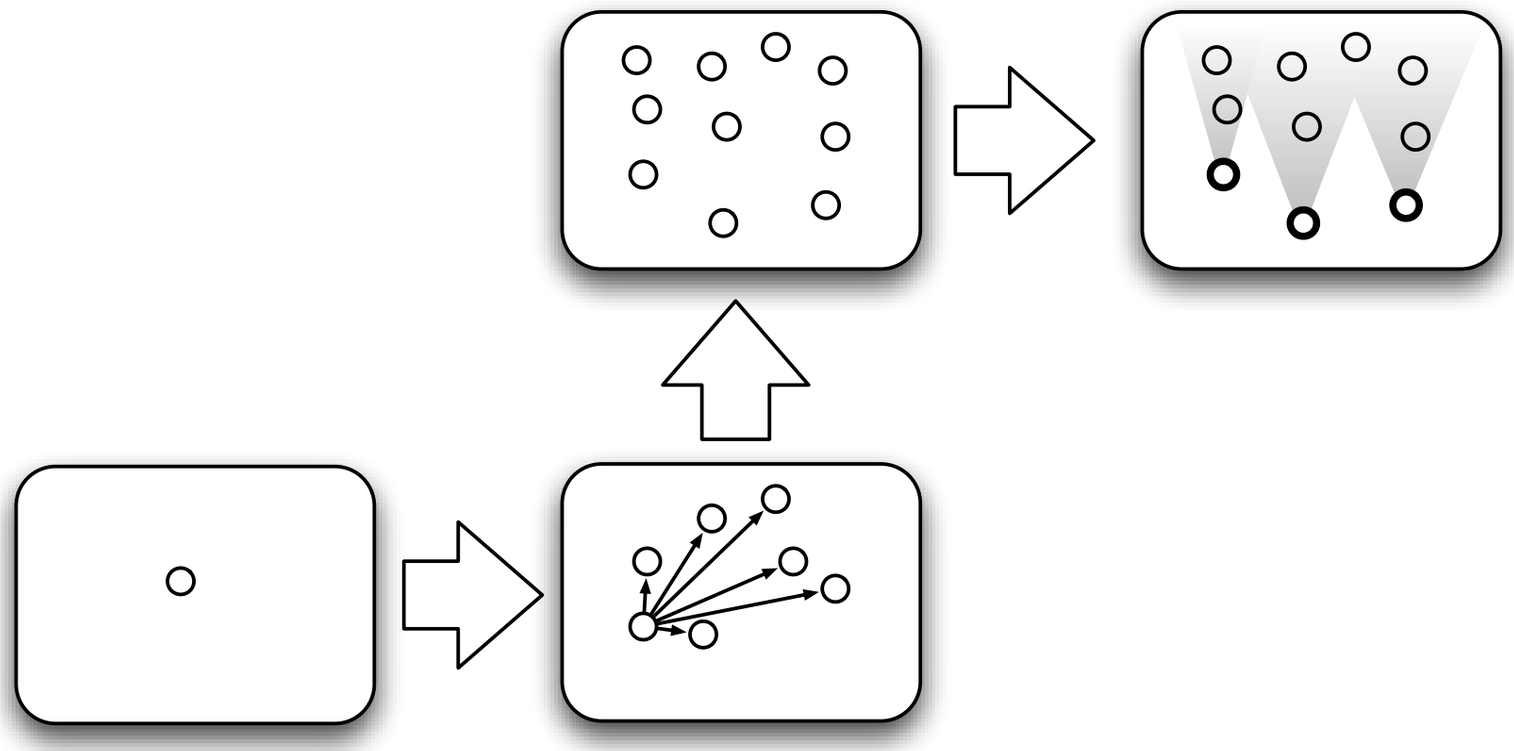
well-structured



recursive set of successors



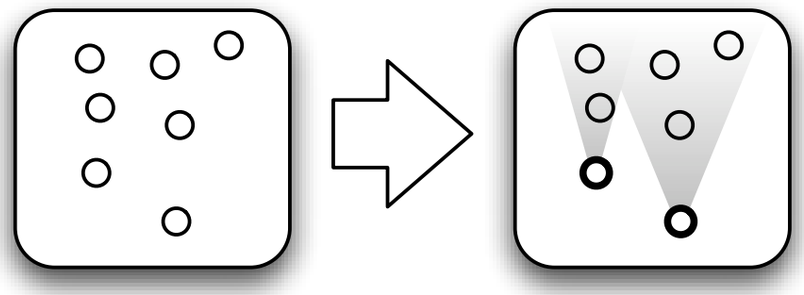
The algoritm



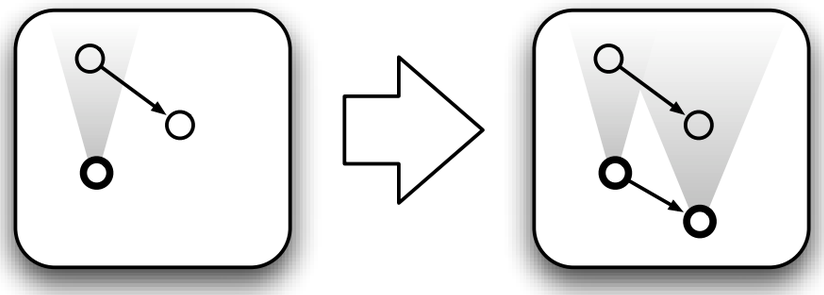
Recipe

The ingredients

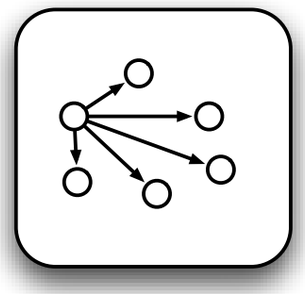
well quasi-order



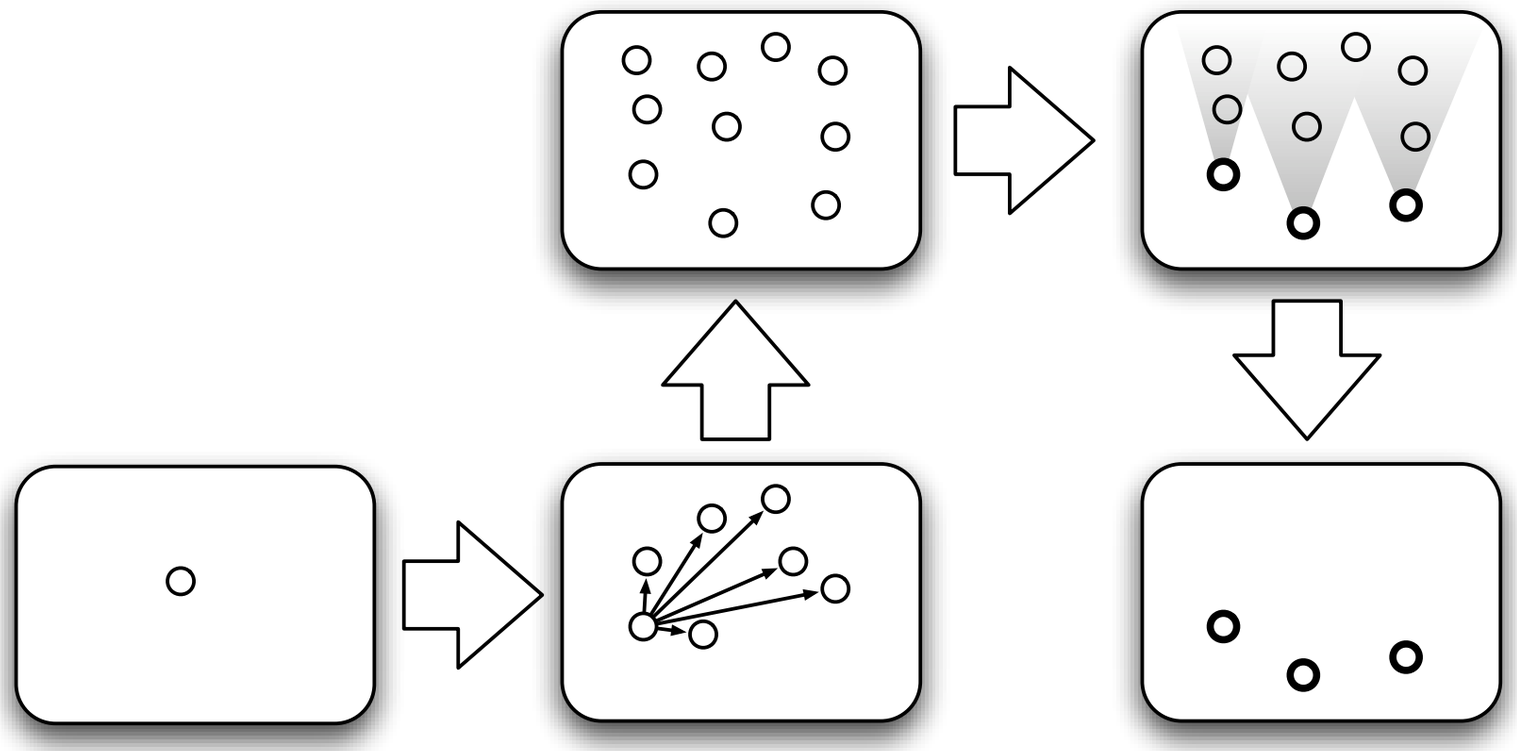
well-structured



recursive set of successors



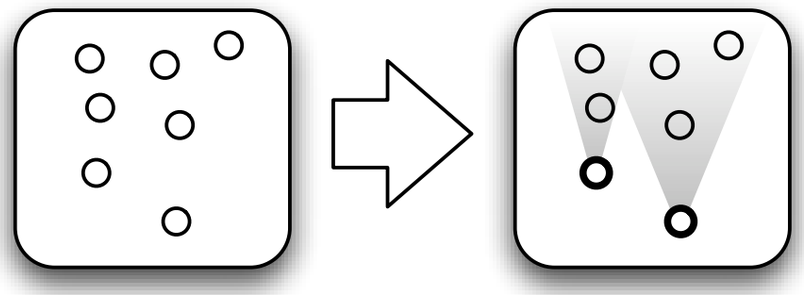
The algoritm



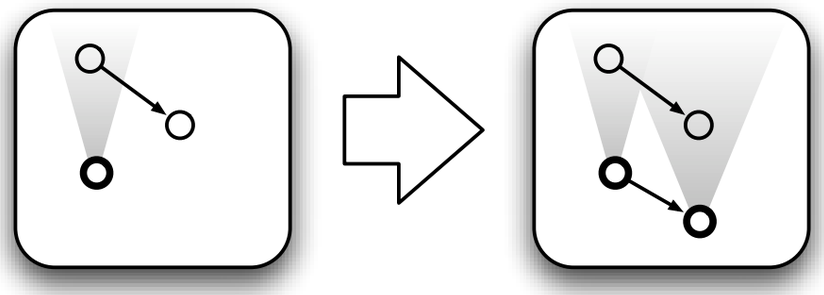
Recipe

The ingredients

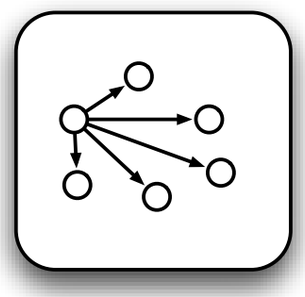
well quasi-order



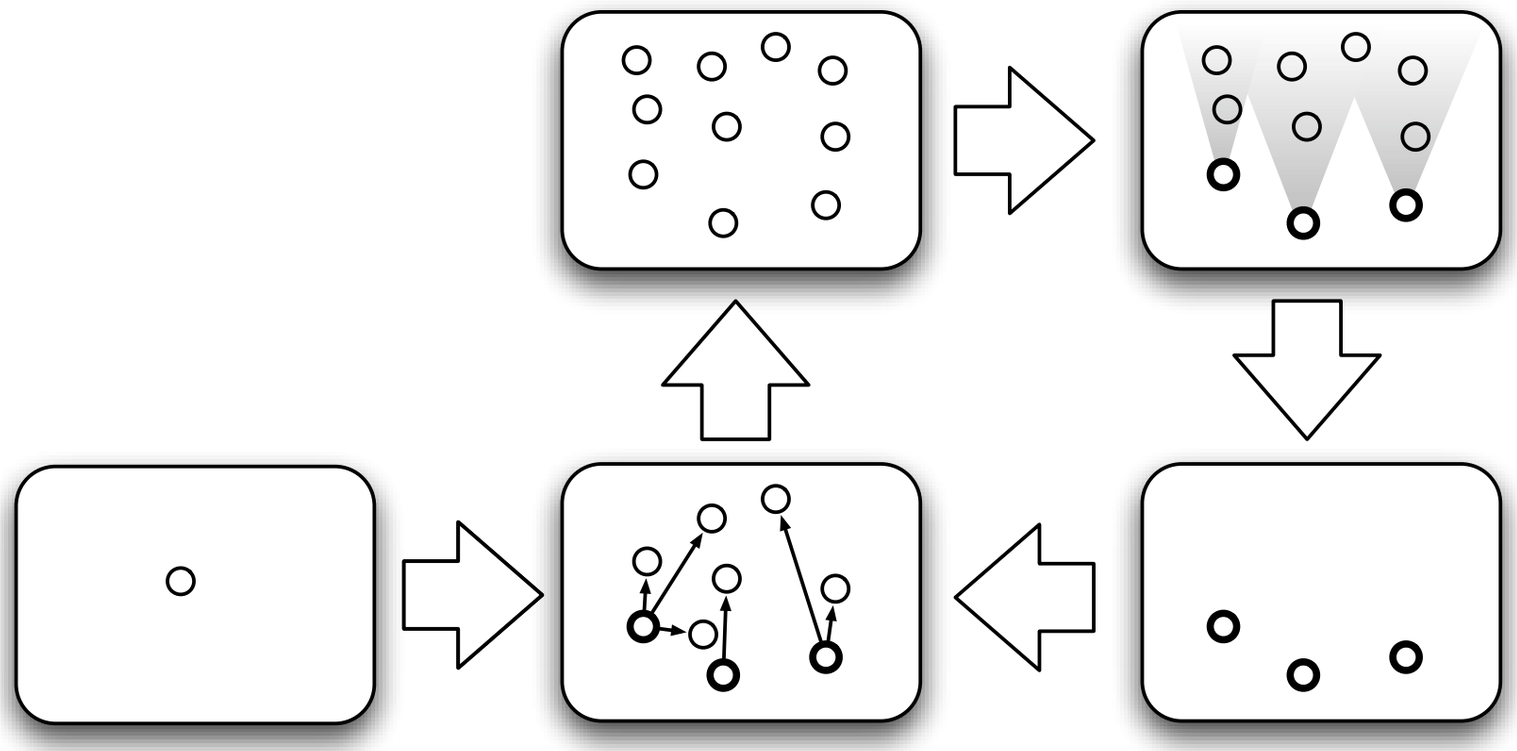
well-structured



recursive set of successors



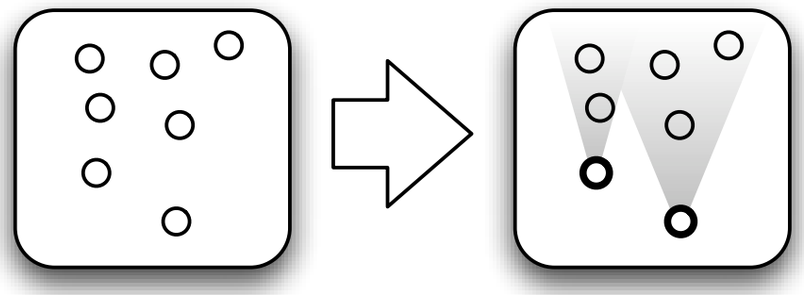
The algoritm



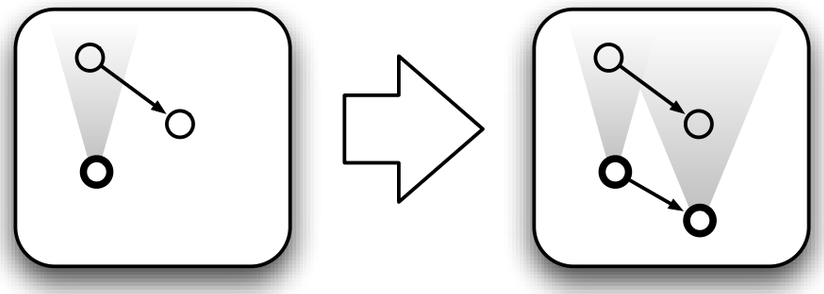
Recipe

The ingredients

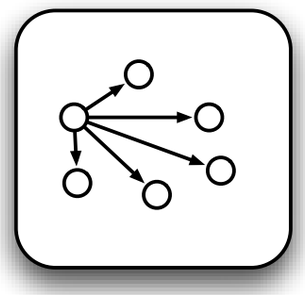
well quasi-order



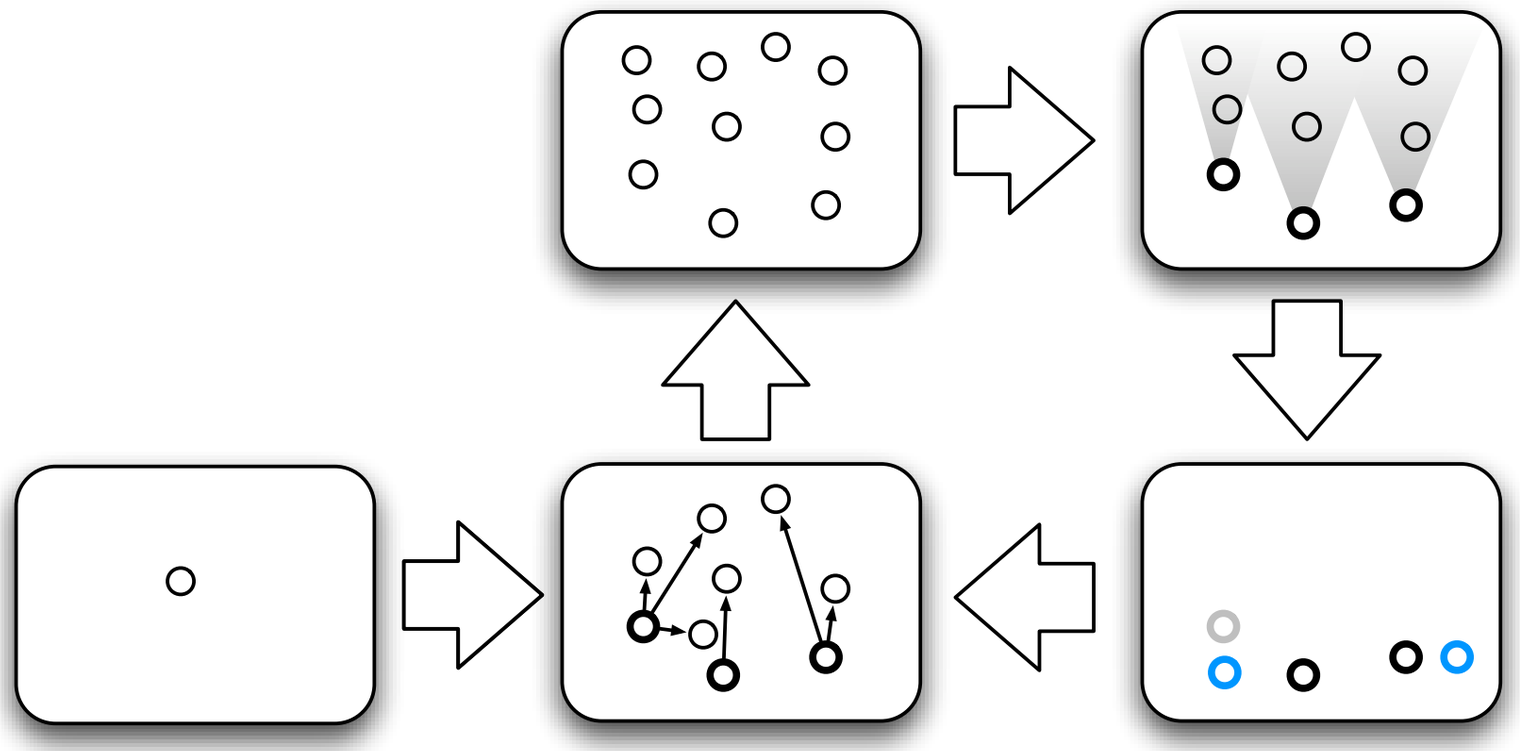
well-structured



recursive set of successors



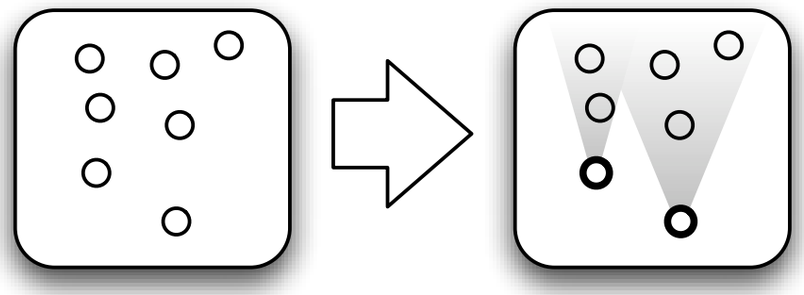
The algoritm



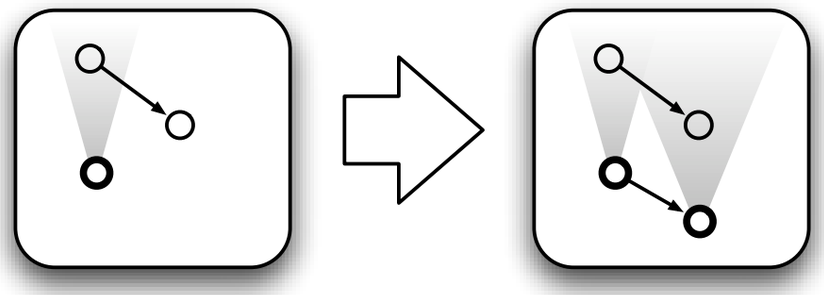
Recipe

The ingredients

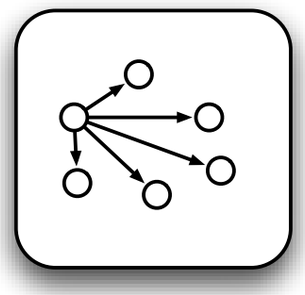
well quasi-order



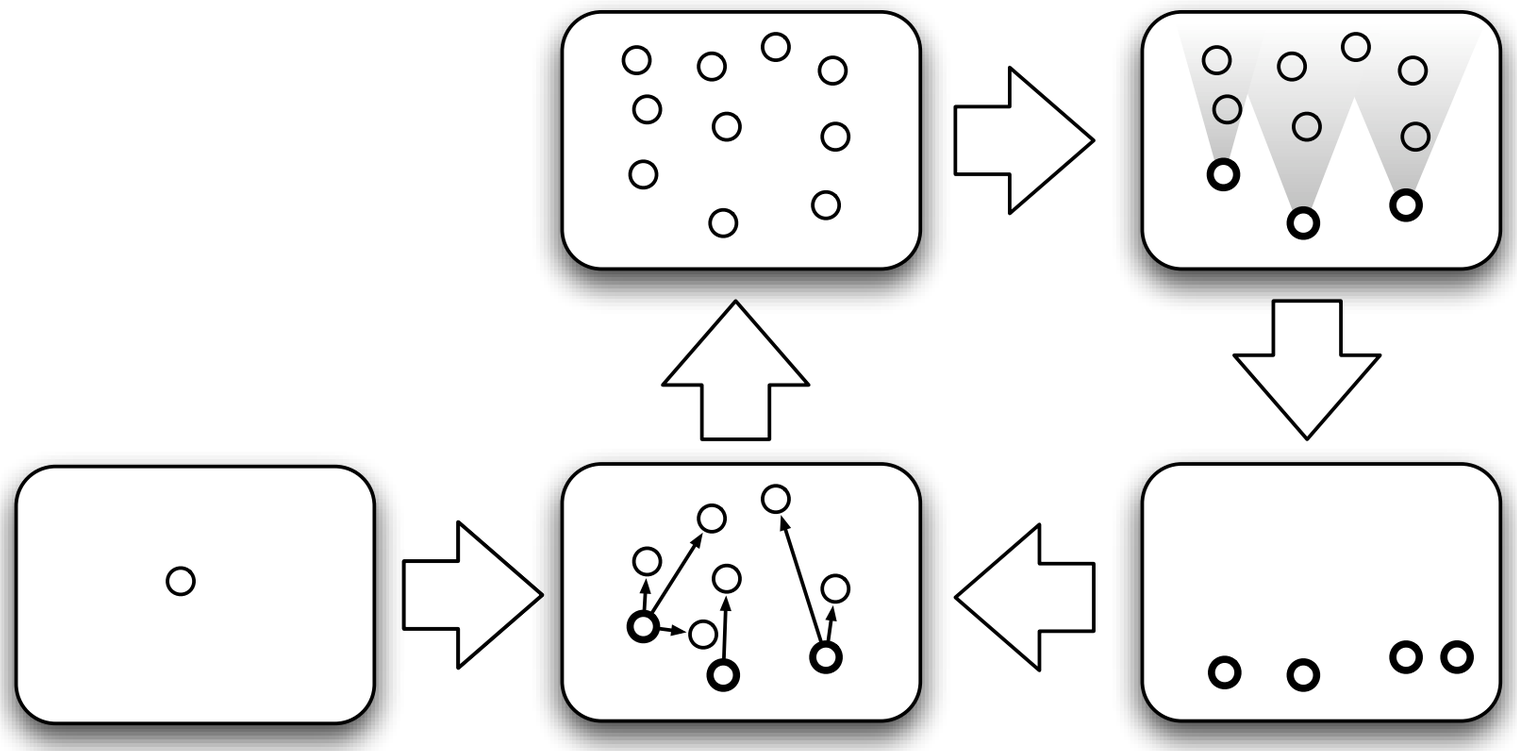
well-structured



recursive set of successors



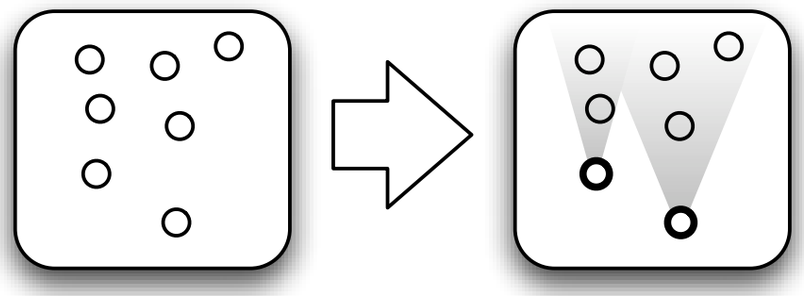
The algoritm



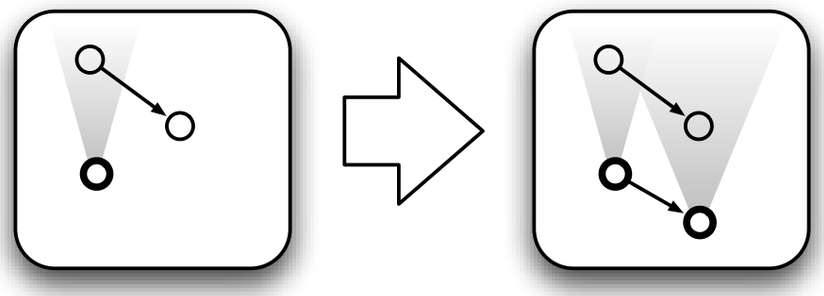
Recipe

The ingredients

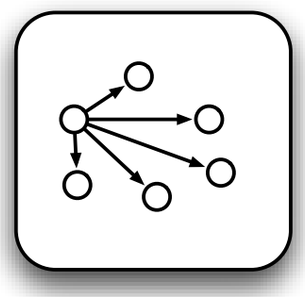
well quasi-order



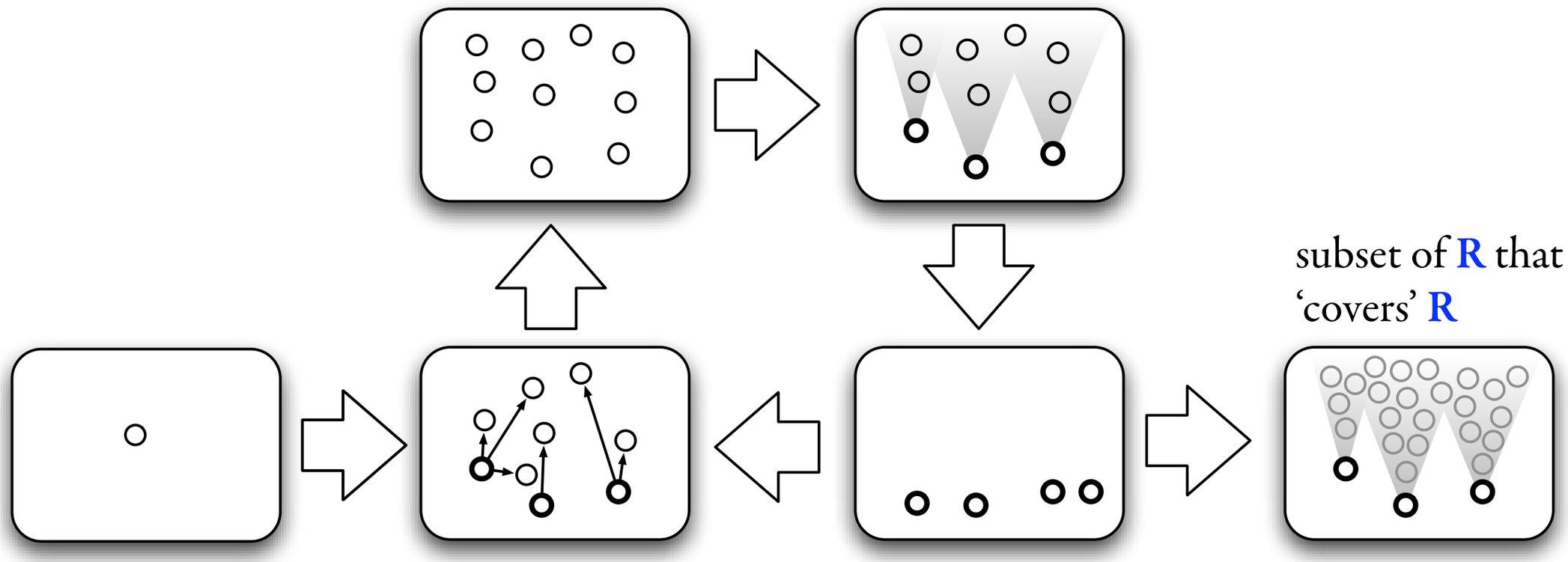
well-structured



recursive set of successors



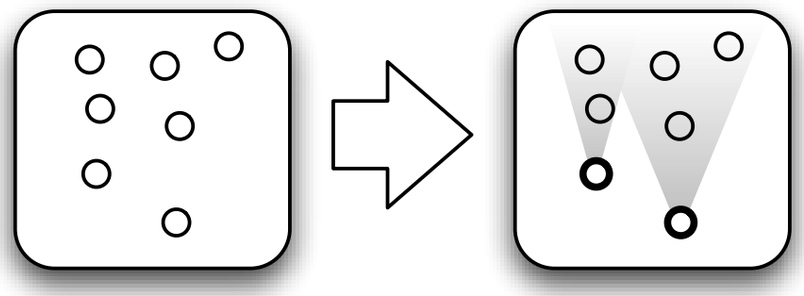
The algoritm



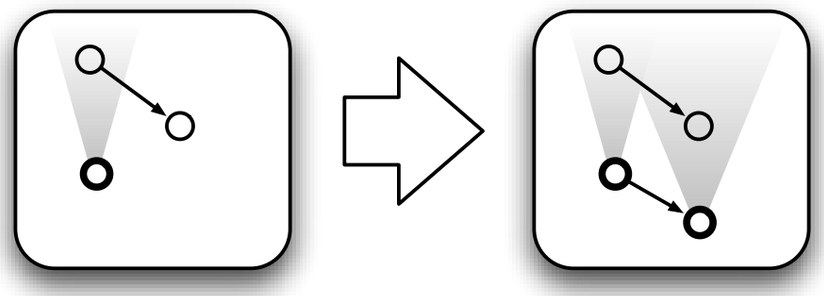
Recipe

The ingredients

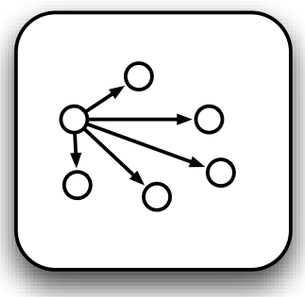
well quasi-order



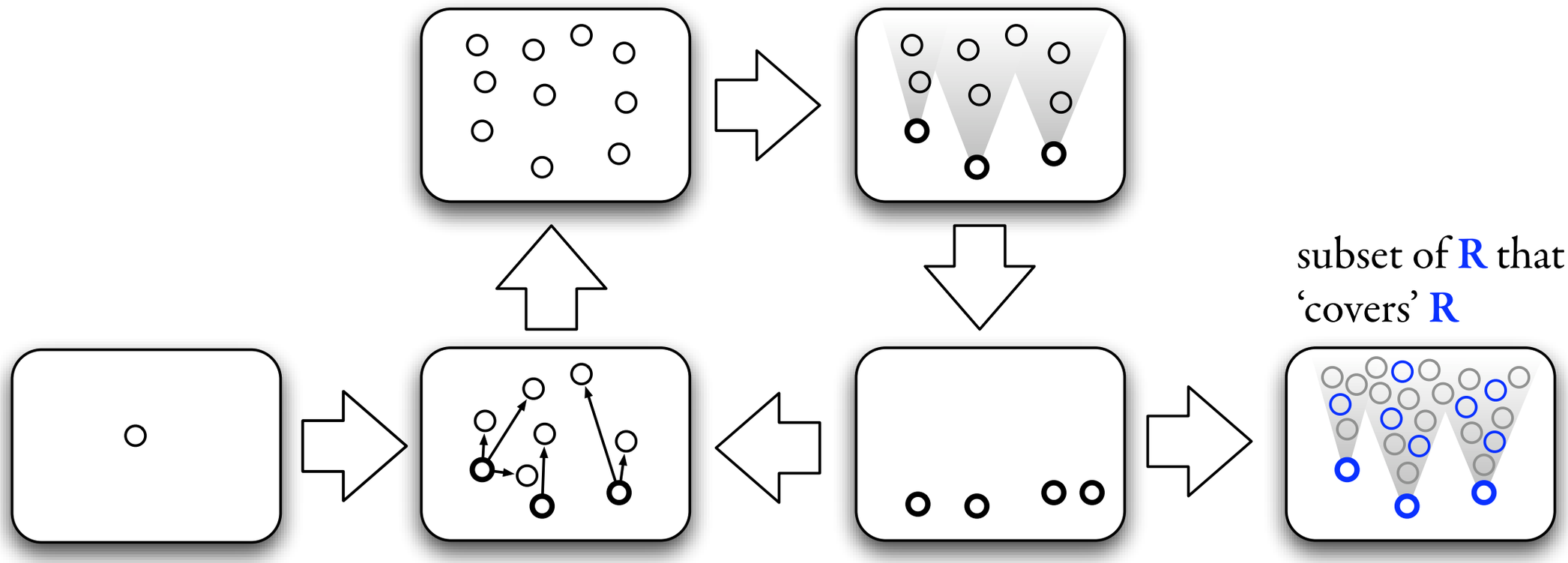
well-structured



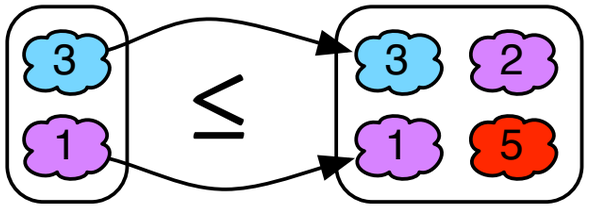
recursive set of successors



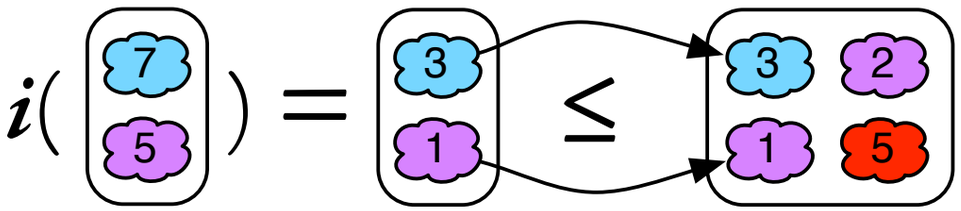
The algoritm



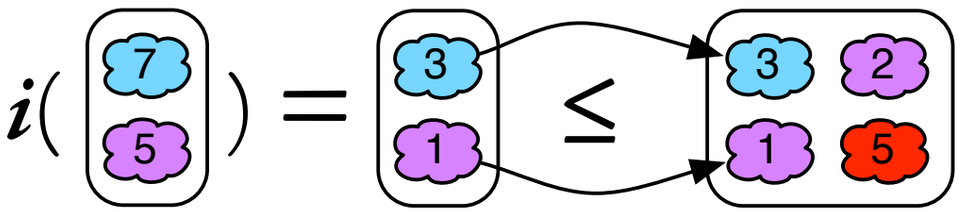
The wqo $\leq$



The wqo $\leq$

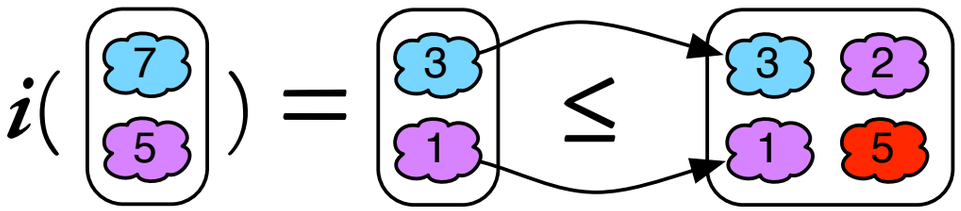


The wqo $\leq$



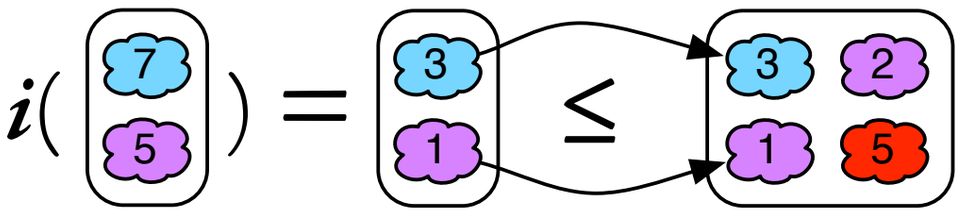
qo reflexive transitive

The wqo $\leq$



wqo
well-
reflexive
transitive
founded

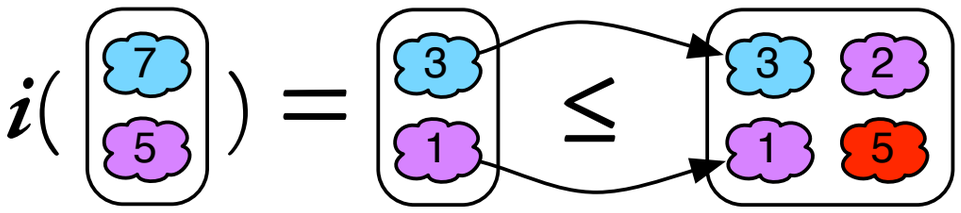
The wqo $\leq$



no infinite antichains

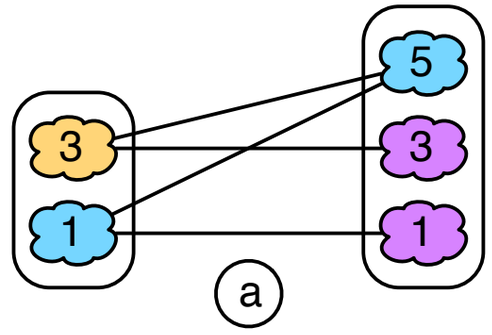
wqo reflexive transitive well-founded

The wqo ≤

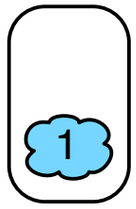


no infinite antichains
wqo reflexive
transitive
well- founded

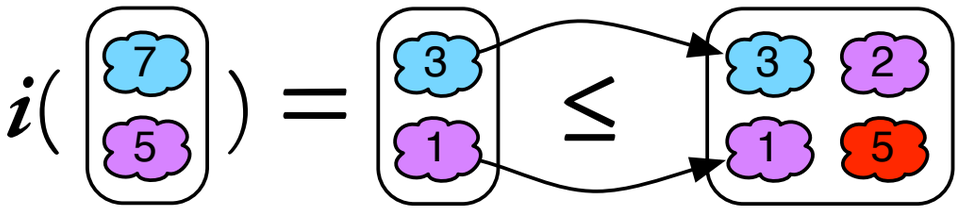
Well structured



VI

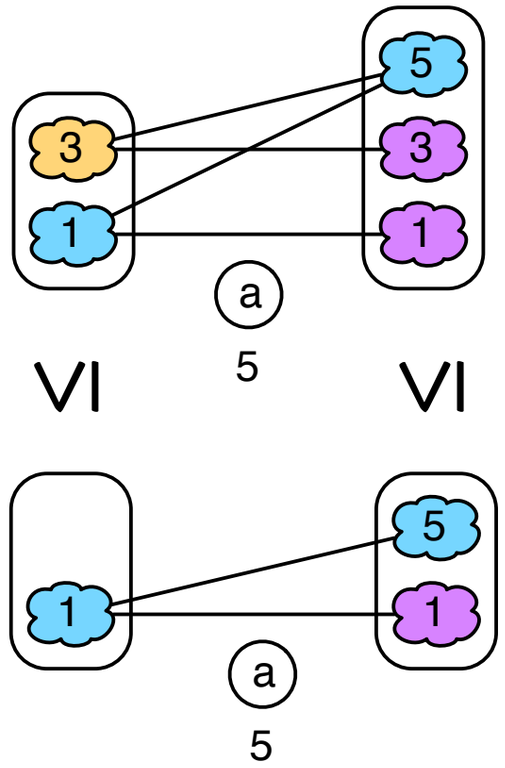


The wqo ≤

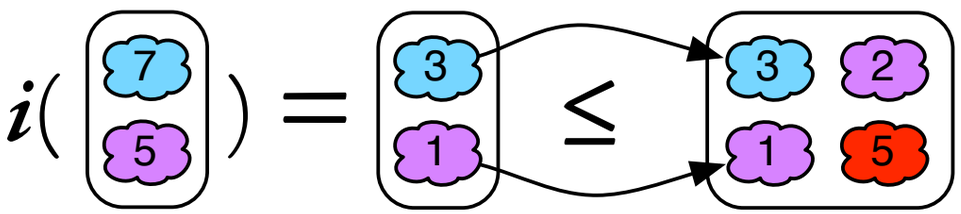


no infinite antichains
wqo reflexive
transitive
well- founded

Well structured



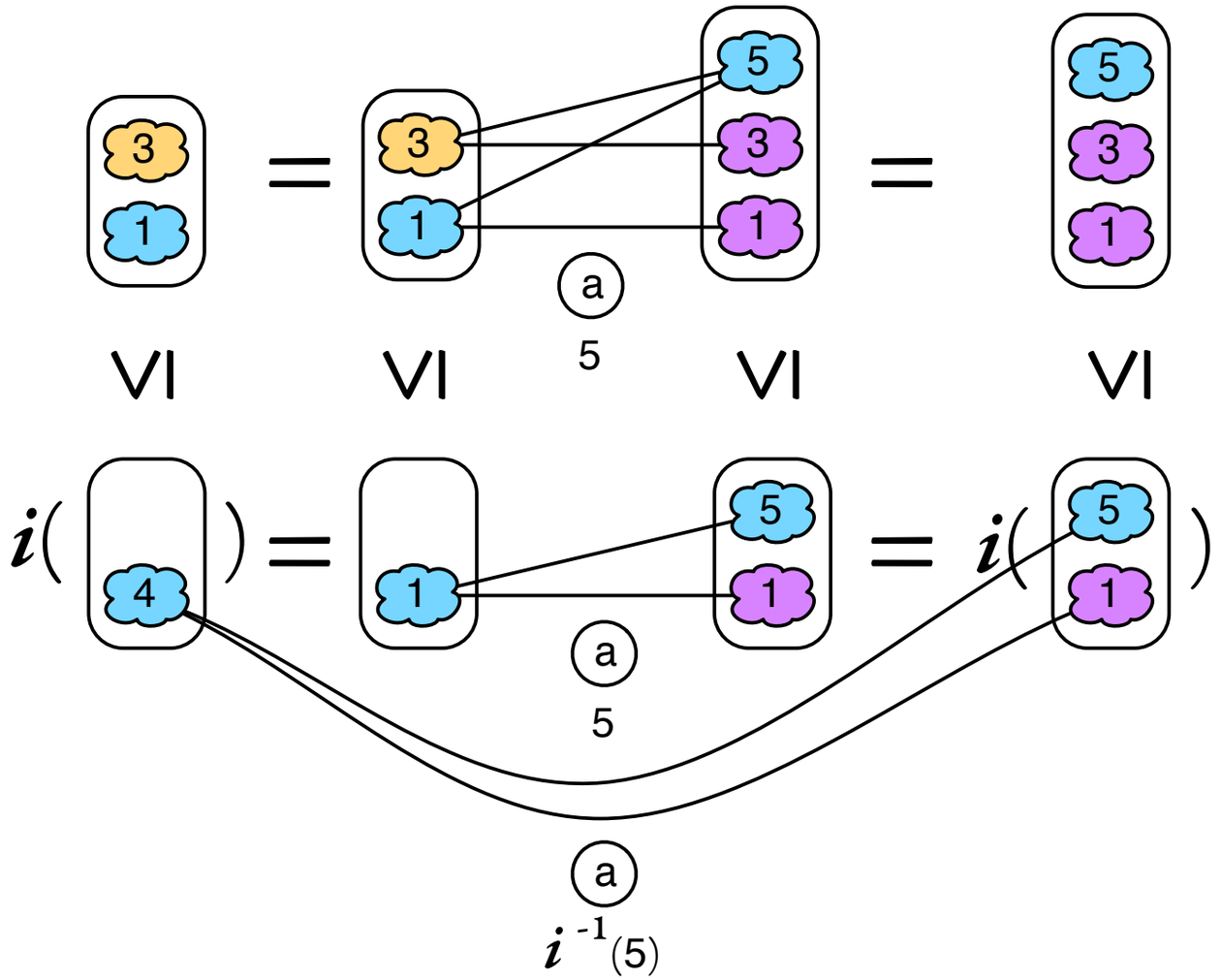
The wqo ≤



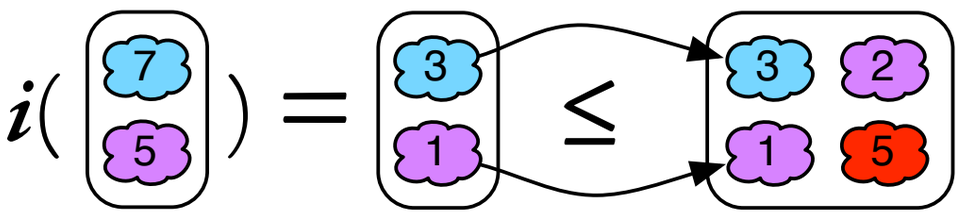
no infinite antichains

wqo reflexive transitive well-founded

Well structured



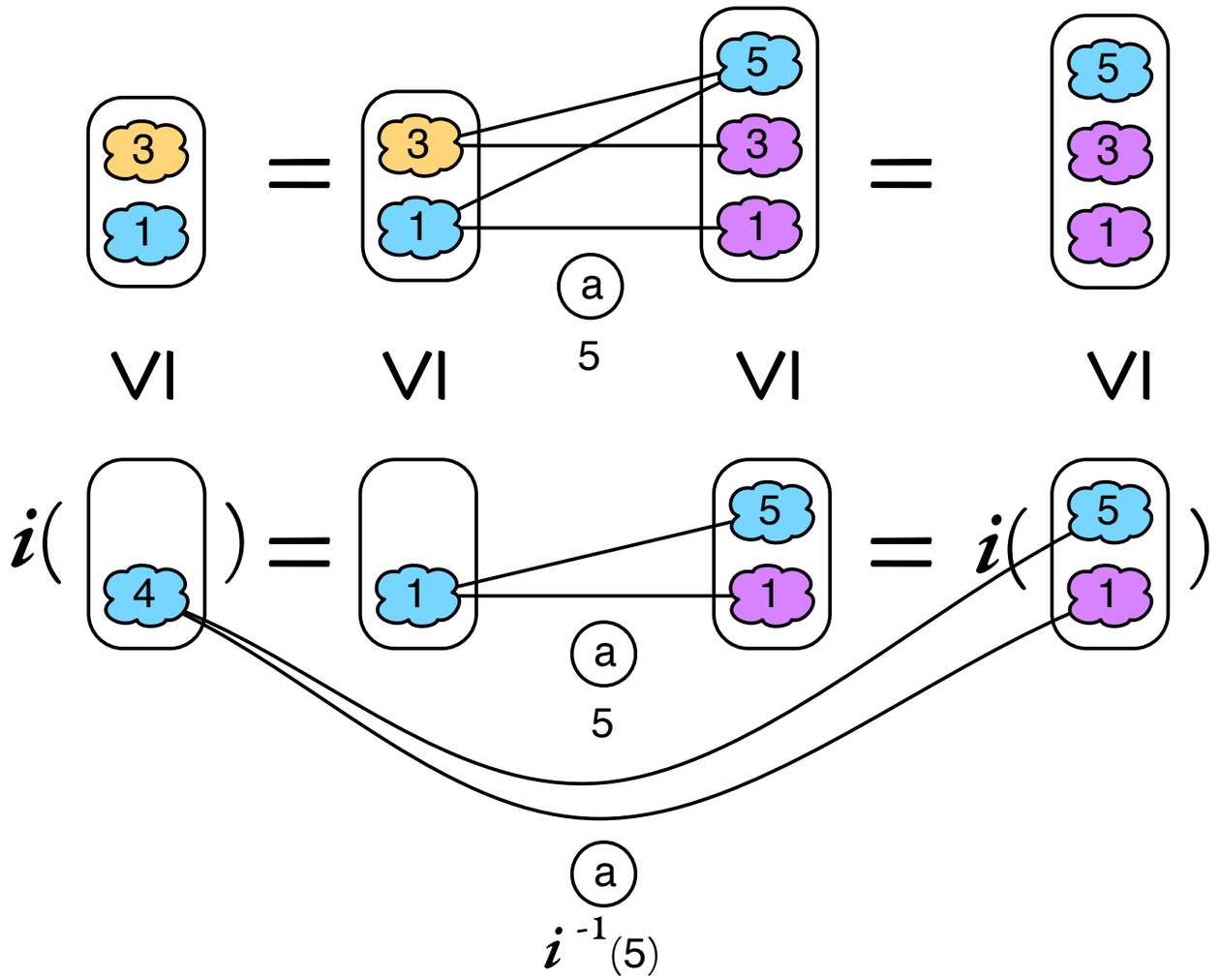
The wqo ≤



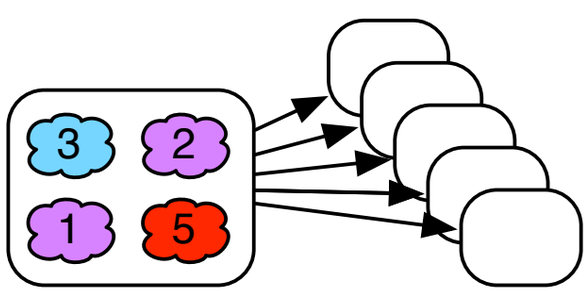
no infinite antichains

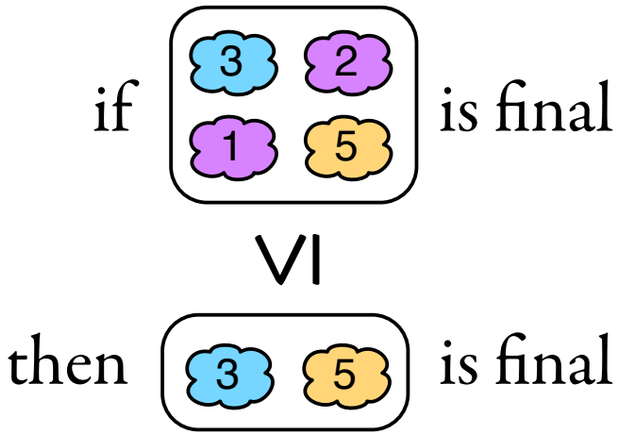
wqo = reflexive + transitive + well-founded

Well structured

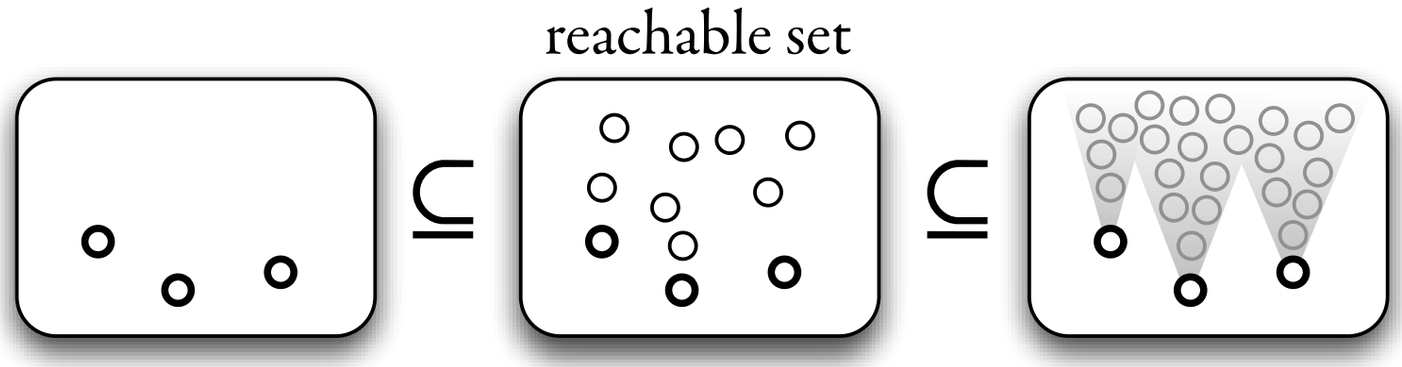


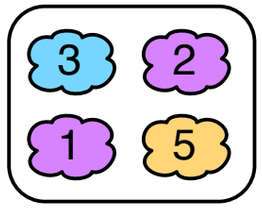
Recursive set of sucessors
(finite up to isomorphism)





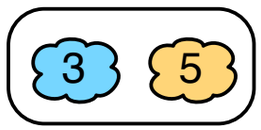
By the algorithm...



if 

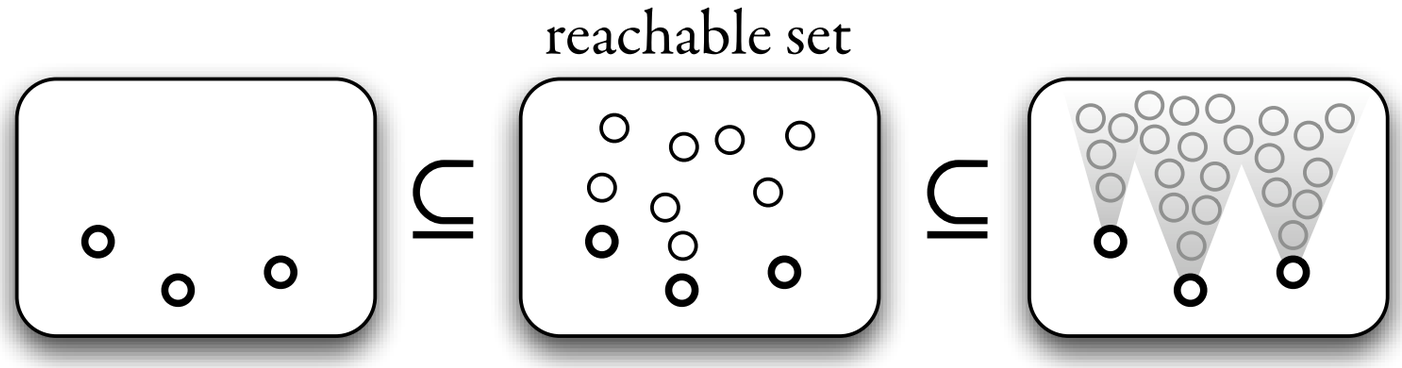
is final

VI

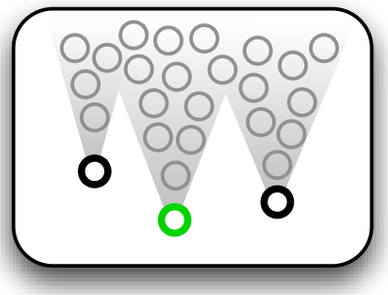
then 

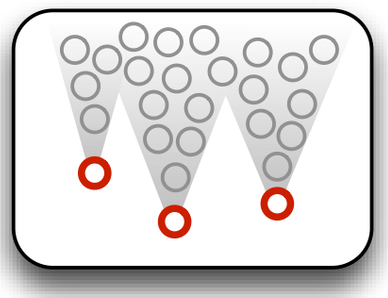
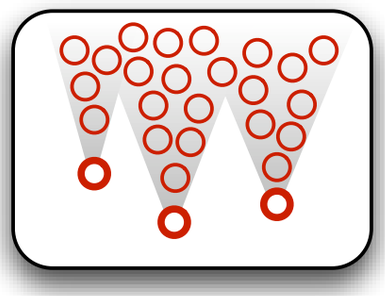
is final

By the algorithm...



We only test the minimal elements

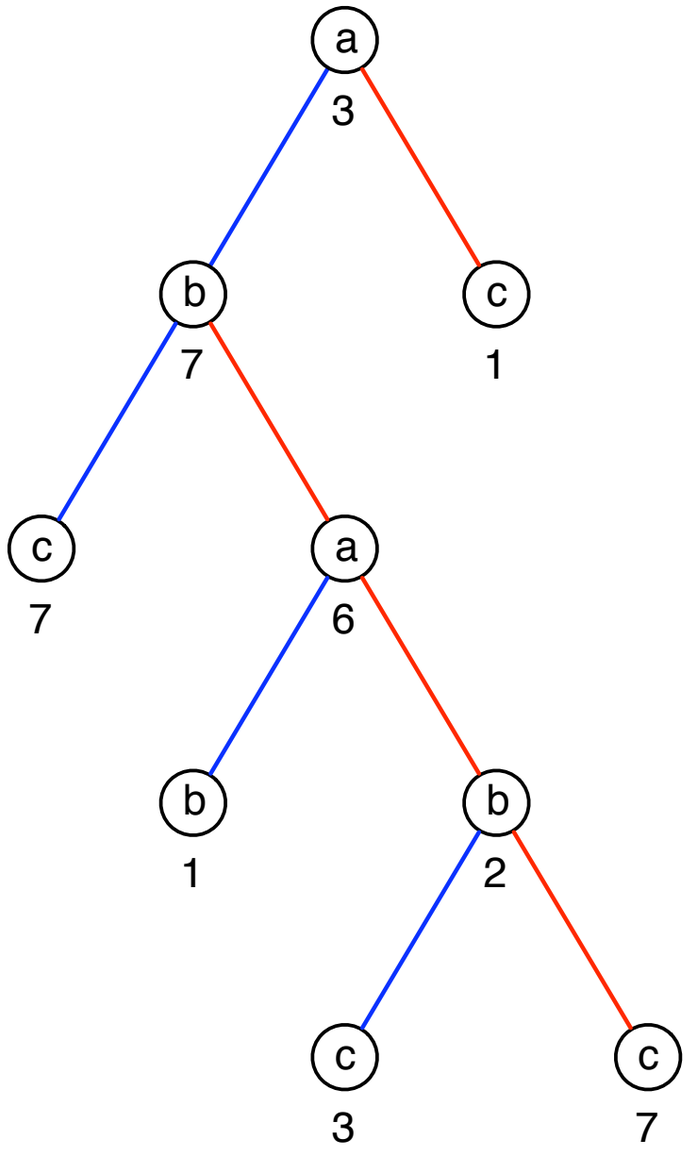
If  then there is an accepting run.

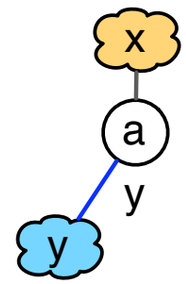
If  then  there is no accepting run.

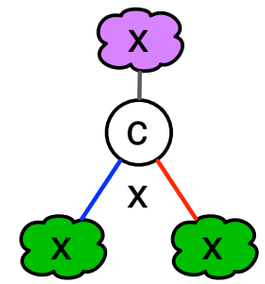
From words to trees

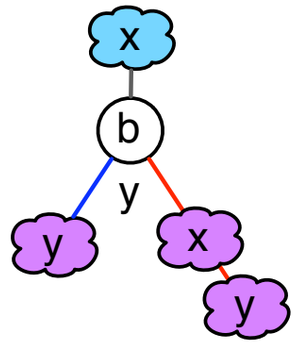
initial state

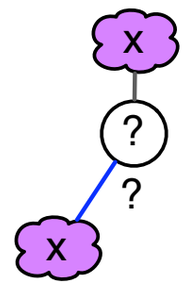
final state

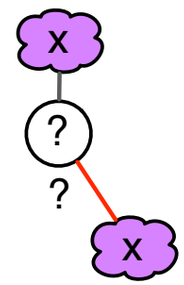








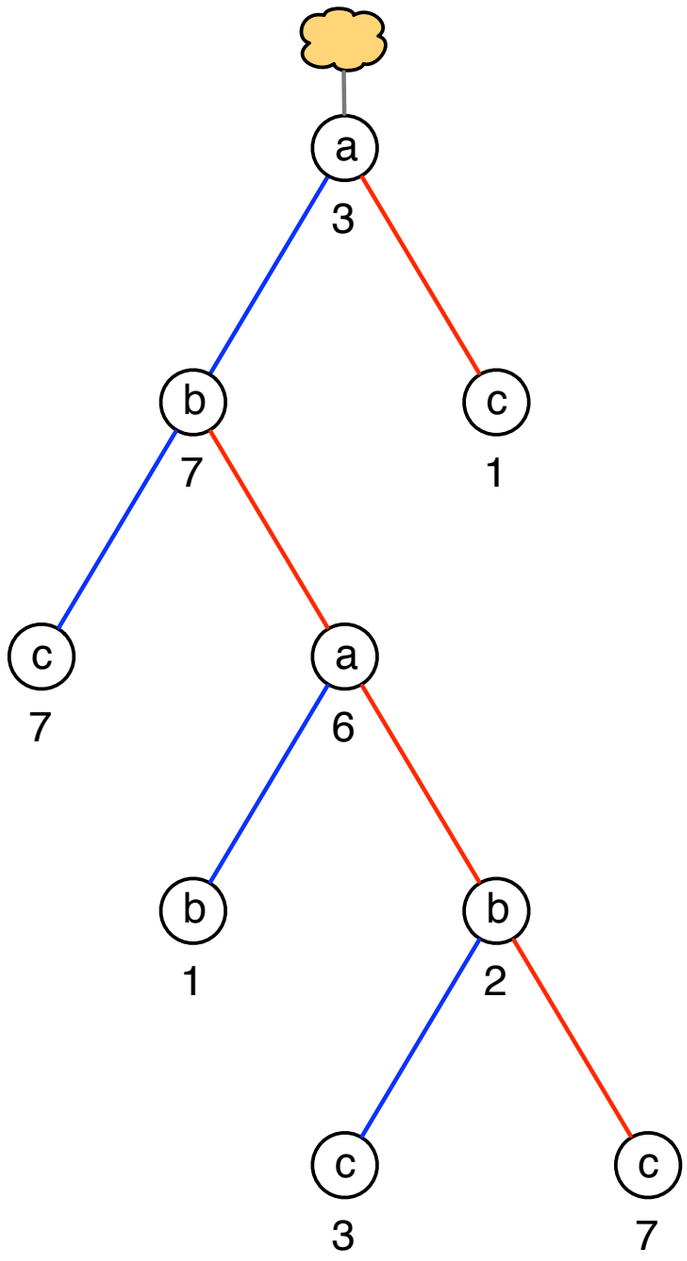


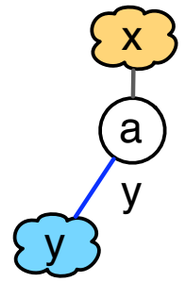


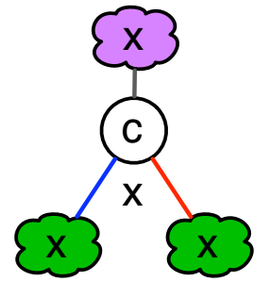
From words to trees

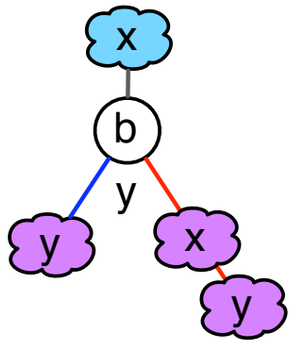
initial state

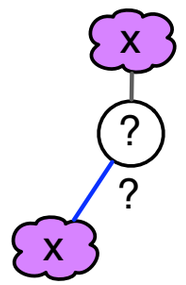
final state

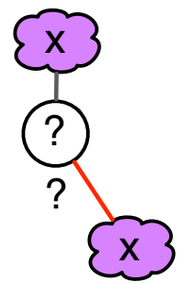








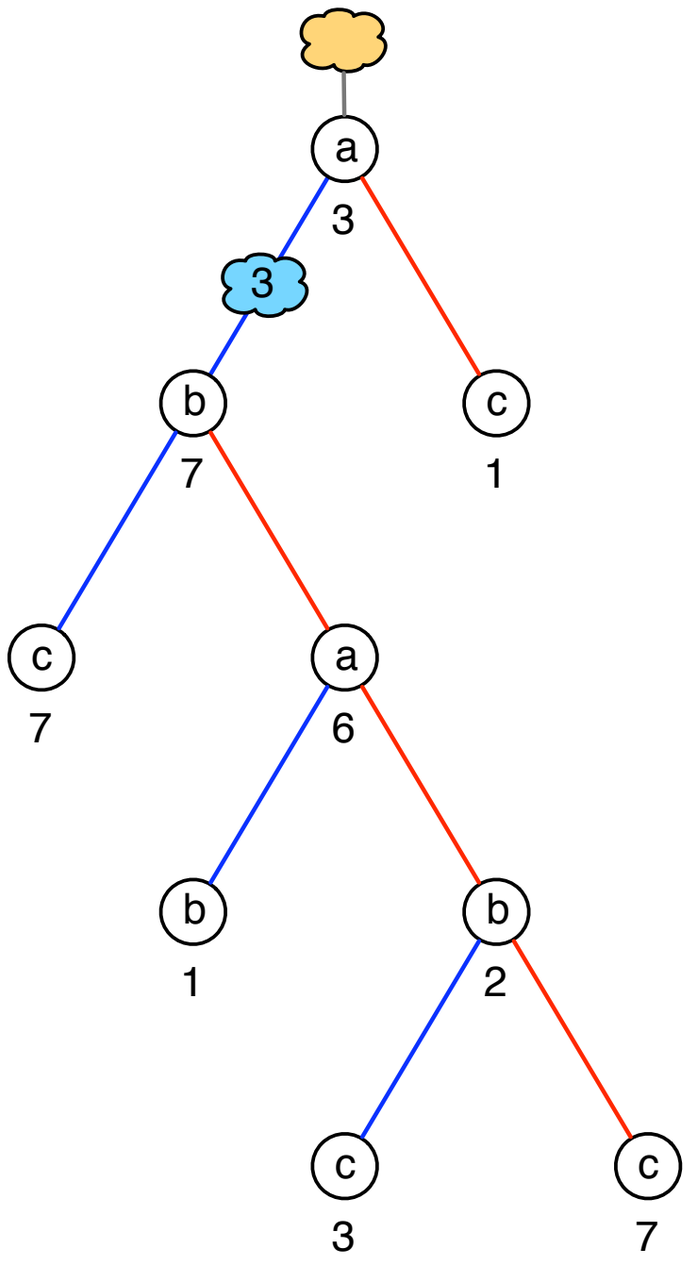


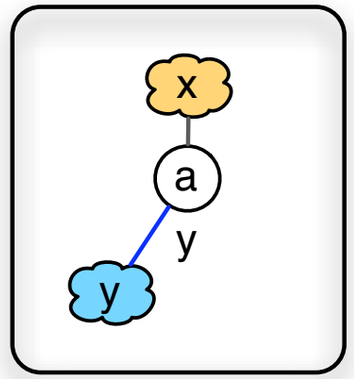


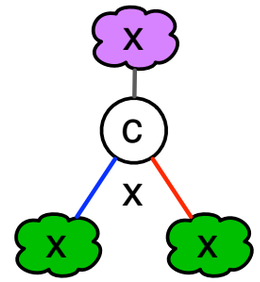
From words to trees

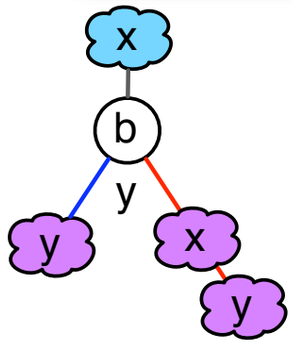
initial state

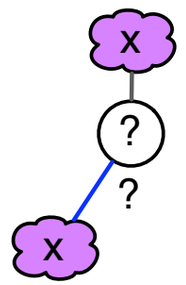
final state

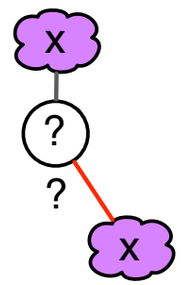








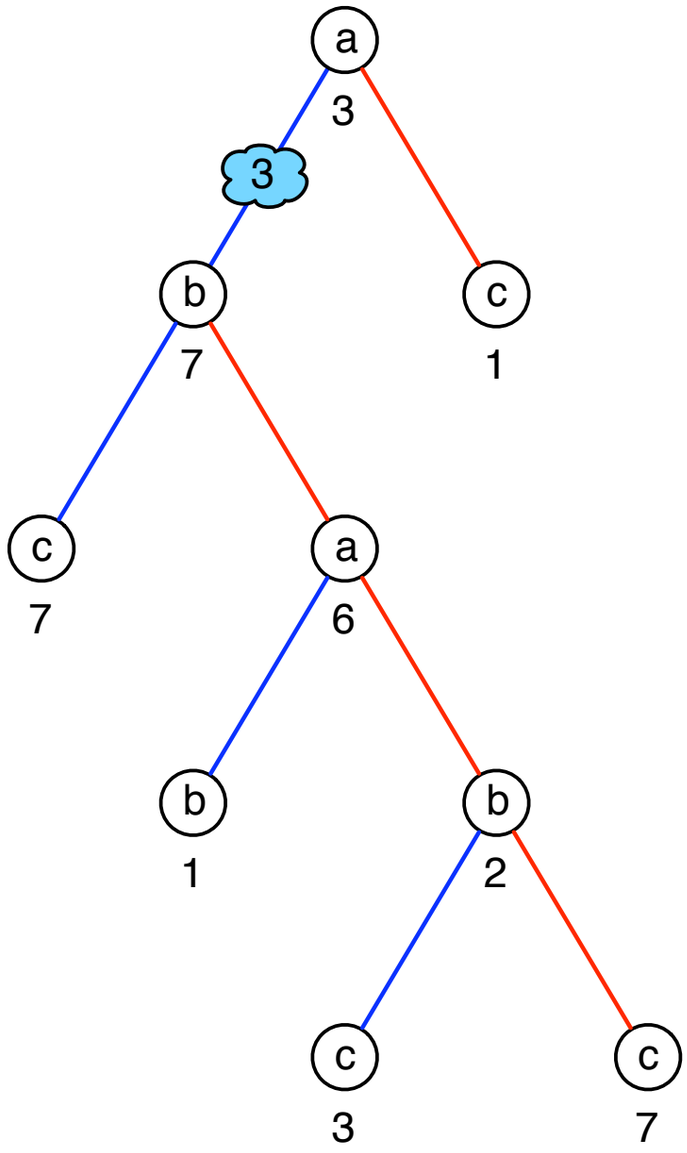


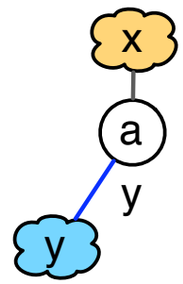


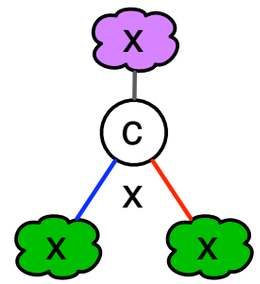
From words to trees

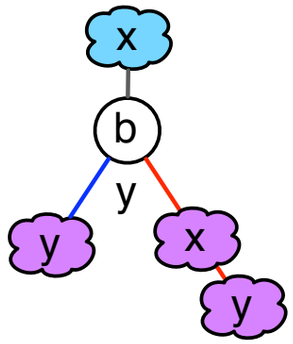
initial state

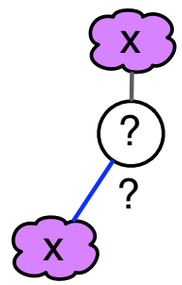
final state

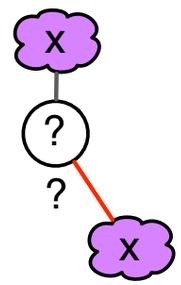








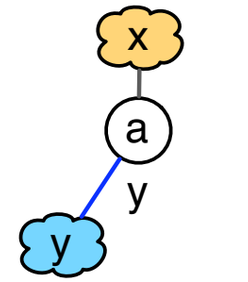


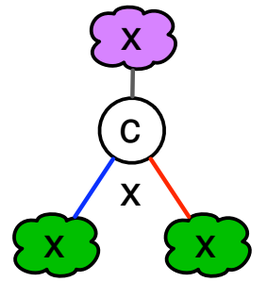


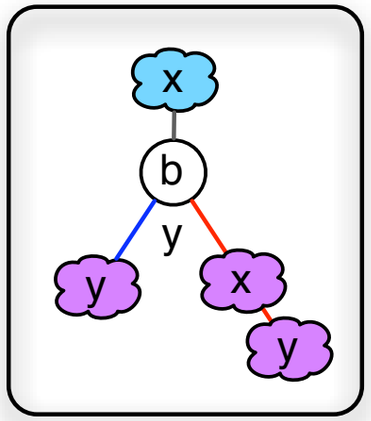
From words to trees

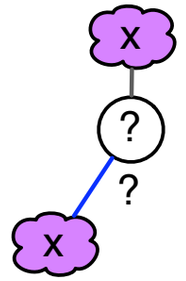
initial state

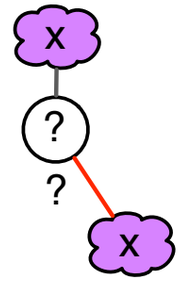
final state

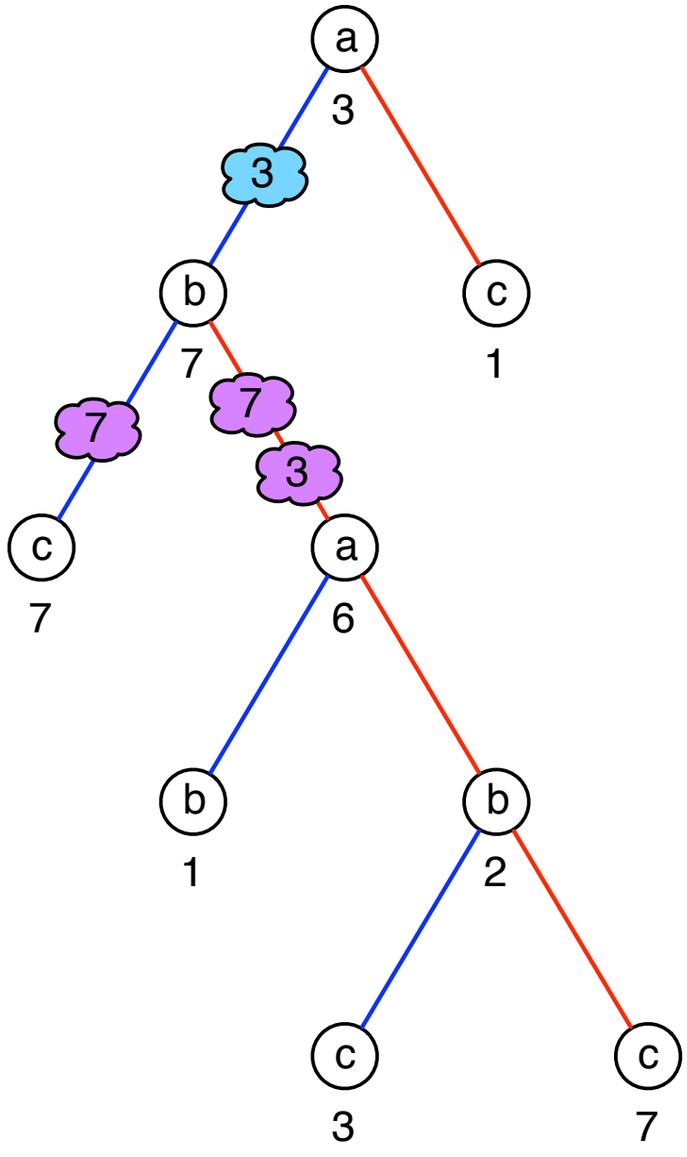








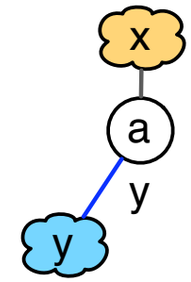


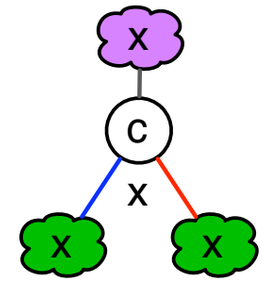


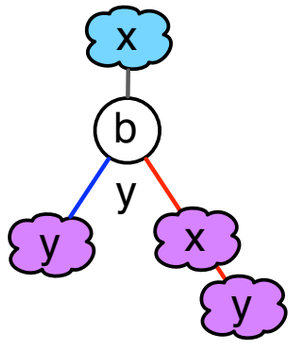
From words to trees

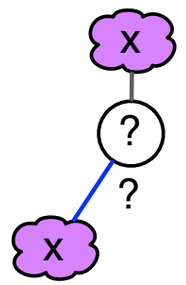
initial state

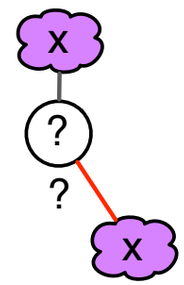
final state

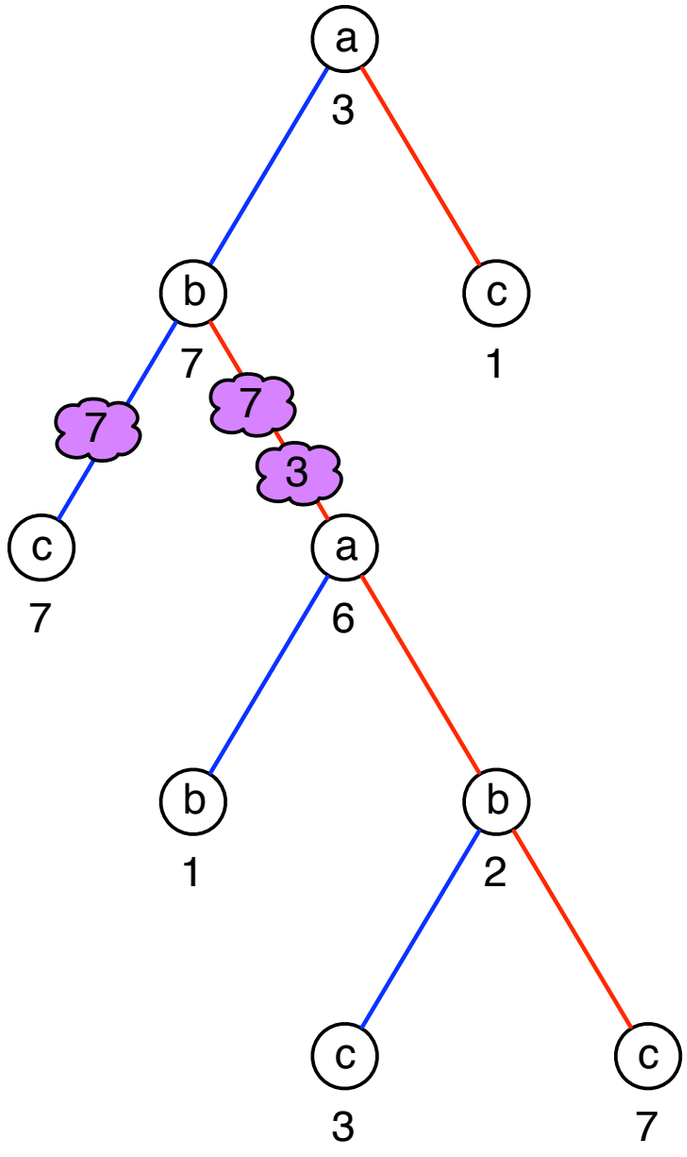








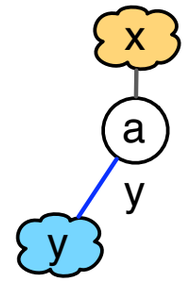


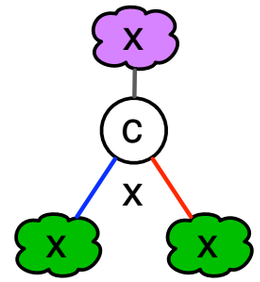


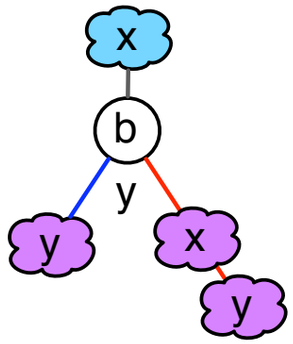
From words to trees

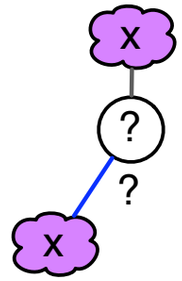
initial state

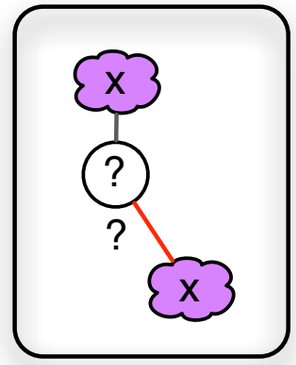
final state

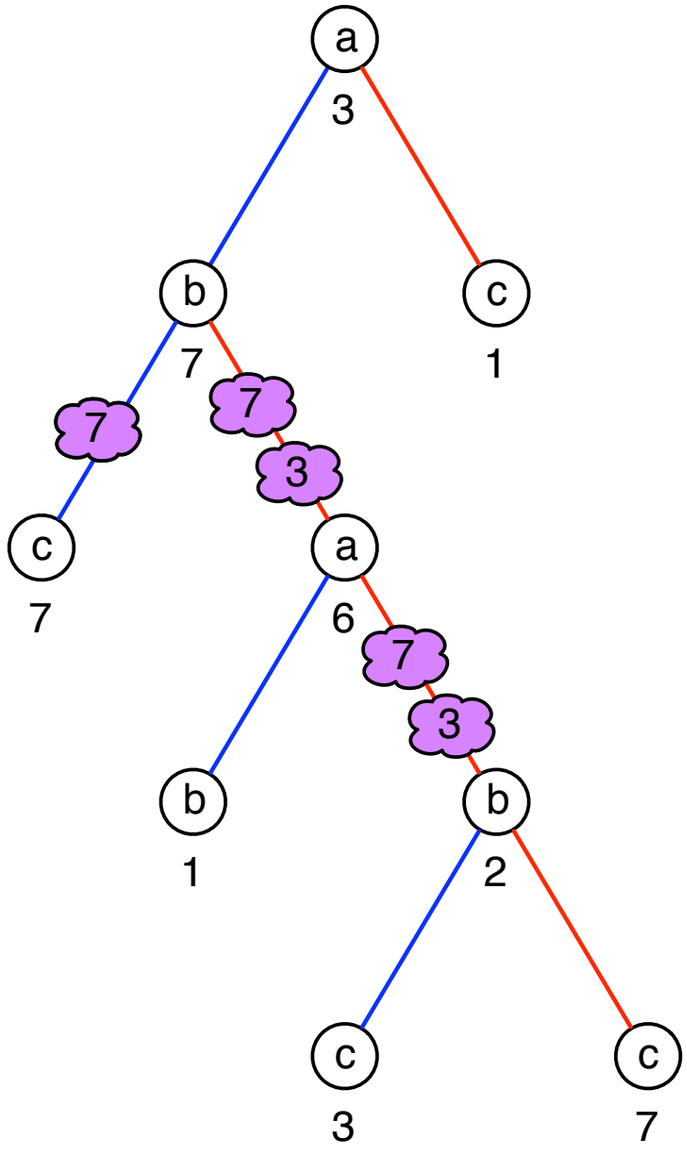








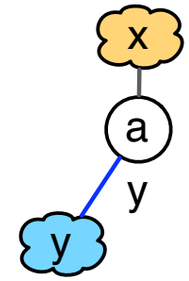


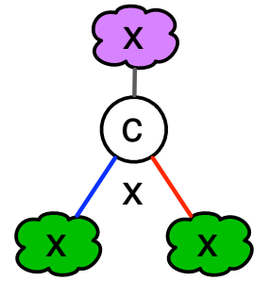


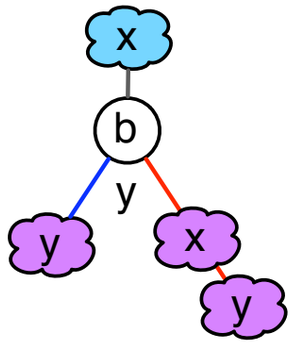
From words to trees

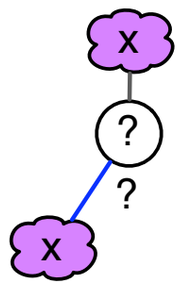
initial state

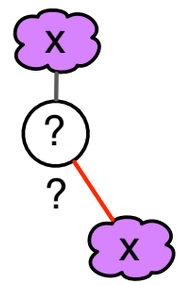
final state

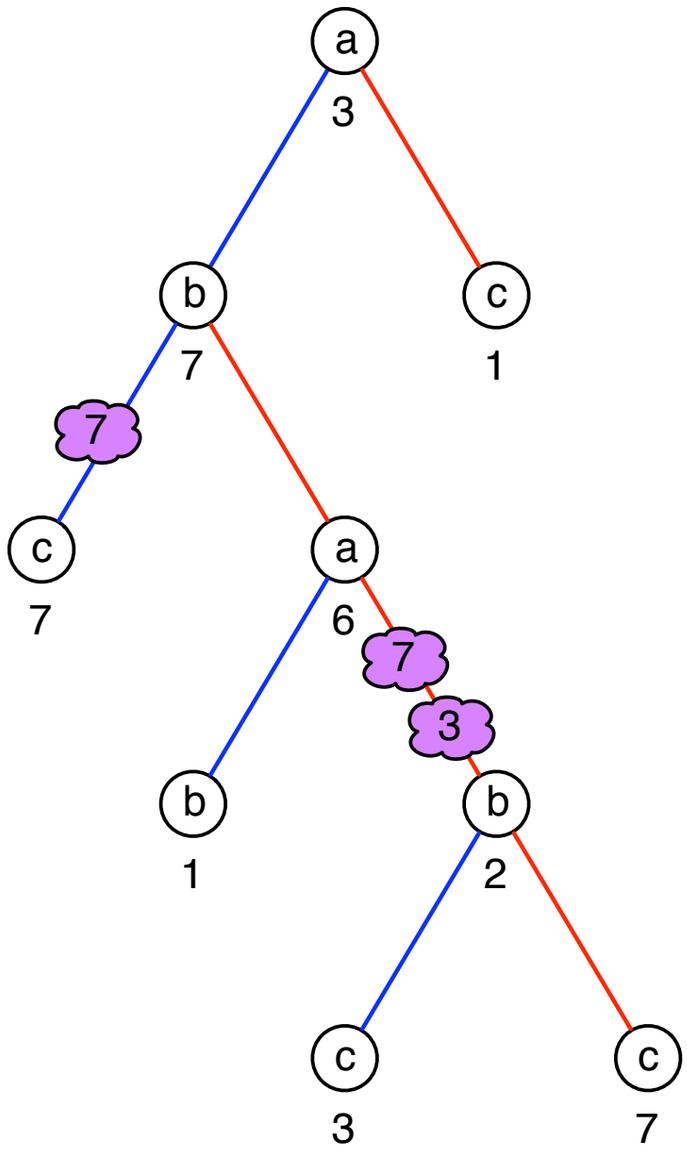








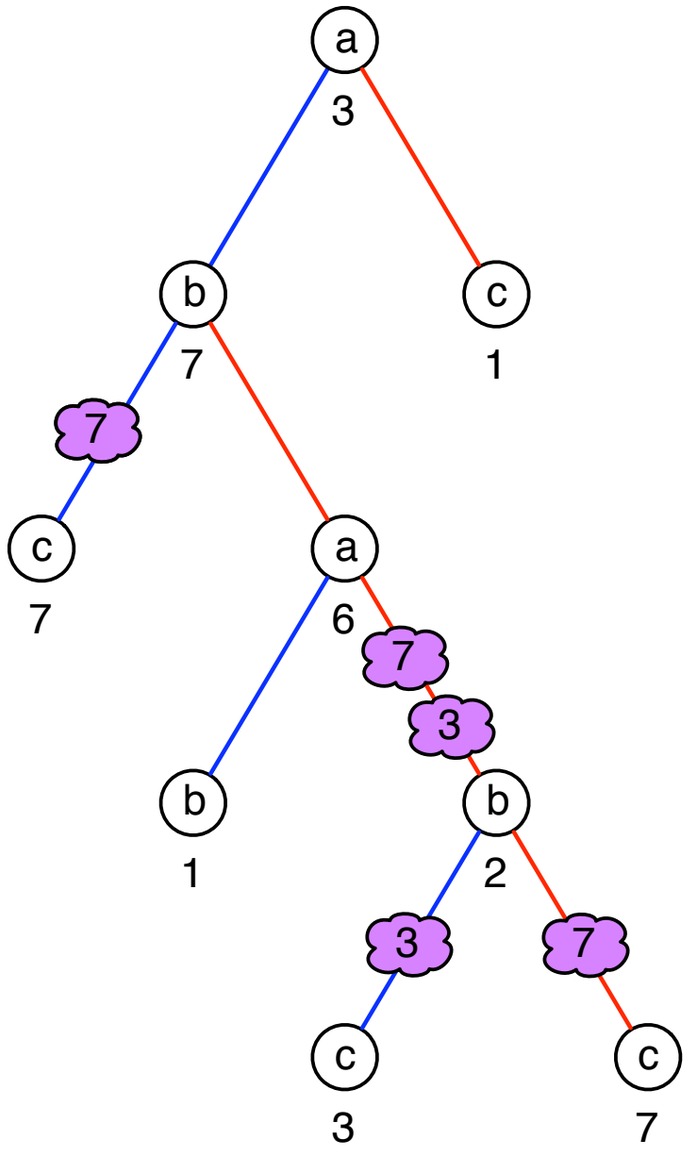


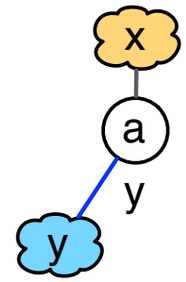


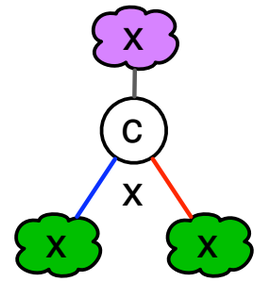
From words to trees

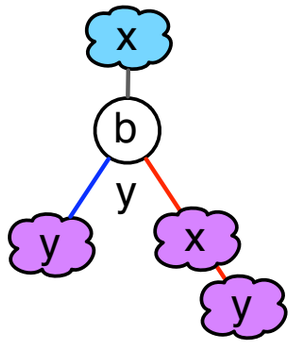
initial state

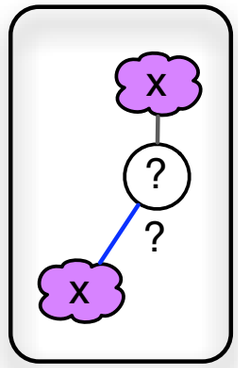
final state

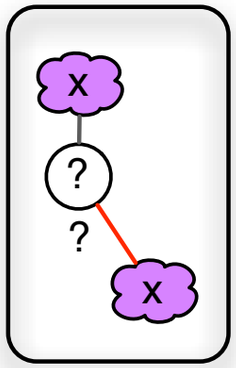








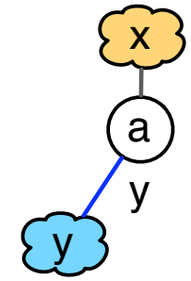


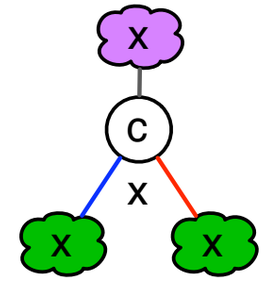


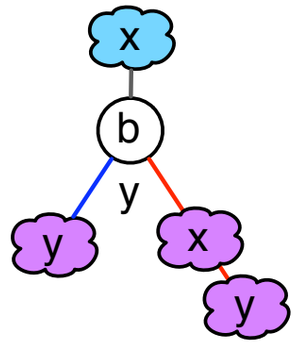
From words to trees

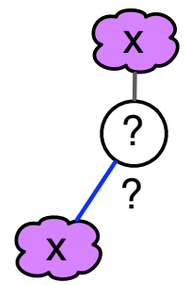
initial state

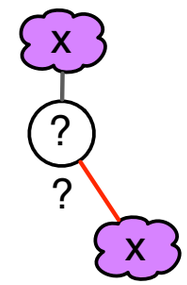
final state

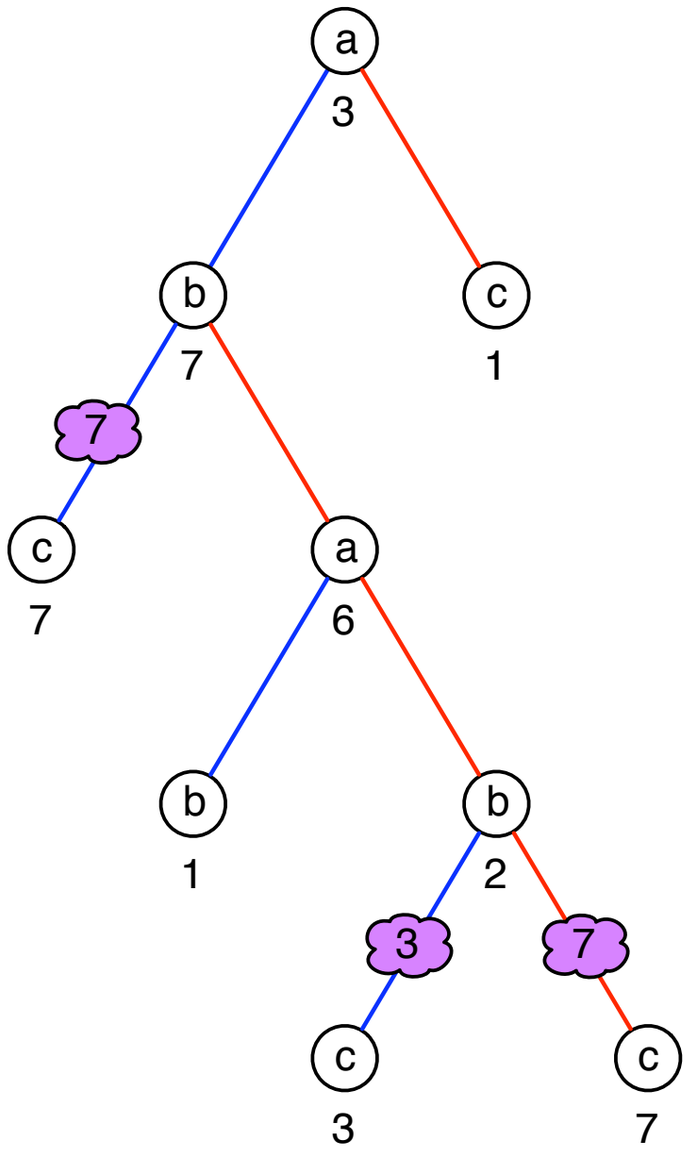








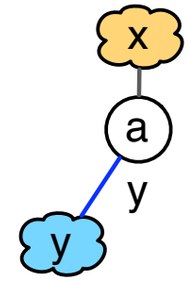


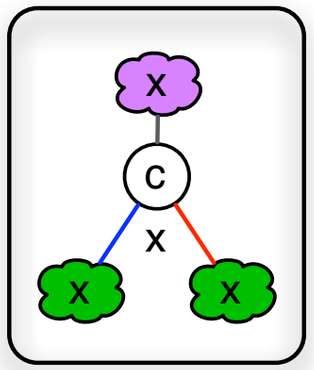


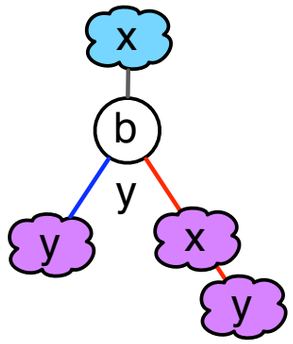
From words to trees

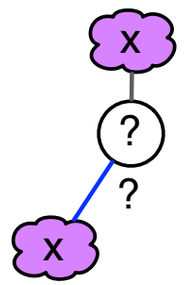
initial state

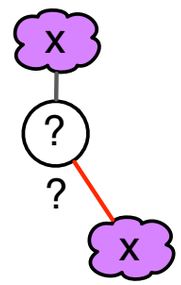
final state

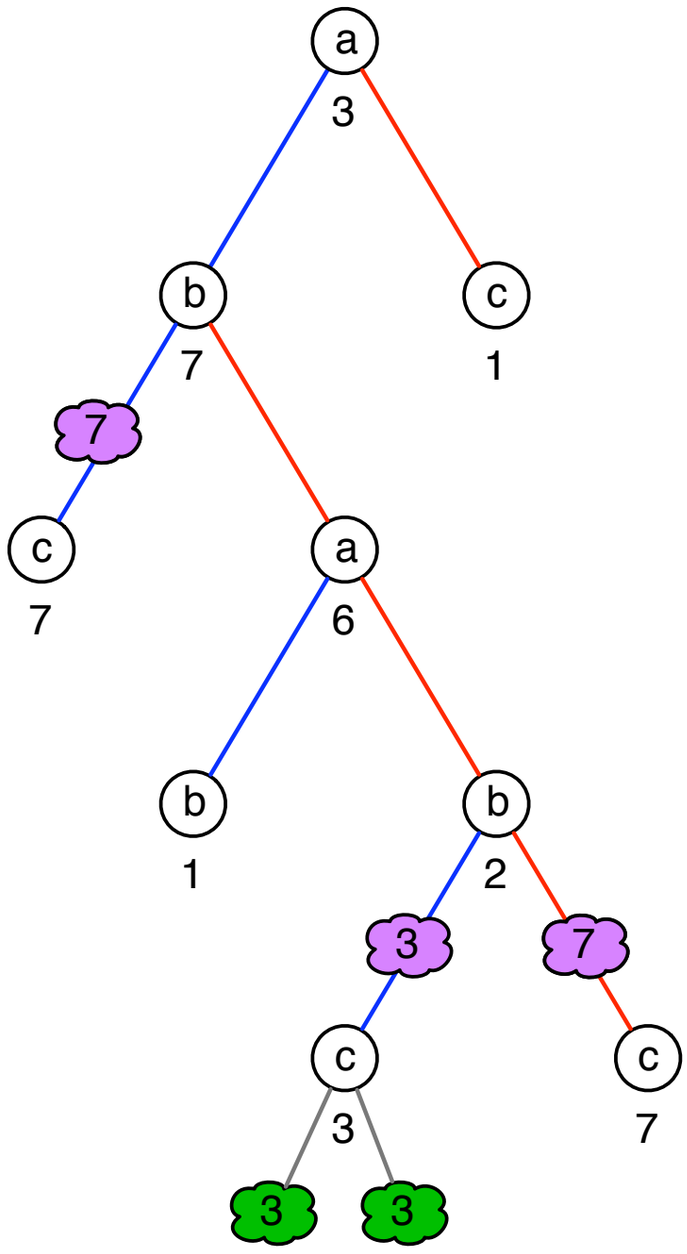








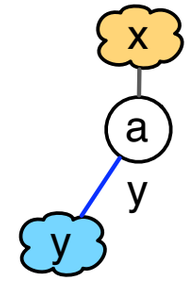


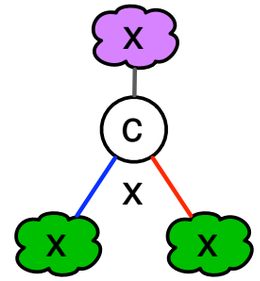


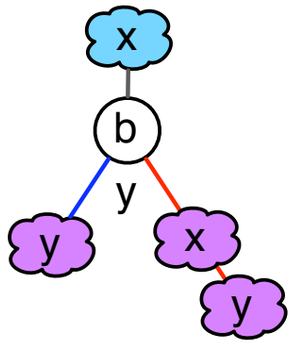
From words to trees

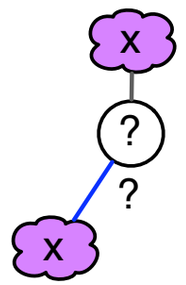
initial state

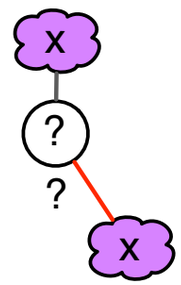
final state

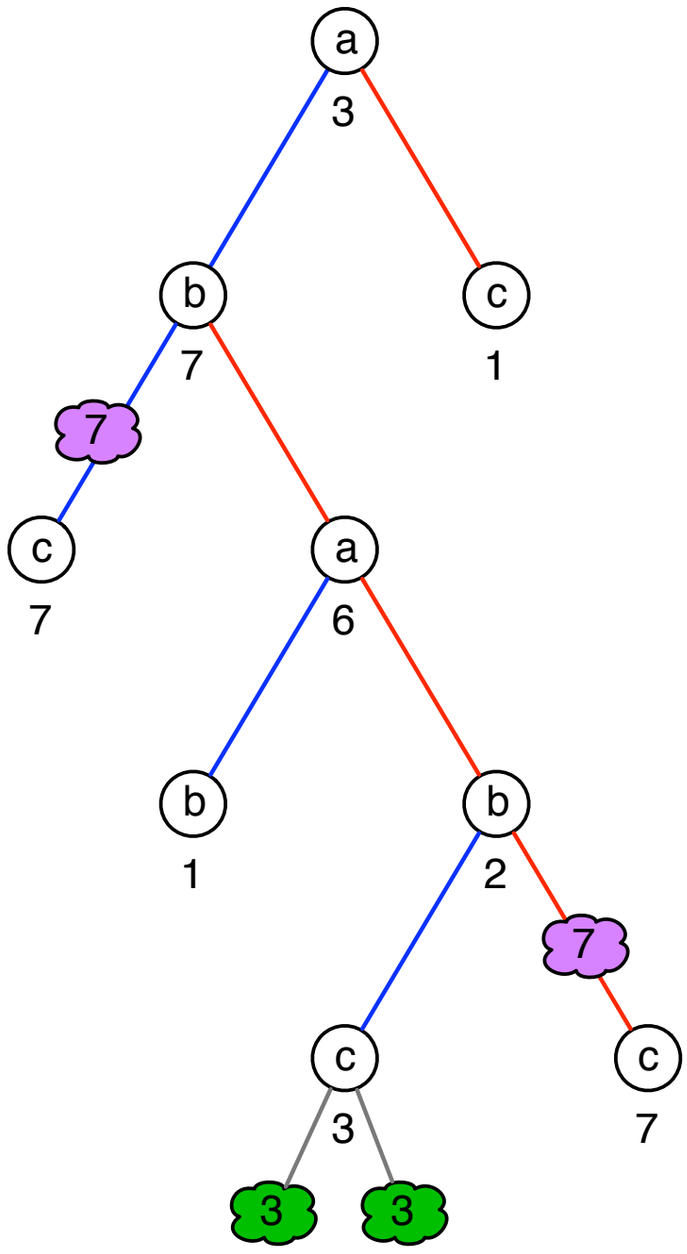








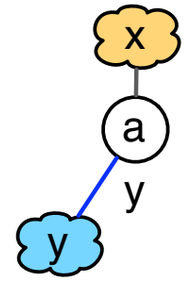


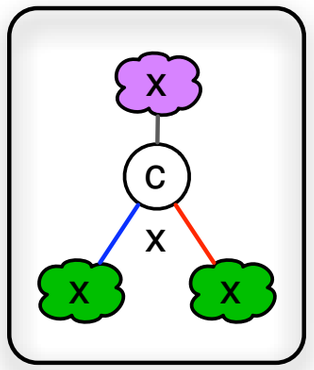


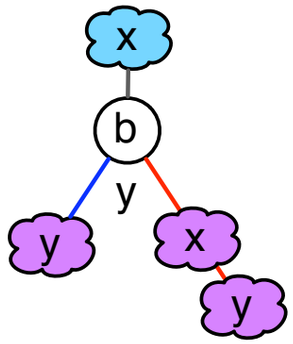
From words to trees

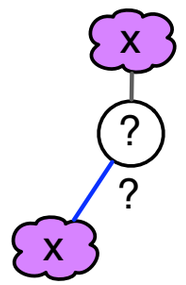
initial state

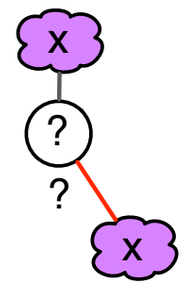
final state

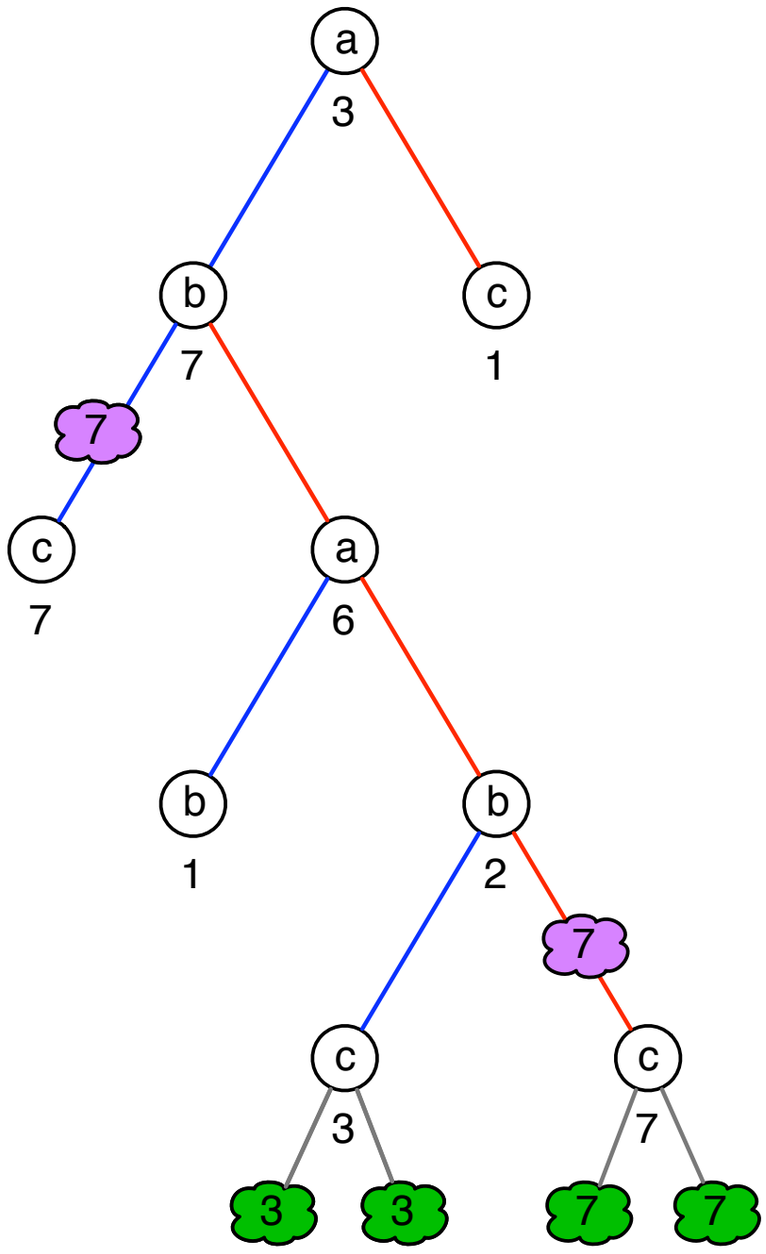








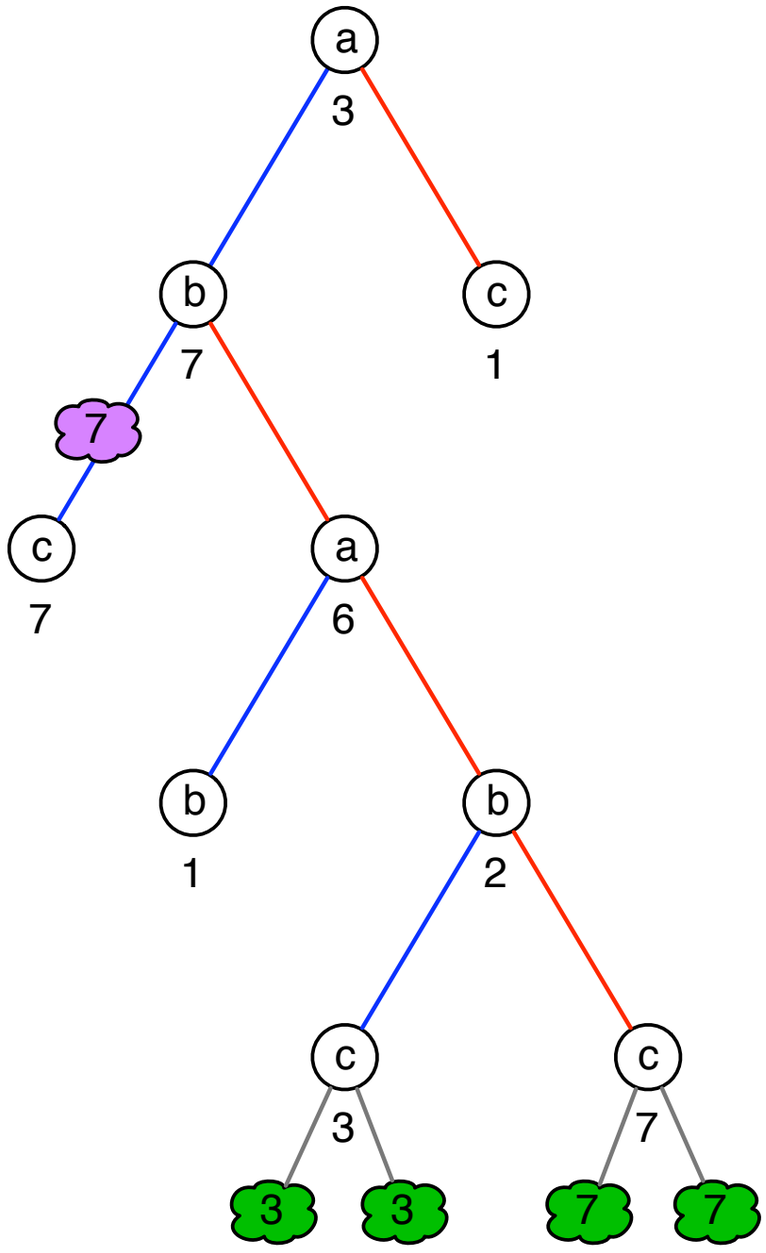


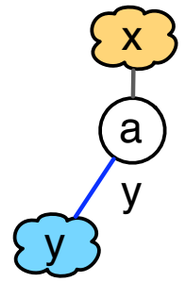


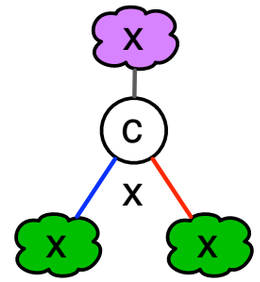
From words to trees

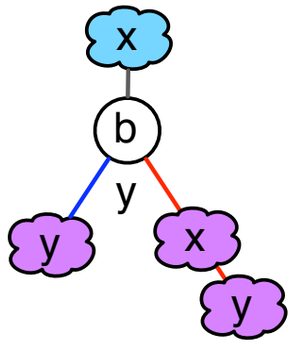
initial state

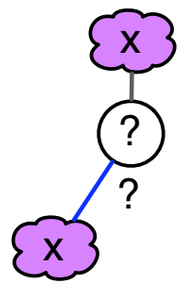
final state

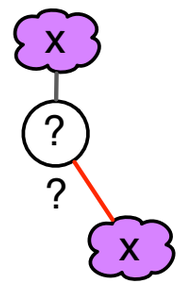








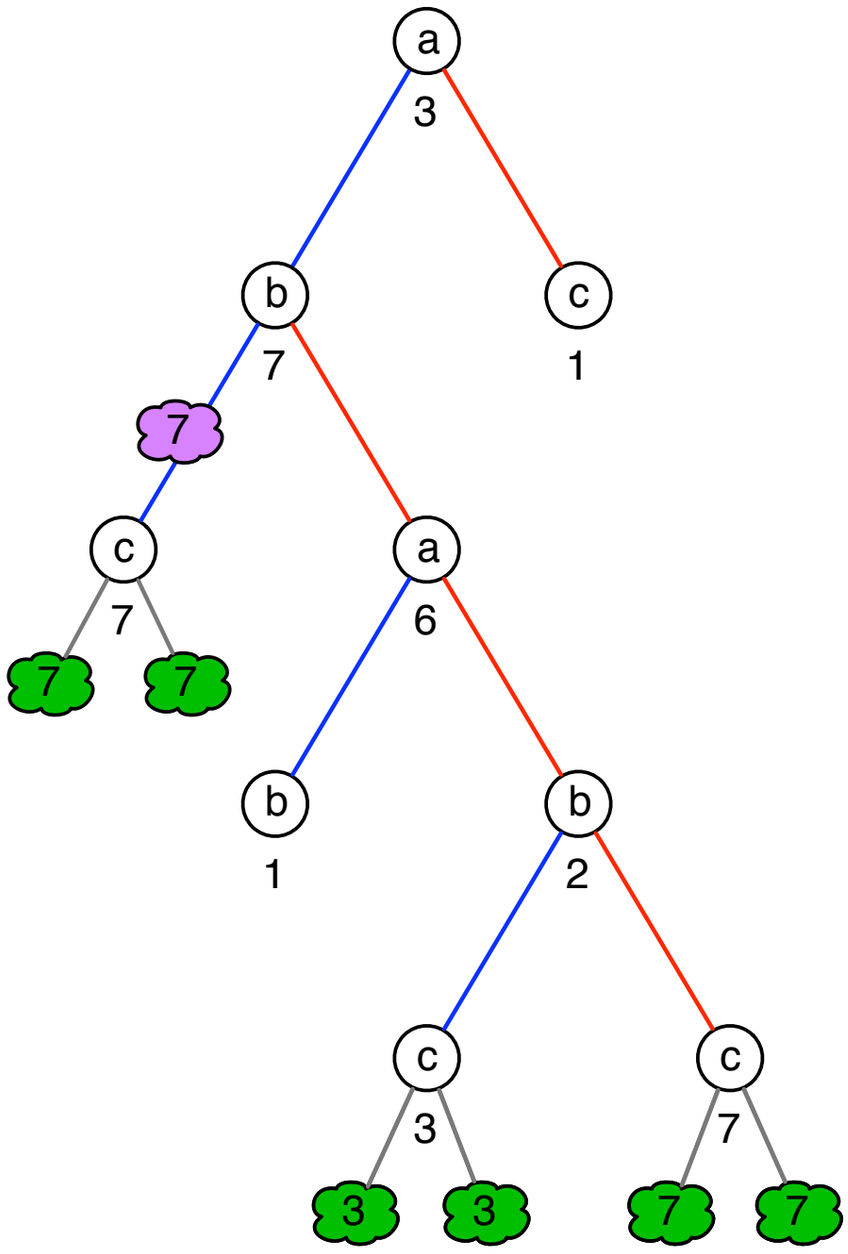


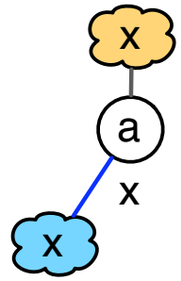


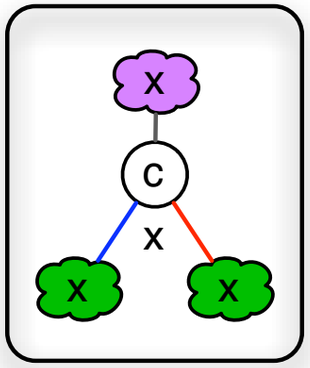
From words to trees

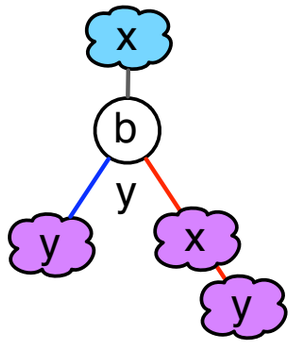
initial state

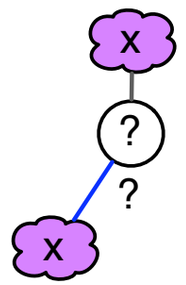
final state

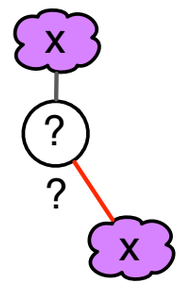








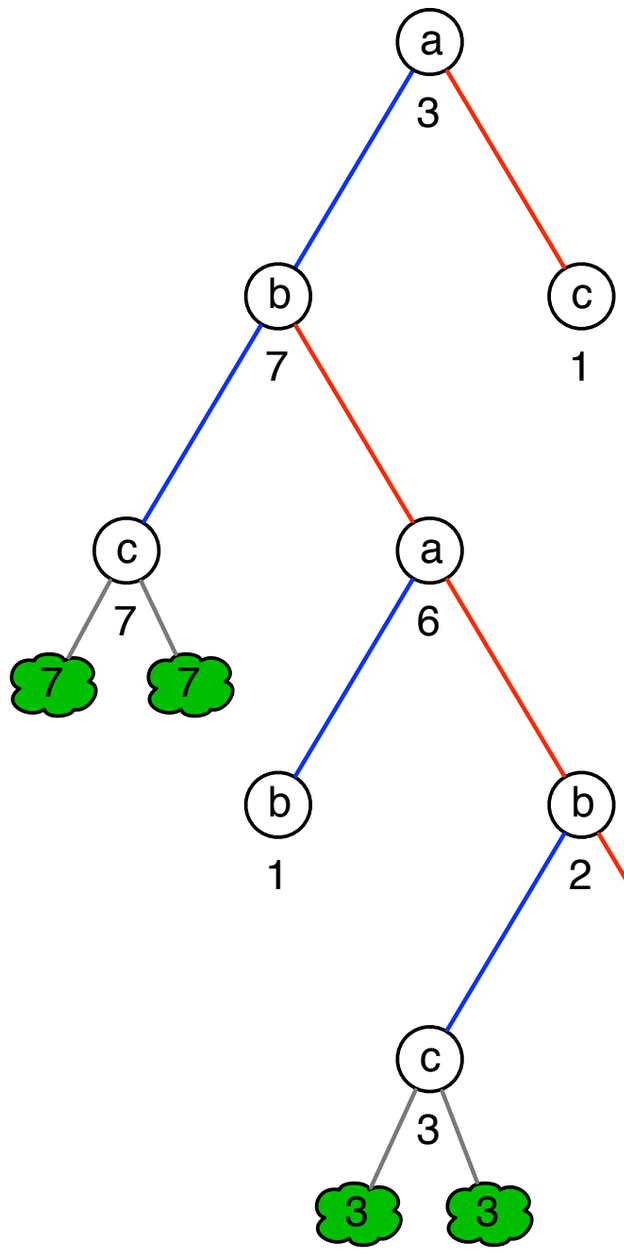


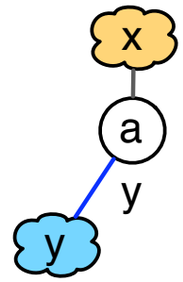


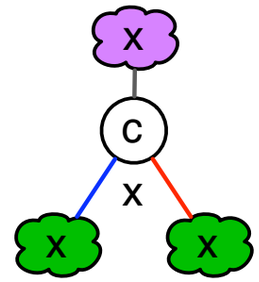
From words to trees

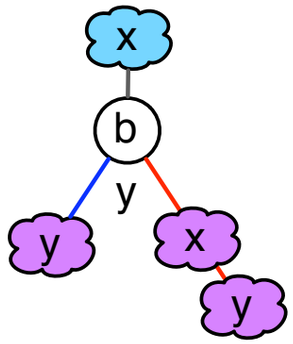
initial state

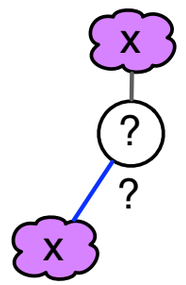
final state

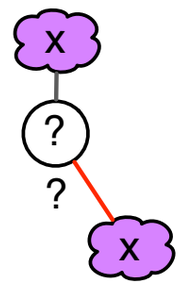








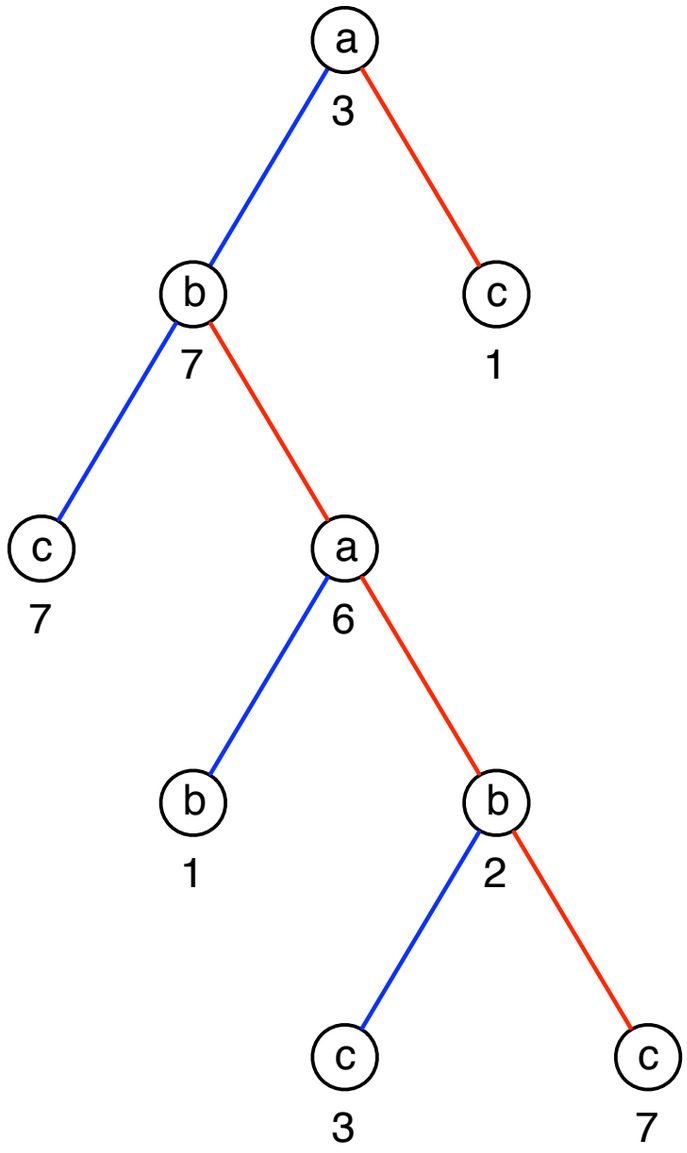


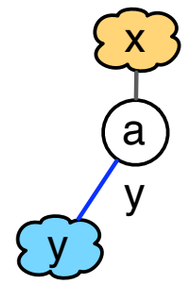


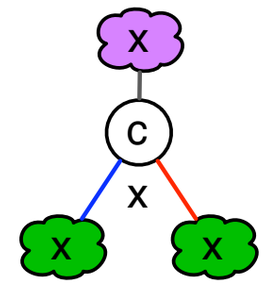
From words to trees

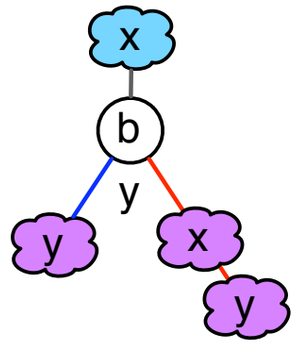
initial state

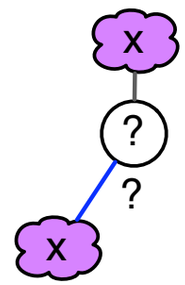
final state

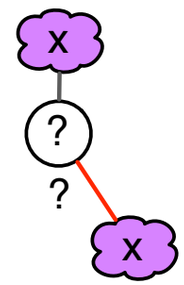








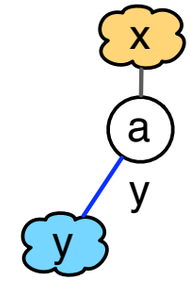


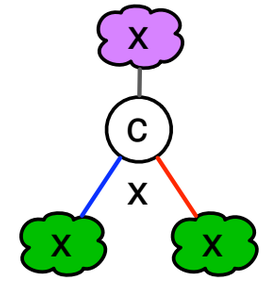


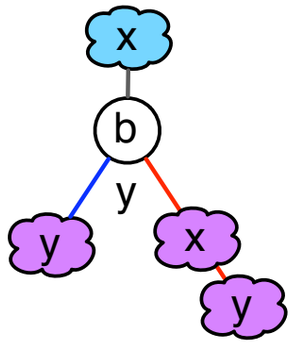
From words to trees

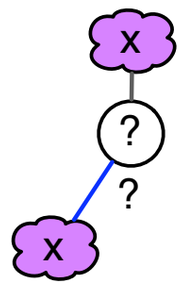
initial state

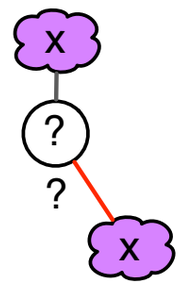
final state

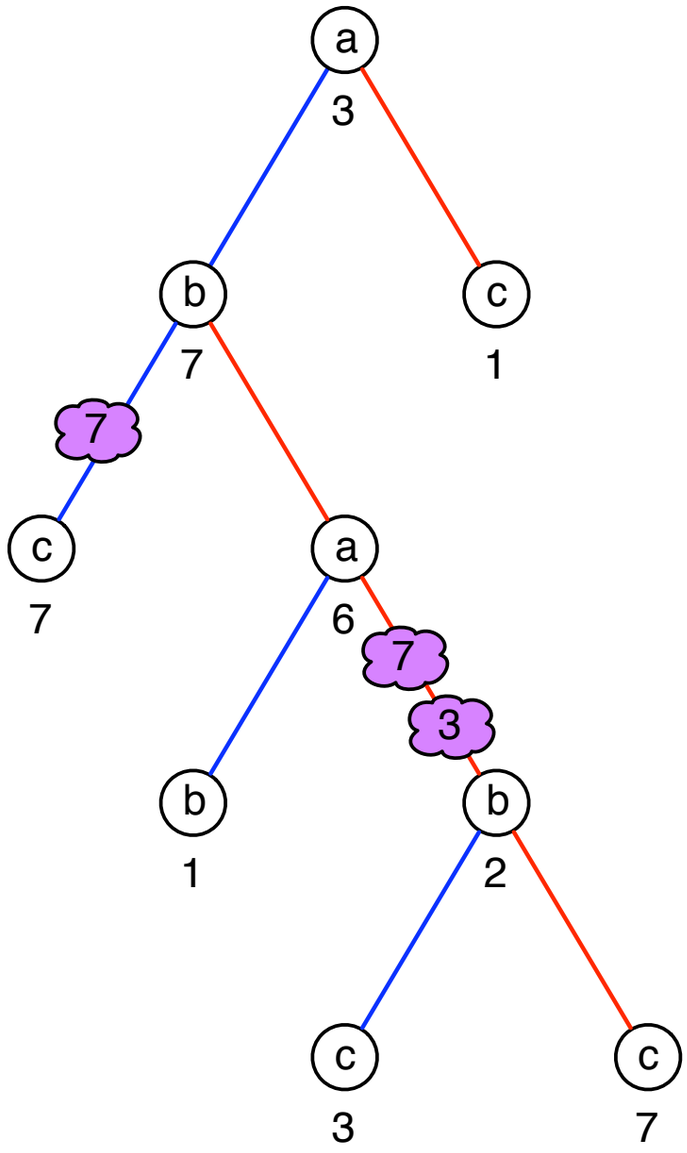


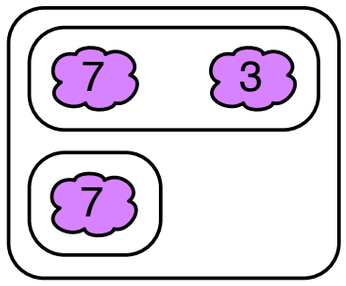








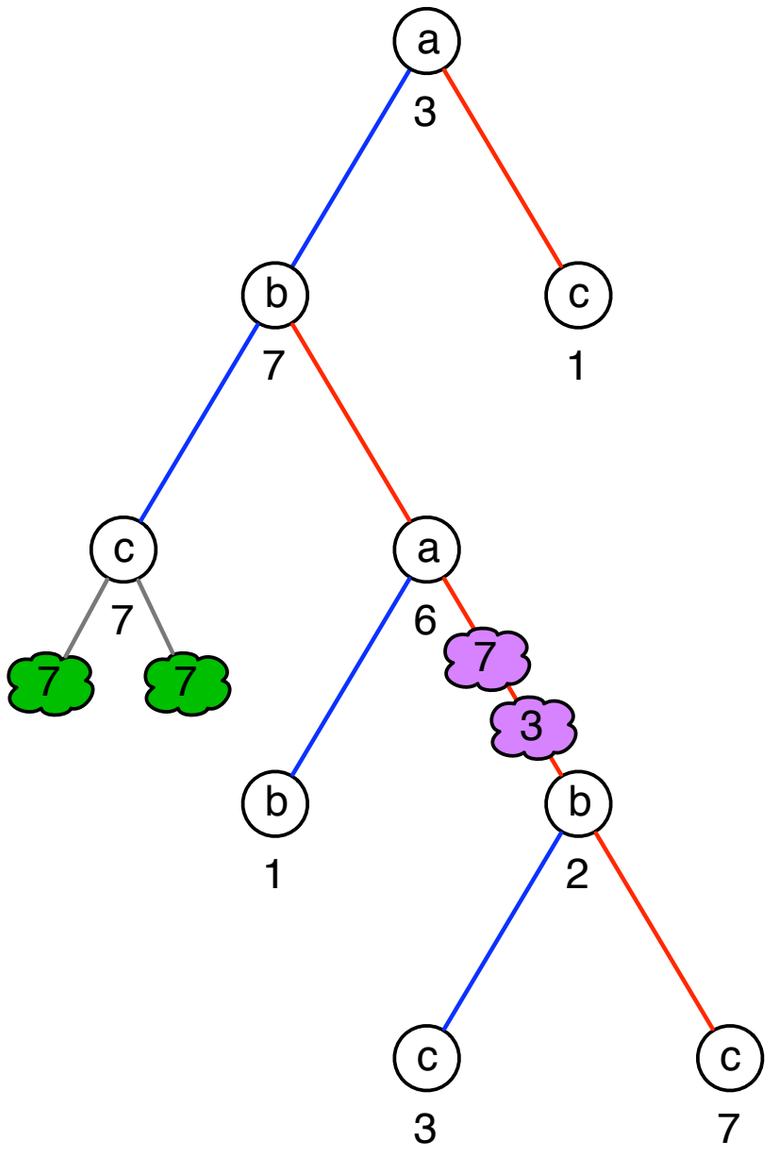


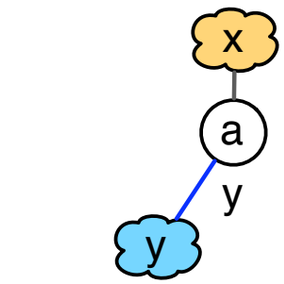


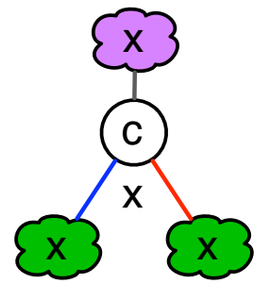
From words to trees

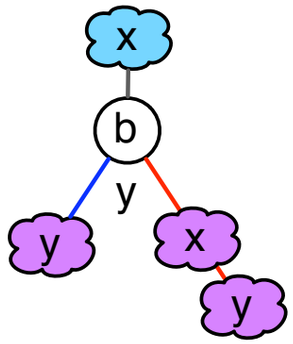
initial state

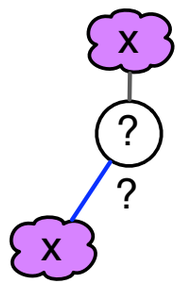
final state

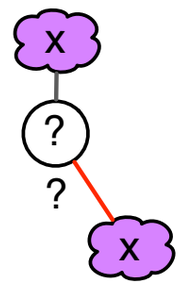


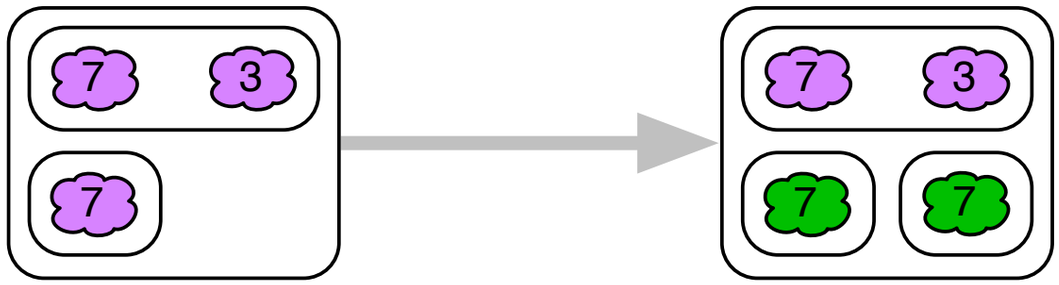








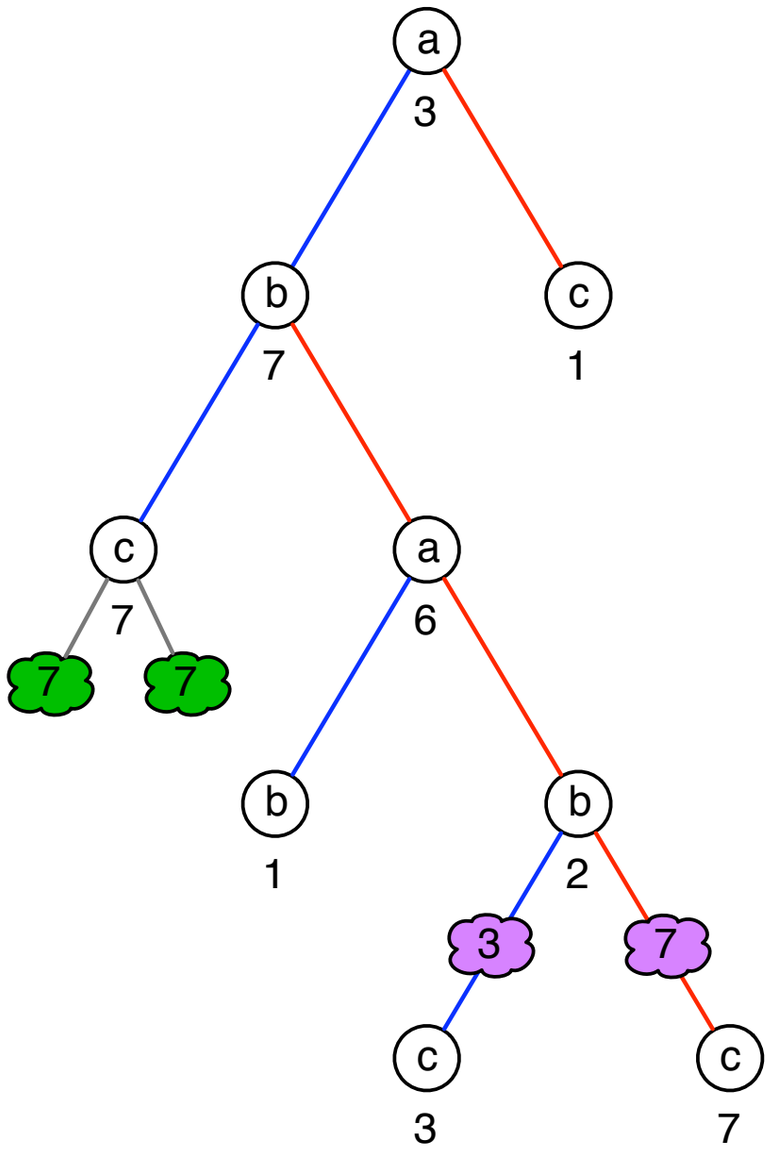


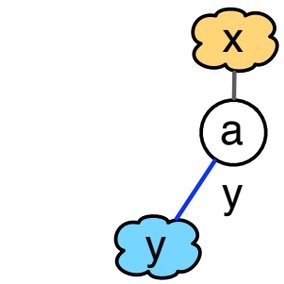


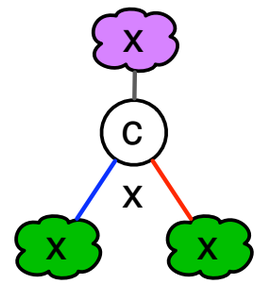
From words to trees

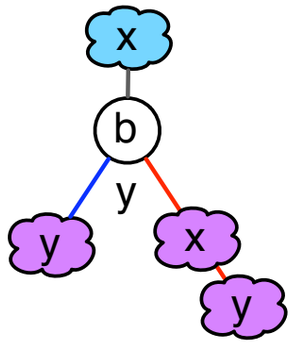
initial state

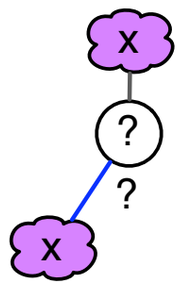
final state

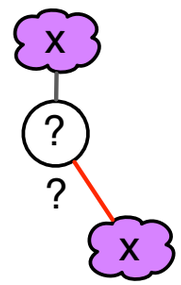


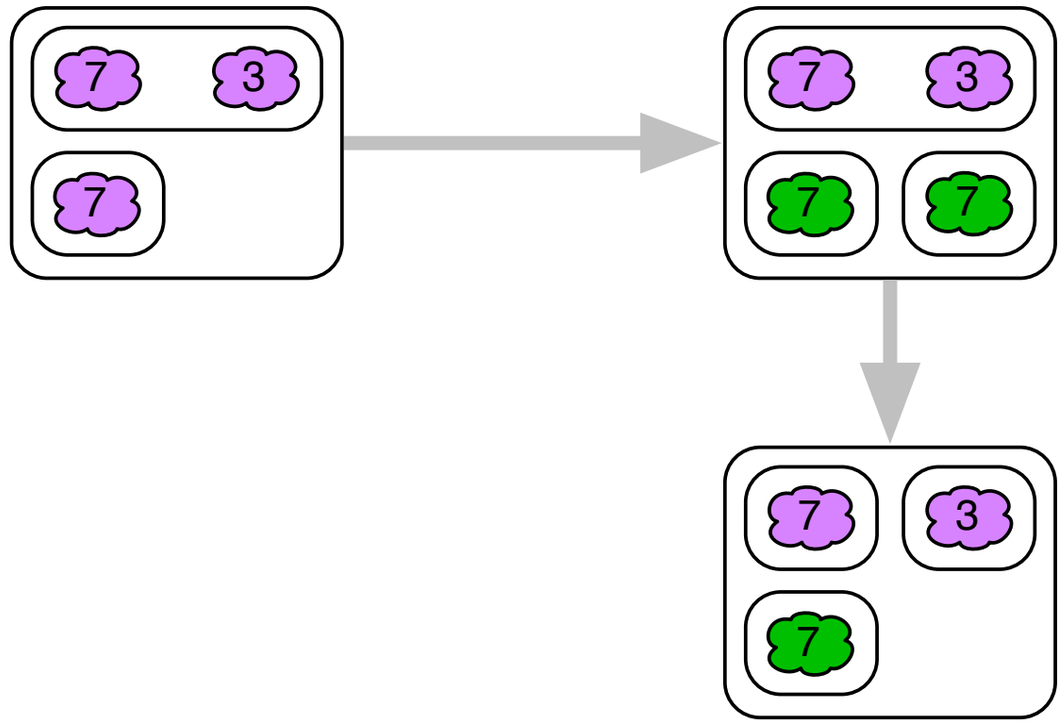








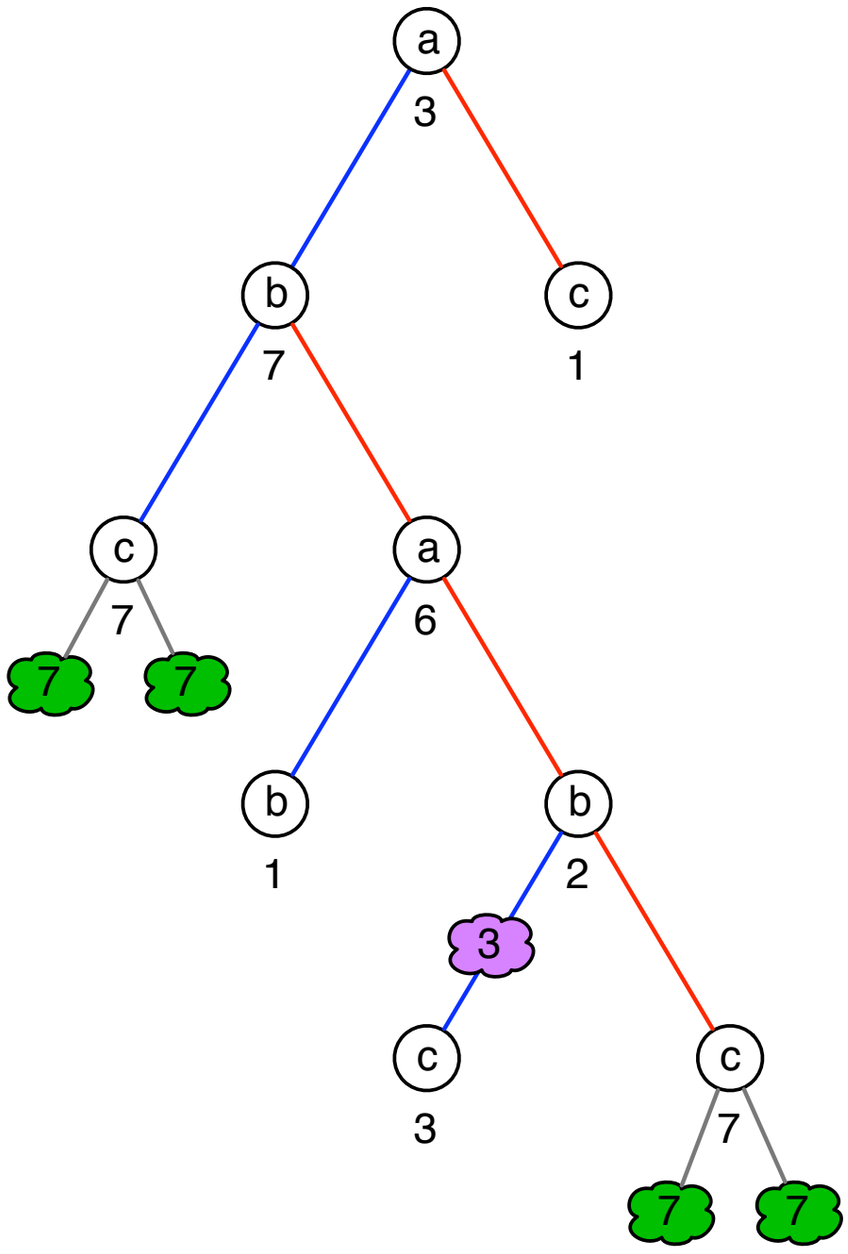


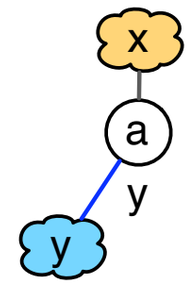


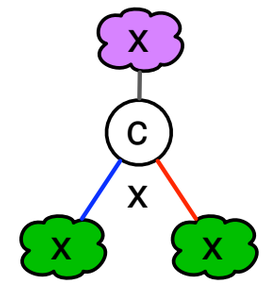
From words to trees

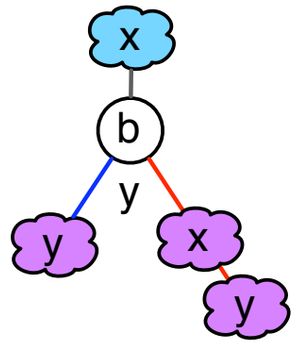
initial state

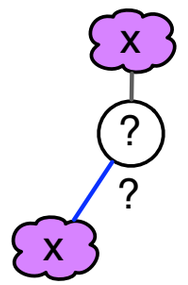
final state

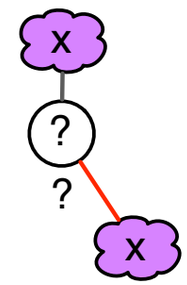


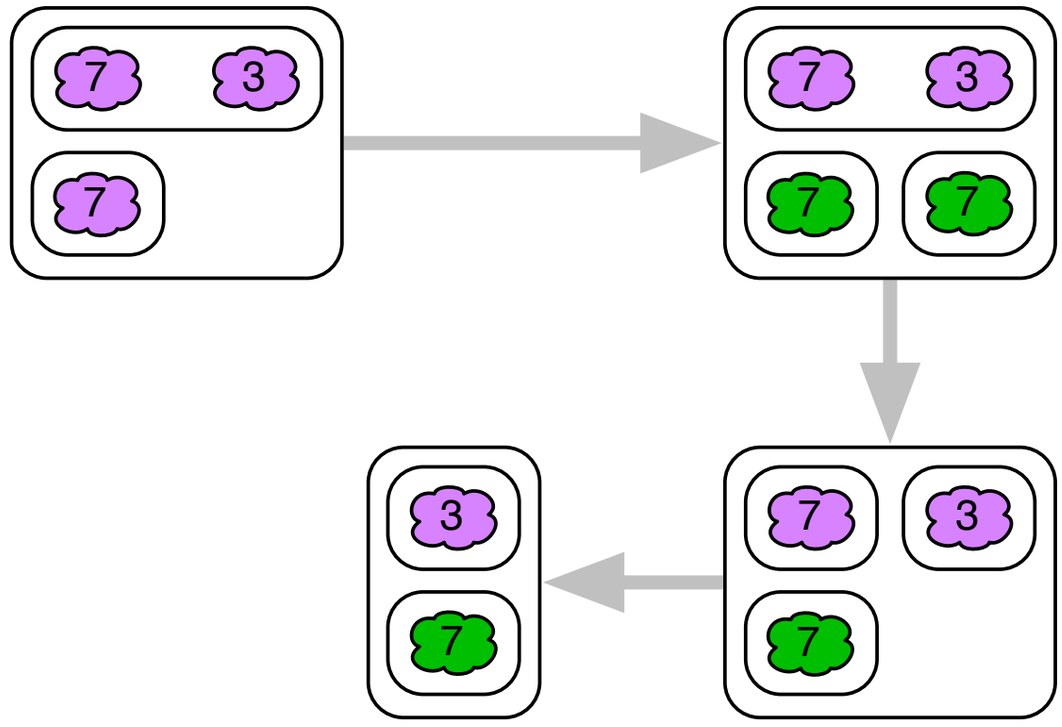








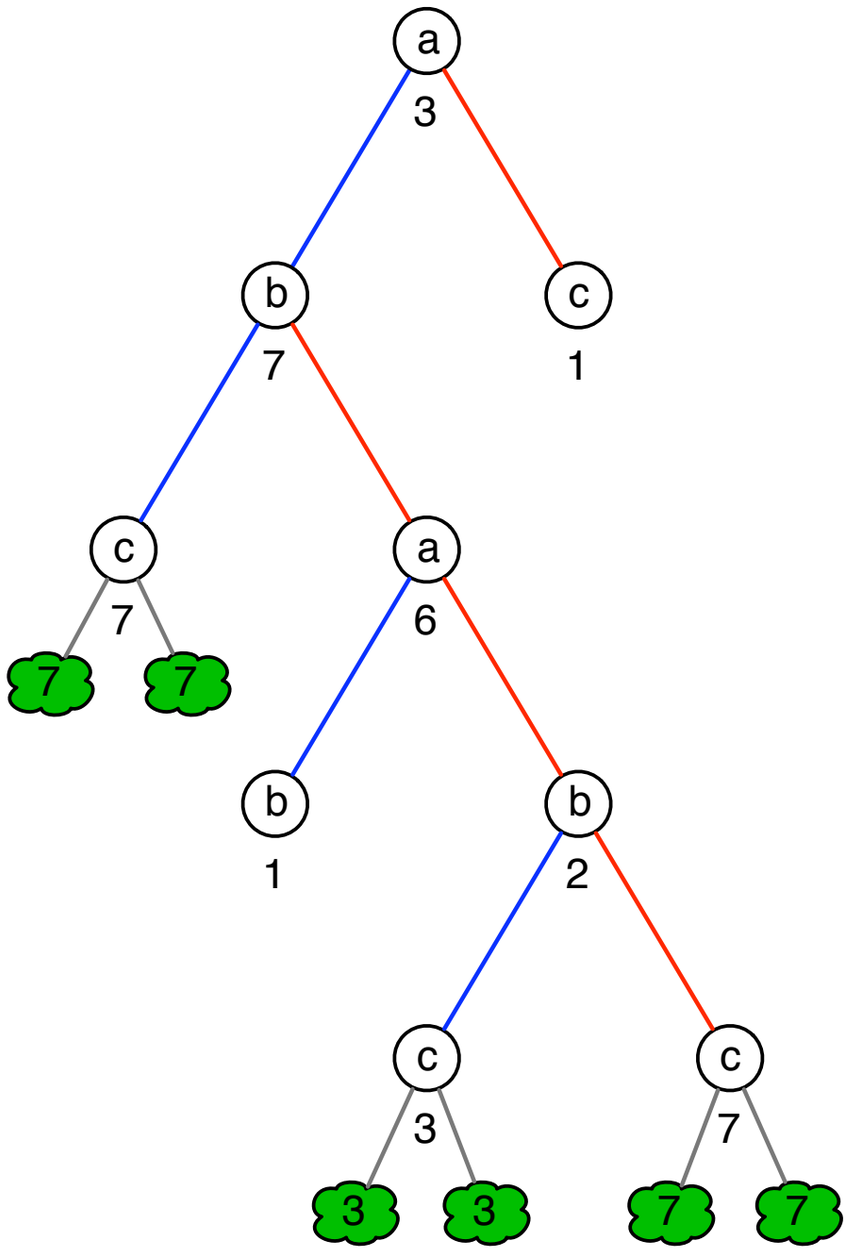


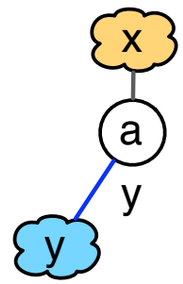


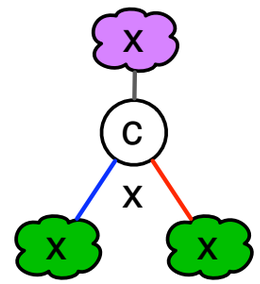
From words to trees

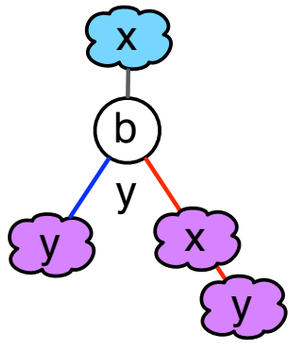
initial state

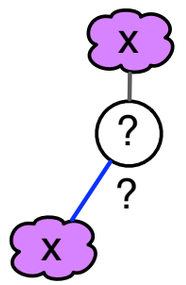
final state

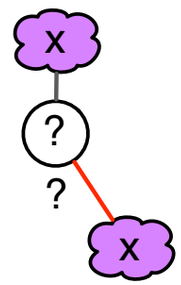


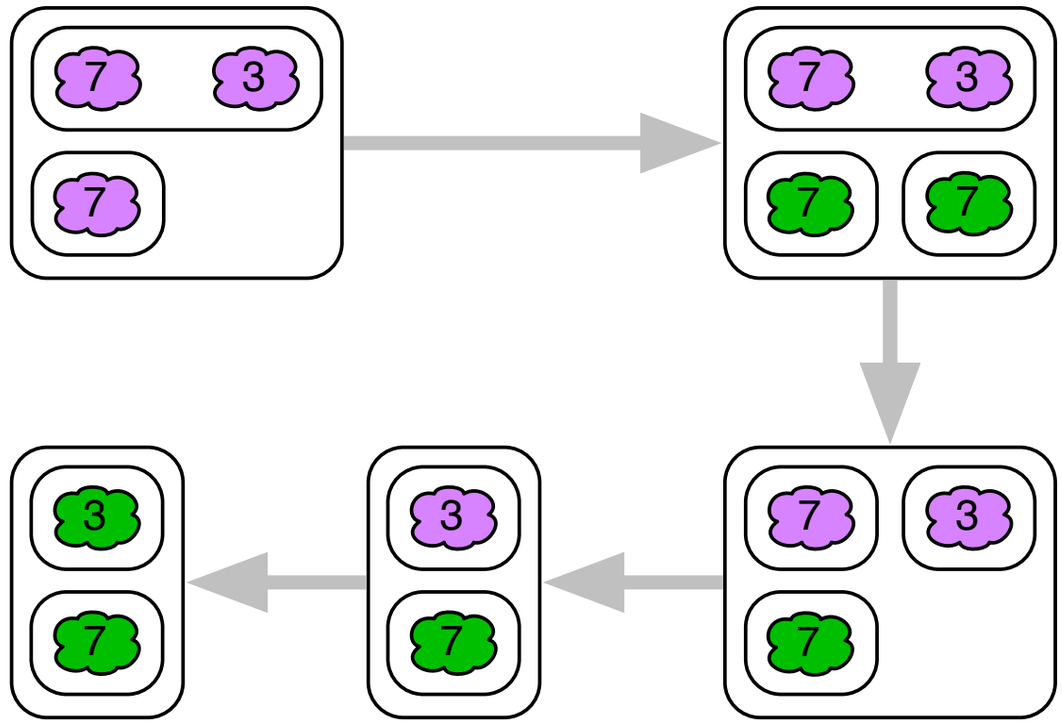








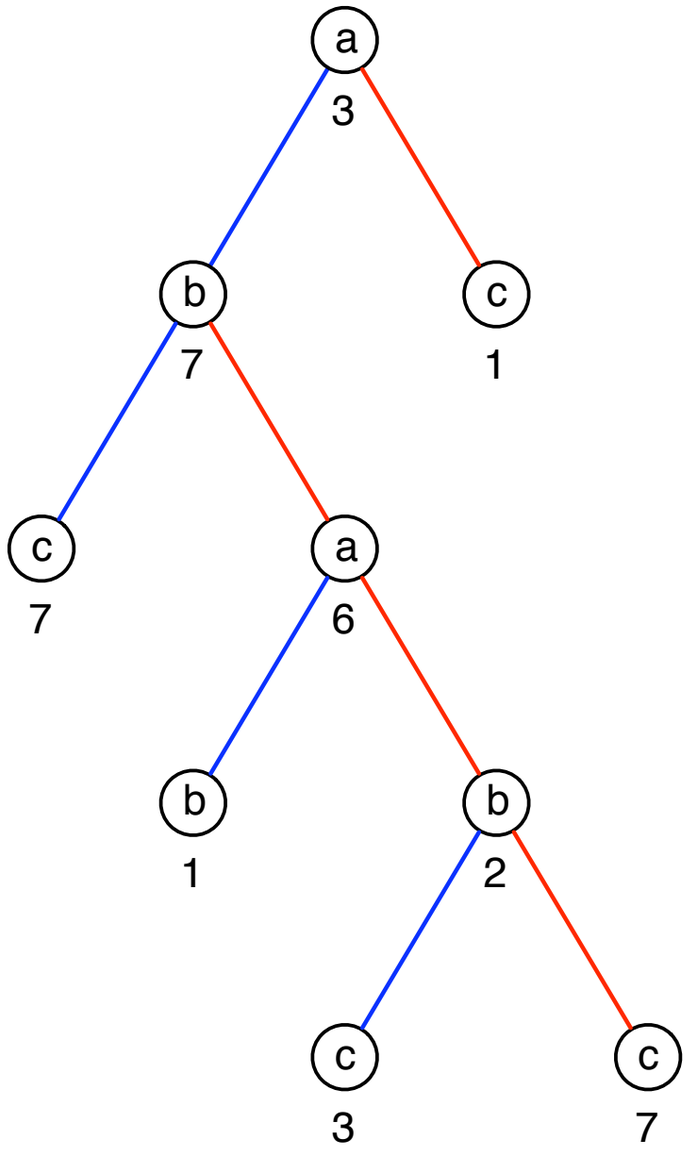


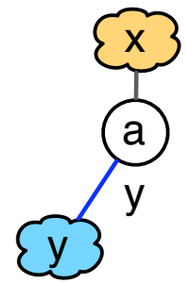


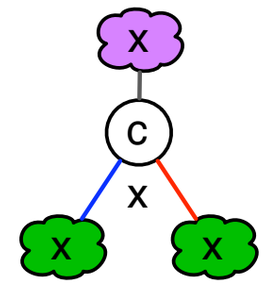
From words to trees

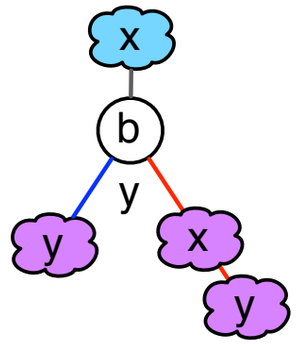
initial state

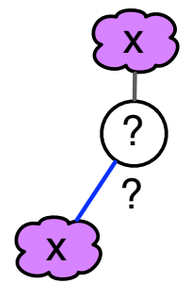
final state

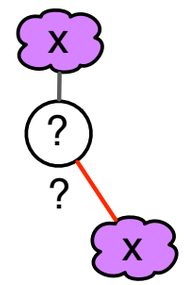


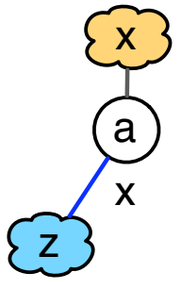


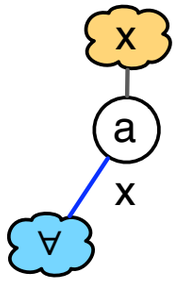




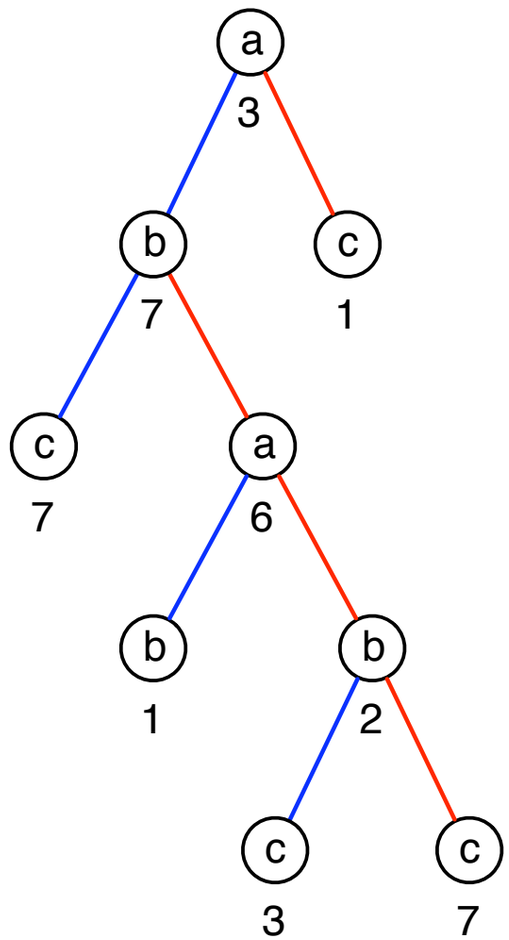


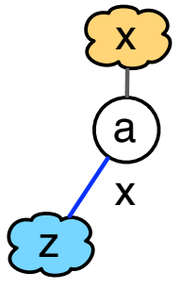


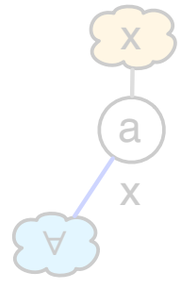
ATRA + 

guess + 

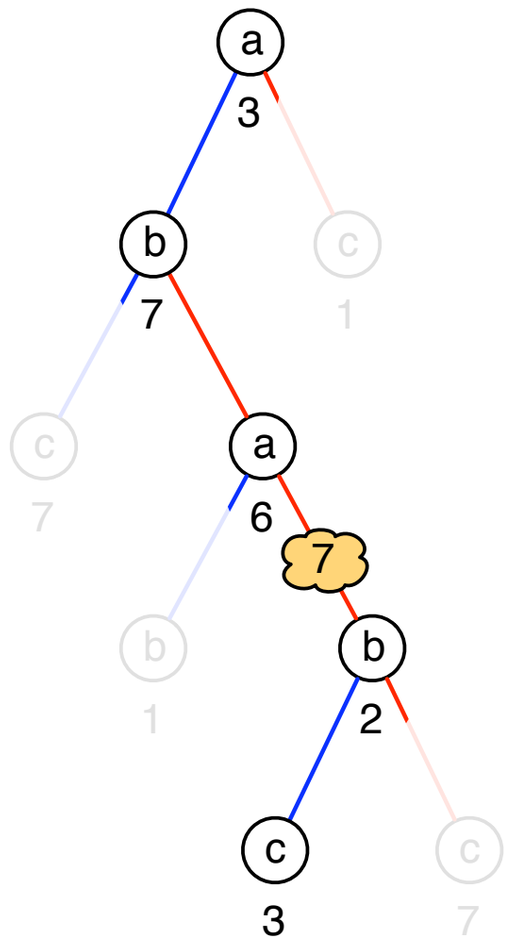
universal

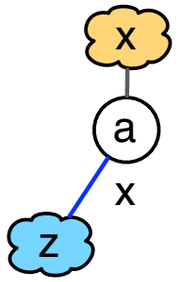


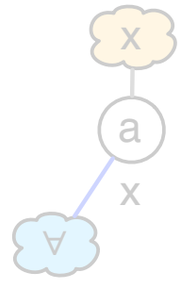
ATRA + 

guess + 

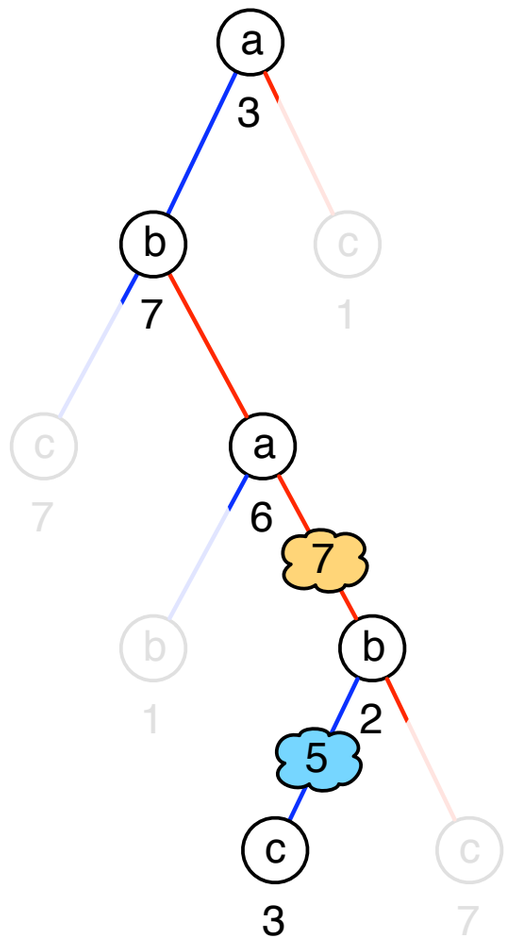
universal

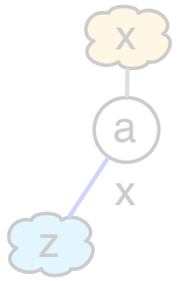


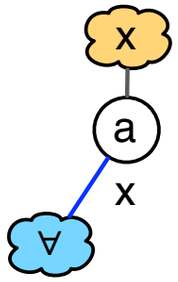
ATRA + 

guess + 

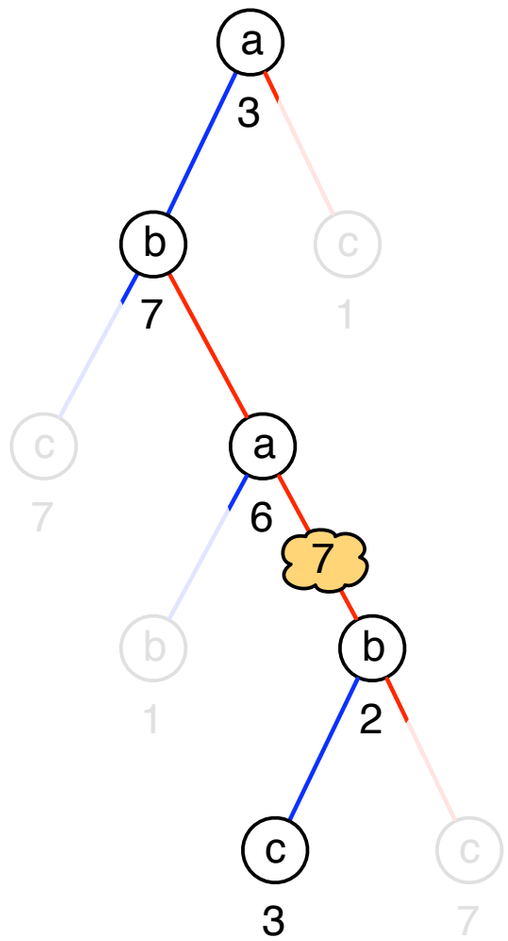
universal

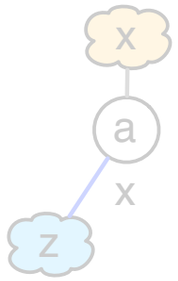


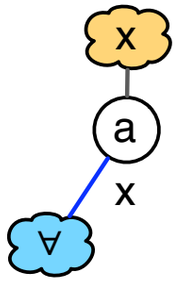
ATRA + 

guess + 

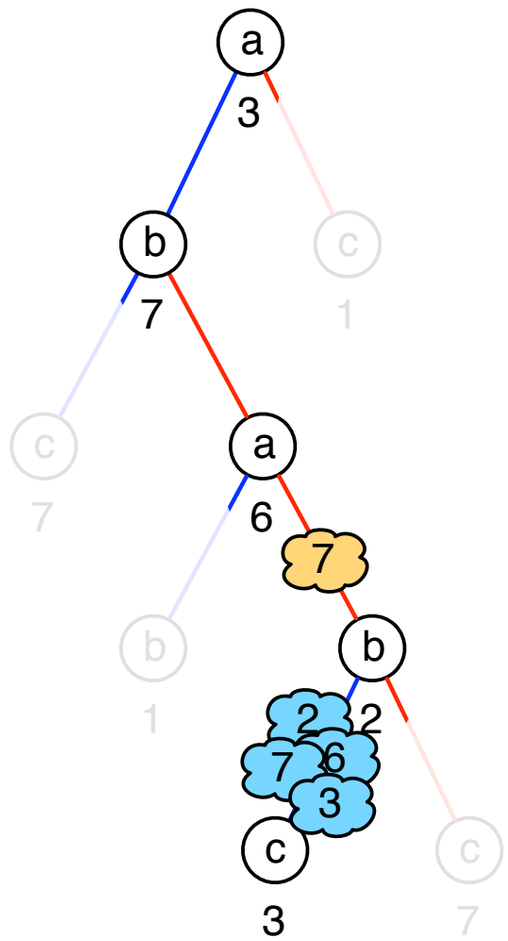
universal

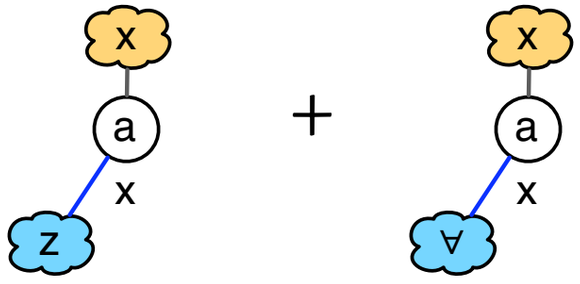


ATRA + 

guess + 

universal



ATRA + 

is decidable.

guess universal

☺ Still decidable emptiness problem

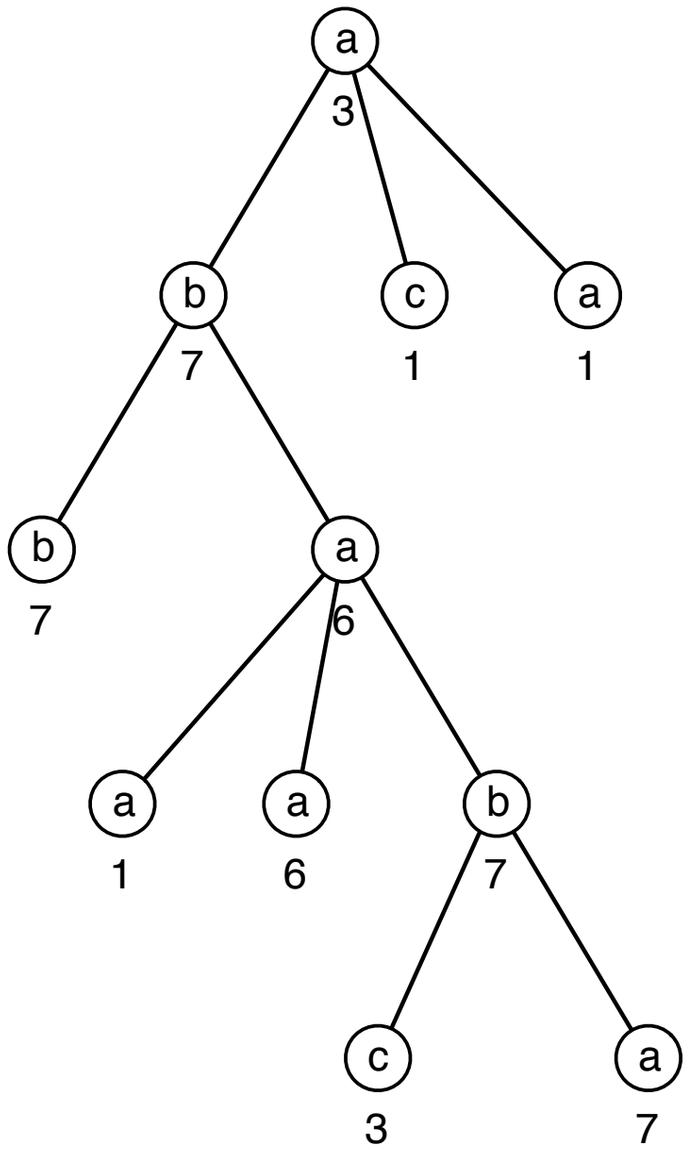
☹ No longer closed under complementation

☠ Can't be closed under complement preserving decidability

Reduction:

XPath ⇝ ATRA + guess + universal

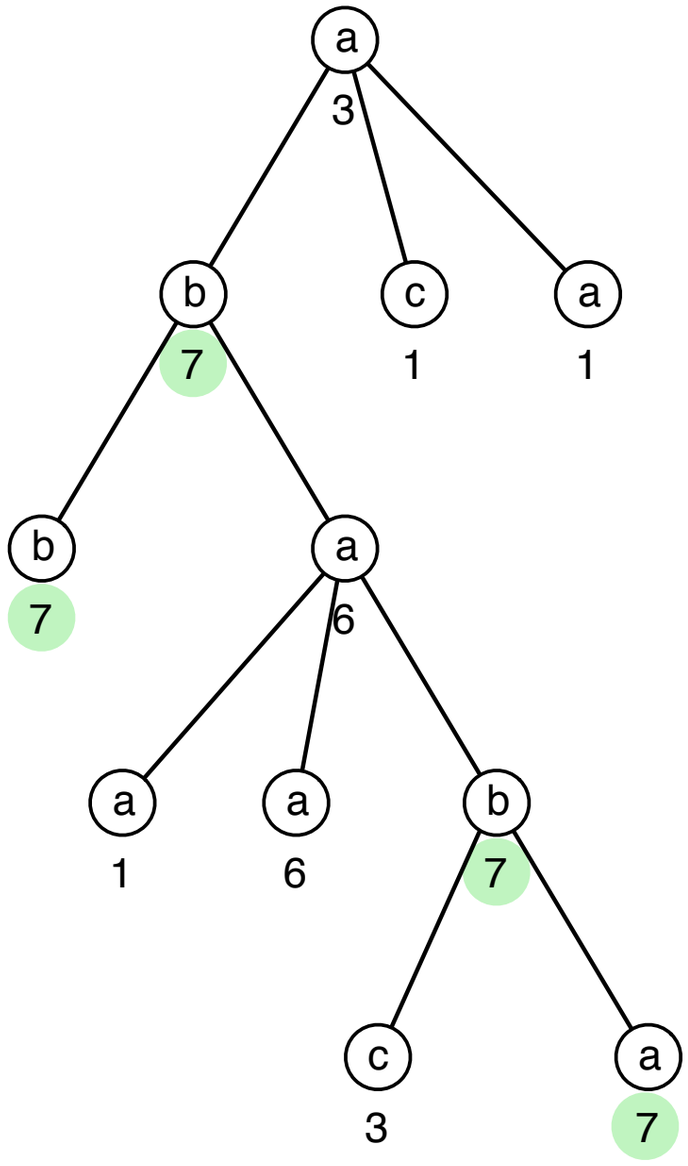
How do we code...?



⋮ [a] = ⋮ [b]

There are an ***a*** and a ***b*** with the same datum

How do we code...?

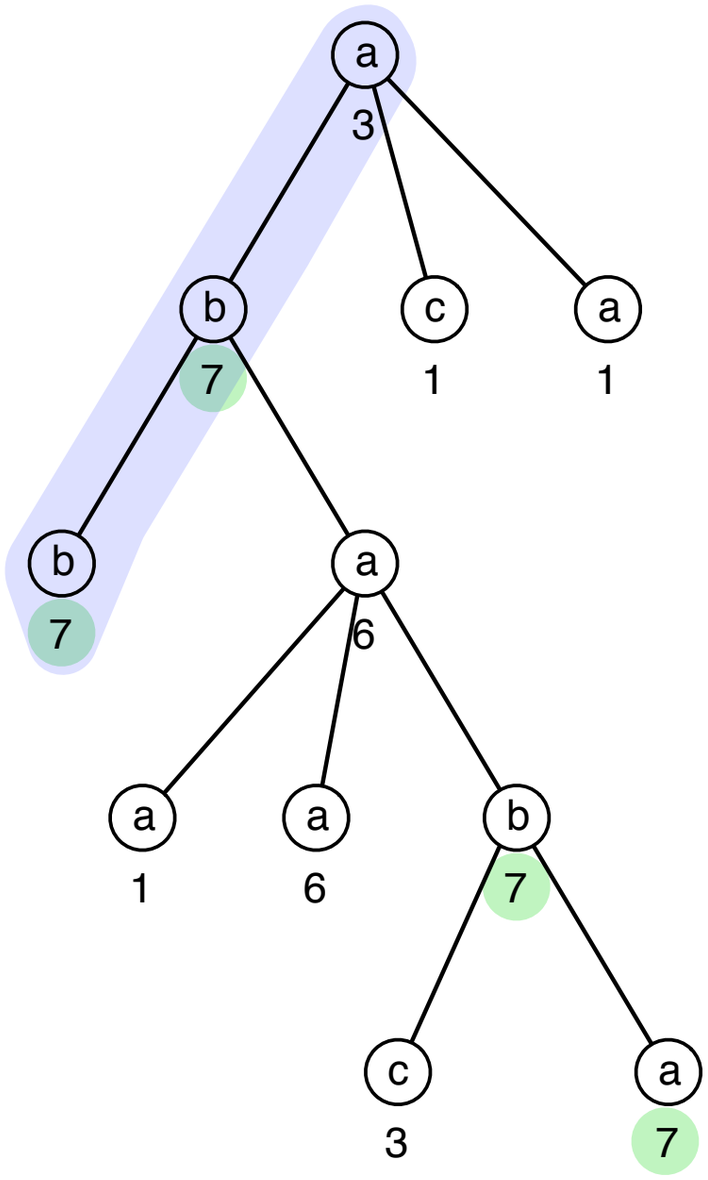


⋮[a] = ⋮[b]

Guess the data value 7.

There are an ***a*** and a ***b*** with the same datum

How do we code...?



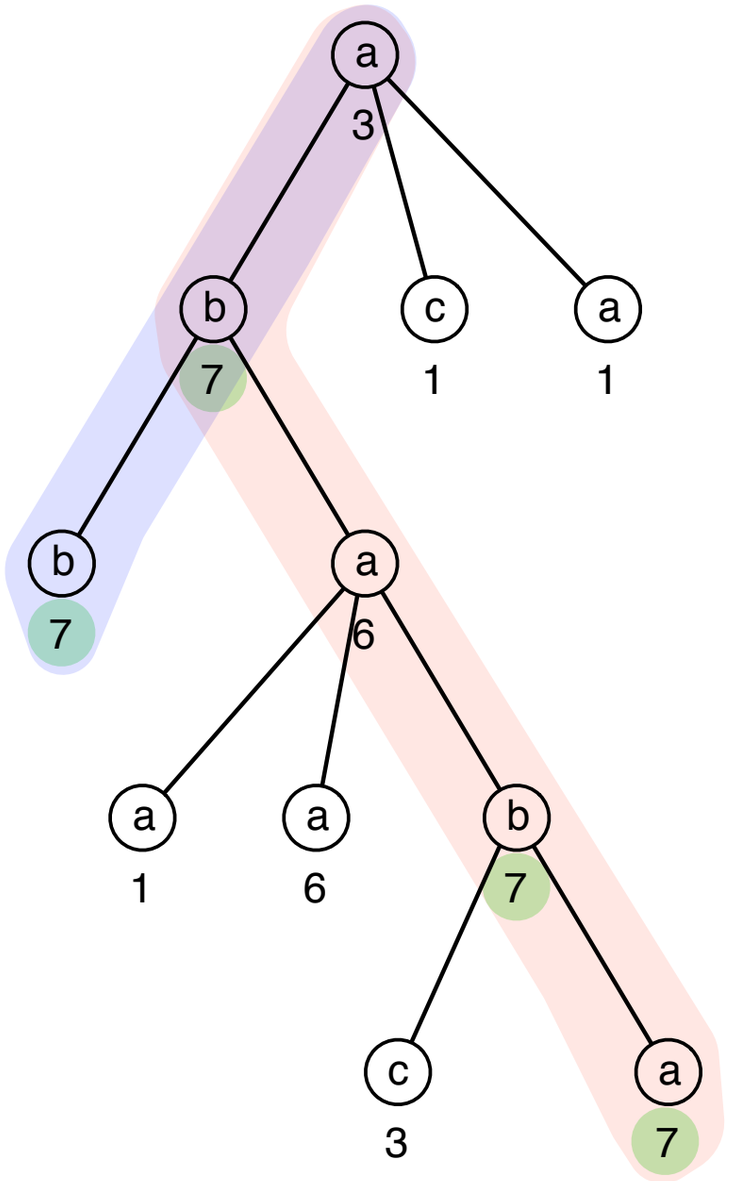
⋮ [a] = ⋮ [b]

There are an ***a*** and a ***b*** with the same datum

Guess the data value 7.

Check that it can be accessed with " ⋮ [b] ".

How do we code...?



⋮ [a] = ⋮ [b]

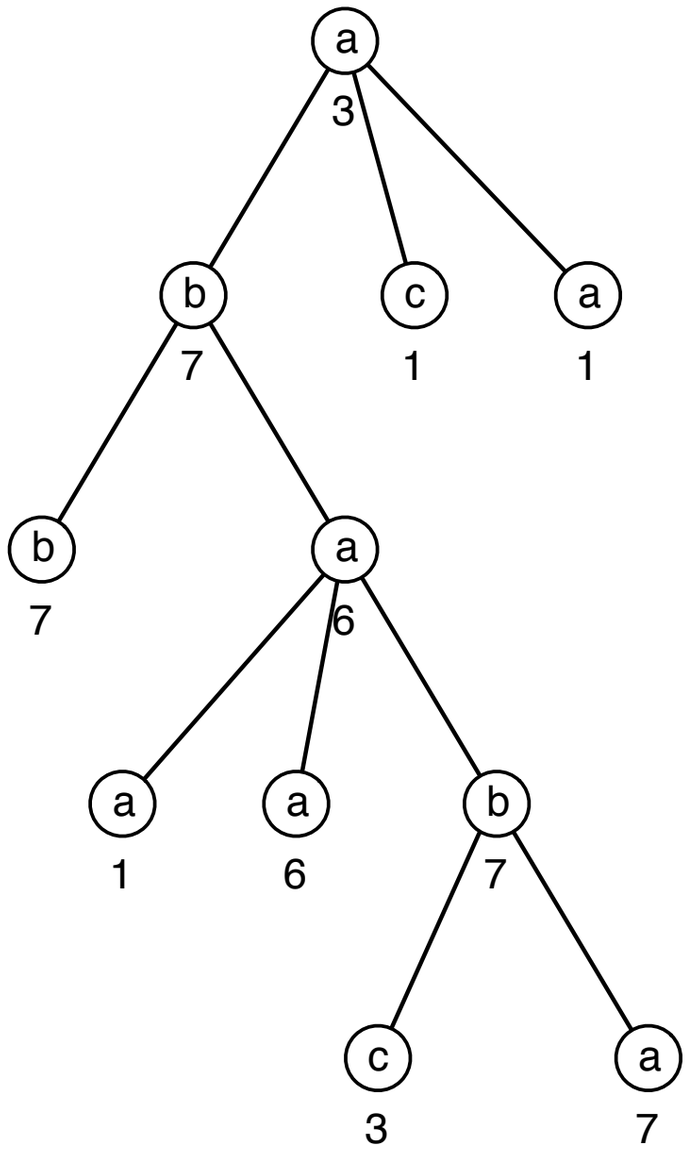
There are an ***a*** and a ***b*** with the same datum

Guess the data value 7.

Check that it can be accessed with " ⋮ [b] ".

Check that it can be accessed with " ⋮ [a] ".

How do we code...?



$\vdots [a] = \vdots [b]$

$\vdots [c] \neq \vdots [c]$

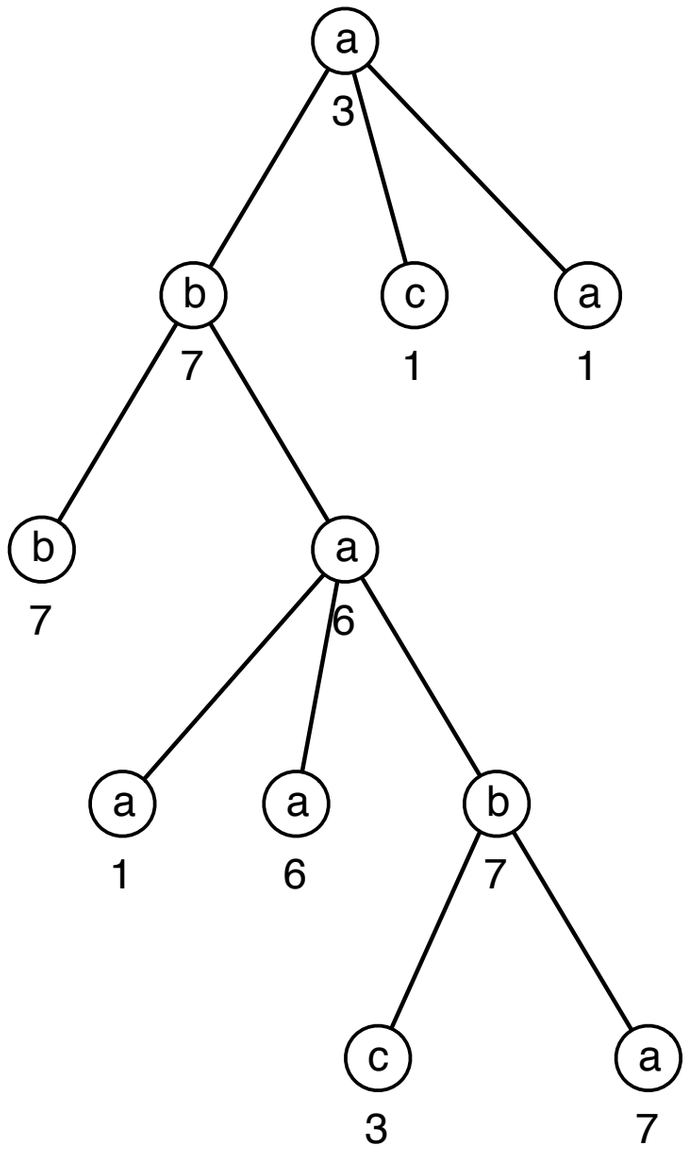
$\neg (\vdots [b] \neq \vdots [b])$

There are an ***a*** and a ***b*** with the same datum

There are two ***c*** with different data values

All ***b*** have the same data value

How do we code...?



$\vdots[a] = \vdots[b]$

There are an ***a*** and a ***b*** with the same datum

$\vdots[c] \neq \vdots[c]$

There are two ***c*** with different data values

$\neg (\vdots[b] \neq \vdots[b])$

All ***b*** have the same data value

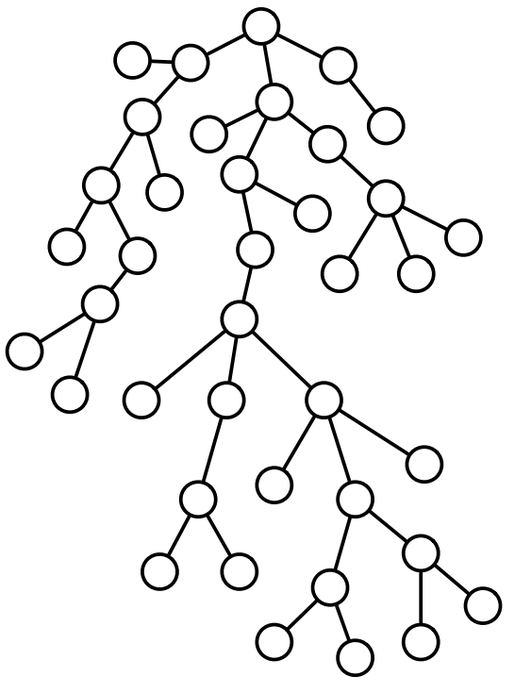
$\neg (\vdots[b] = \vdots[c])$

There is no data value shared by a ***b*** and a ***c***

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

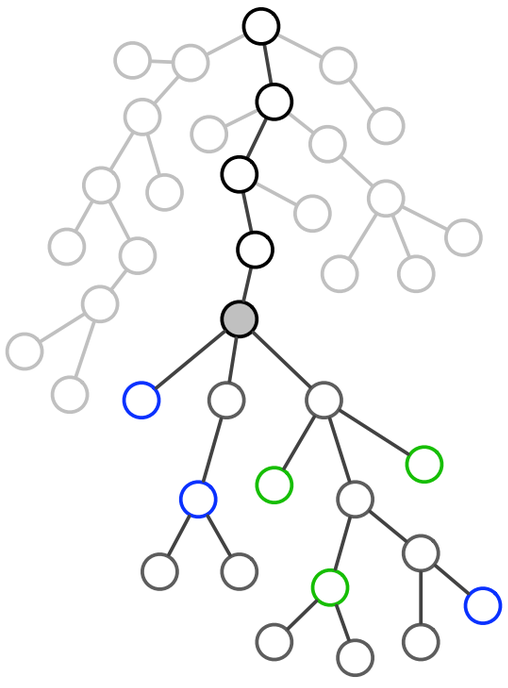
There is no data value shared by a ***b*** and a ***c***



How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***

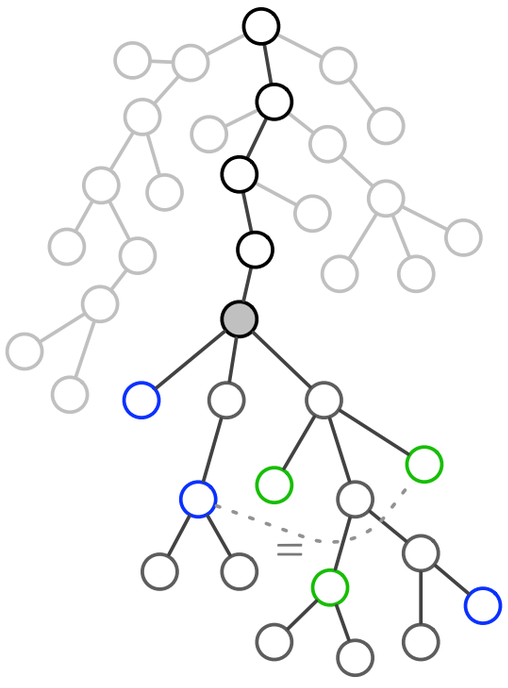


If it is unsatisfiable: there was a α=β before.
Otherwise, assume ≠ data values.

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***

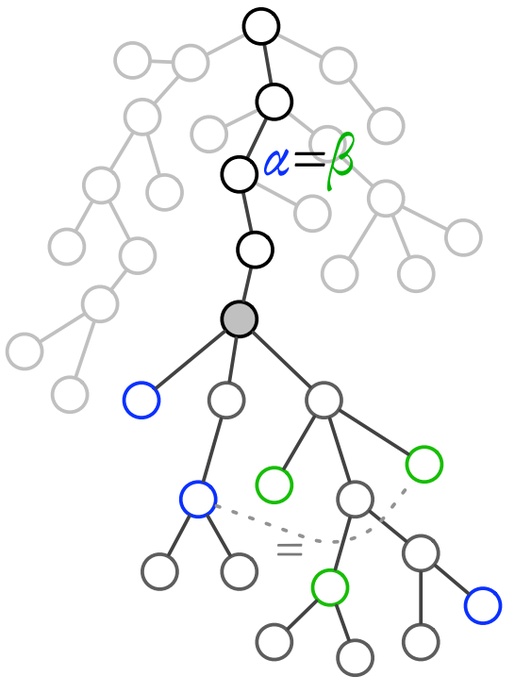


If it is unsatisfiable: there was a α=β before.
Otherwise, assume ≠ data values.

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***

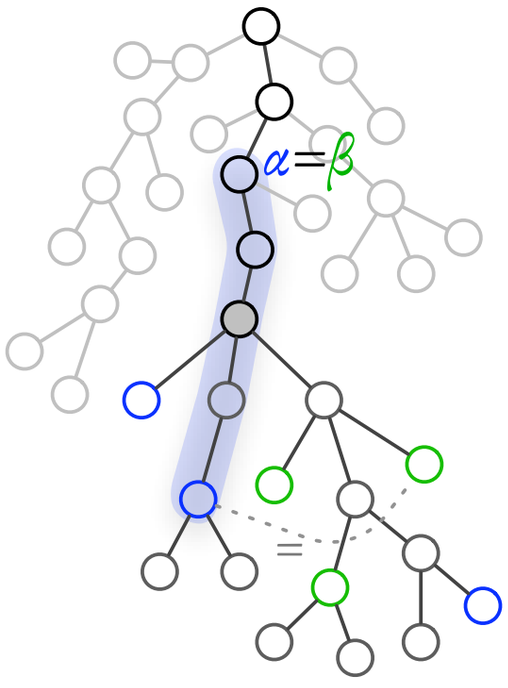


If it is unsatisfiable: there was a $\alpha=\beta$ before.
Otherwise, assume ≠ data values.

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***

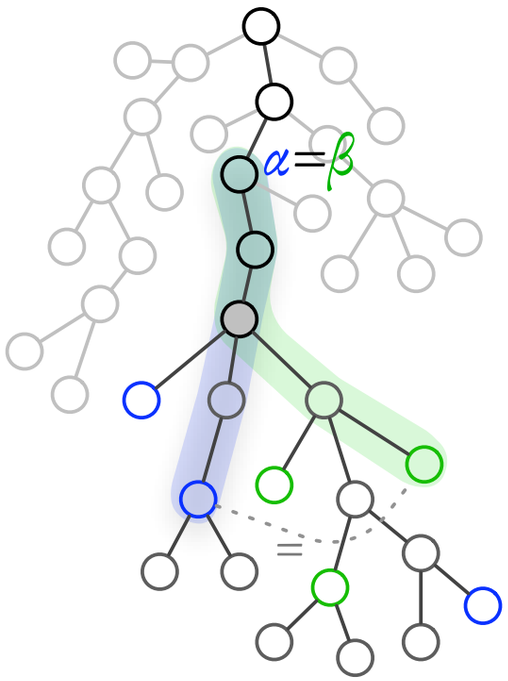


If it is unsatisfiable: there was a $\alpha=\beta$ before.
Otherwise, assume ≠ data values.

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***

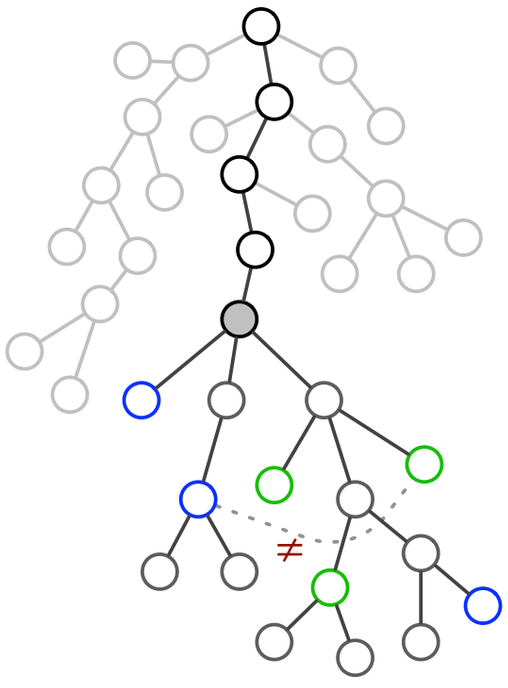


If it is unsatisfiable: there was a $\alpha=\beta$ before.
Otherwise, assume ≠ data values.

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***

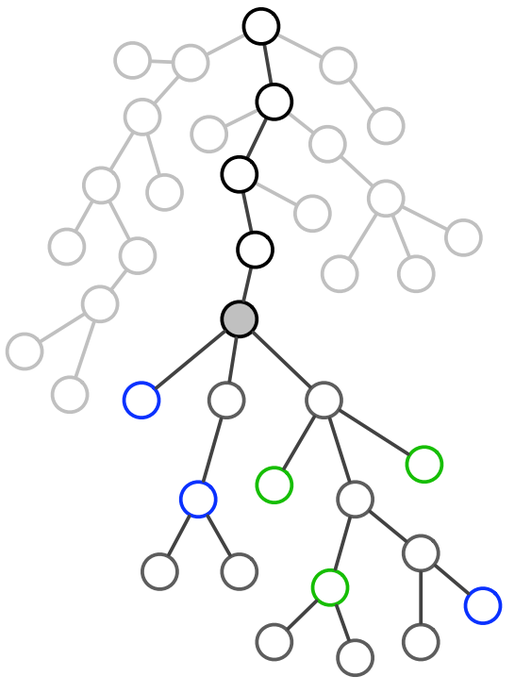


If it is unsatisfiable: there was a α=β before.
Otherwise, assume ≠ data values.

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***



If it is unsatisfiable: there was a α=β before.
Otherwise, assume ≠ data values.

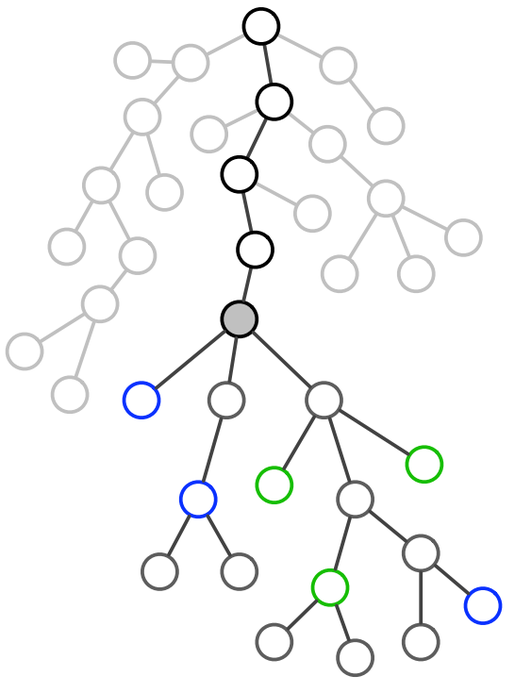
For all data values considered in the run:

* we cannot reach ⋮ [**b**] with the value, or
* we cannot reach ⋮ [**c**] with the value.

How do we code...?

$\neg$ (⋮ [**b**] = ⋮ [**c**])

There is no data value shared by a ***b*** and a ***c***



If it is unsatisfiable: there was a α=β before.
Otherwise, assume ≠ data values.

For all data values considered in the run:

* we cannot reach ⋮ [**b**] with the value, or
* we cannot reach ⋮ [**c**] with the value.

Spread: for all data values that
were *read* in the word, or
were *guessed* by some previous configuration

Satisfiability of XPath

NP

Positive-XPath ♠

ExpTime

Downward ♥
XPath($\downarrow, \downarrow_*$)

XPath without data tests ♠

Decidable, non-PR

Forward ♦
XPath($\downarrow, \downarrow_*, \rightarrow, \rightarrow^*$)

Vertical ♣
XPath($\downarrow, \downarrow_*, \uparrow, \uparrow^*$)

Undecidable

full XPath ♠

♥ [F. PODS'09]
♦ [F. ICDT'10]
♣ [F., Segoufin, STACS'10]
♠ [Geerts, Fan, PODS'05]

Vertical XPath

$\mathsf{XPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*)$ is decidable, non-primitive recursive ♣

♣ [Figueira, Segoufin, STACS 2011]

Vertical XPath

$\mathsf{XPath}(\downarrow, \downarrow_*, \uparrow, \uparrow^*)$ is decidable, non-primitive recursive ♣

+ unary key constraints : undecidable
+ regular languages / DTDs : undecidable
+ unary foreign key constraints : decidable

♣ [Figueira, Segoufin, STACS 2011]

XPath on data words

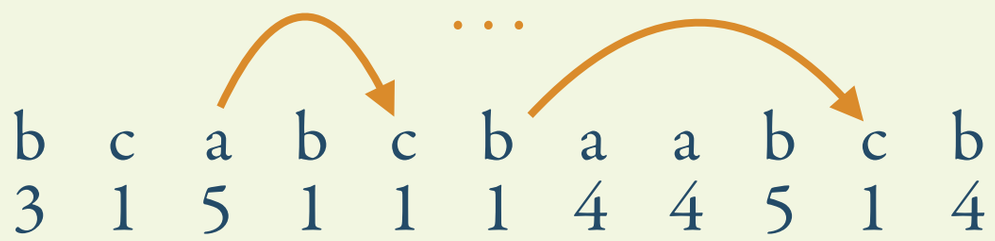
b	c	a	b	c	b	a	a	b	c	b
3	1	5	1	1	1	4	4	5	1	4

XPath on data words

b	c	a	b	c	b	a	a	b	c	b
3	1	5	1	1	1	4	4	5	1	4

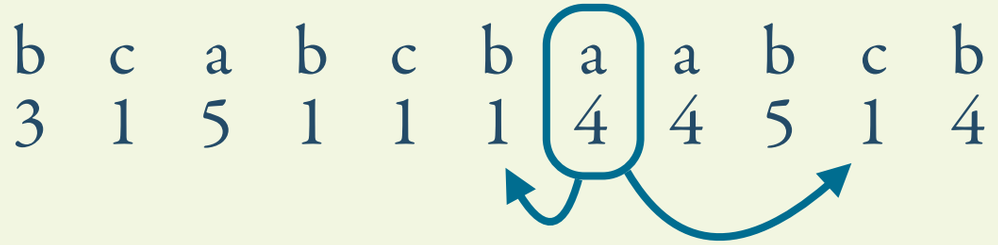
eg: $\longrightarrow *[\mathbf{b}] \longrightarrow [\mathbf{c}]$

XPath on data words



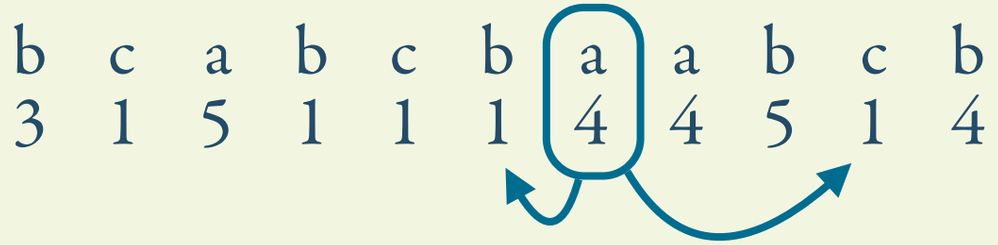
eg: $\longrightarrow *[\mathbf{b}] \longrightarrow [\mathbf{c}]$

XPath on data words



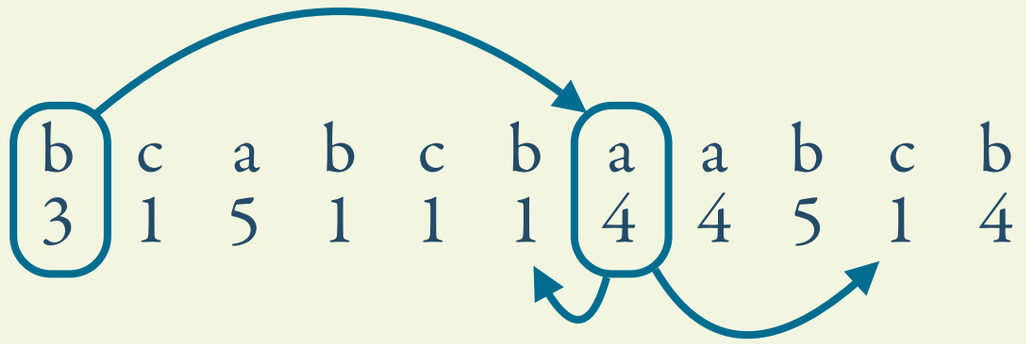
eg: $\langle \longleftarrow [\mathbf{b}] = \longrightarrow^* [\mathbf{b}] \longrightarrow [\mathbf{c}] \rangle$

XPath on data words



$$\textit{eg:} \langle \longrightarrow^*[\mathbf{a} \wedge \langle \longleftarrow[\mathbf{b}] = \longrightarrow^*[\mathbf{b}] \longrightarrow [\mathbf{c}] \rangle$$

XPath on data words



eg: $\langle \longrightarrow^*[\mathbf{a} \wedge \langle \longleftarrow[\mathbf{b}] = \longrightarrow^*[\mathbf{b}] \longrightarrow [\mathbf{c}] \rangle]\rangle$

Satisfiability of XPath on data words

XPath($\longrightarrow^+$, $^+\longleftarrow$): undecidable ♣

♠ [Demri, Lazić, 2006]

♣ [F., Segoufin, 2009]

Satisfiability of XPath on data words

XPath($\longrightarrow^+$, $^+\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$,$^*\longleftarrow$): undecidable ♣

XPath($\longrightarrow$, $\longrightarrow^*$,$^*\longleftarrow$): undecidable ♣

♠ [Demri, Lazić, 2006]

♣ [F., Segoufin, 2009]

Satisfiability of XPath on data words

XPath($\longrightarrow^+$, $^+\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$,$^*\longleftarrow$): undecidable ♣

XPath($\longrightarrow$, $\longrightarrow^*$,$^*\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$): decidable, non-PR ♠ ♣

♠ [Demri, Lazić, 2006]

♣ [F., Segoufin, 2009]

Satisfiability of XPath on data words

XPath($\longrightarrow^+$, $^+\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$,$^*\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow$, $\longrightarrow^*$,$^*\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$): decidable, non-PR ♠ ♣

In particular, any fragment with $\longrightarrow^+$ or $^+\!\longleftarrow$ is **undecidable** or has a **non-PR complexity**

♠ [Demri, Lazić, 2006]

♣ [F., Segoufin, 2009]

Satisfiability of XPath on data words

XPath($\longrightarrow^+$, $^+\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$,$^*\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow$, $\longrightarrow^*$,$^*\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$): decidable, non-PR ♠ ♣

In particular, any fragment with $\longrightarrow^+$ or $^+\!\longleftarrow$ is **undecidable** or has a **non-PR complexity**

why?

♠ [Demri, Lazić, 2006]

♣ [F., Segoufin, 2009]

Satisfiability of XPath on data words

XPath($\longrightarrow^+$, $^+\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$,$^*\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow$, $\longrightarrow^*$,$^*\!\longleftarrow$): undecidable ♣

XPath($\longrightarrow^+$): decidable, non-PR ♠ ♣

In particular, any fragment with $\longrightarrow^+$ or $^+\!\longleftarrow$ is **undecidable** or has a **non-PR complexity**

why?

SAT-XPath($\longrightarrow$,$\longrightarrow^+$)

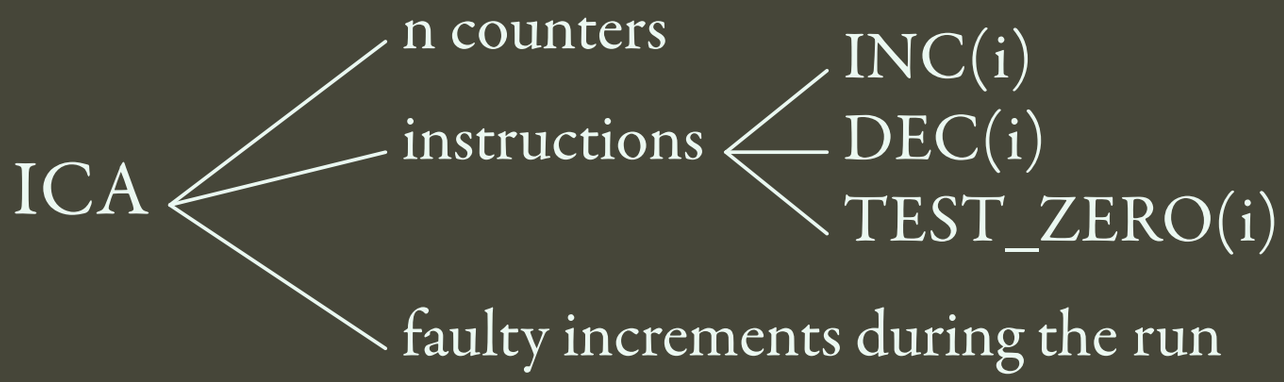
$\downarrow$

SAT-XPath($\longrightarrow^+$)

♠ [Demri, Lazić, 2006]

♣ [F., Segoufin, 2009]

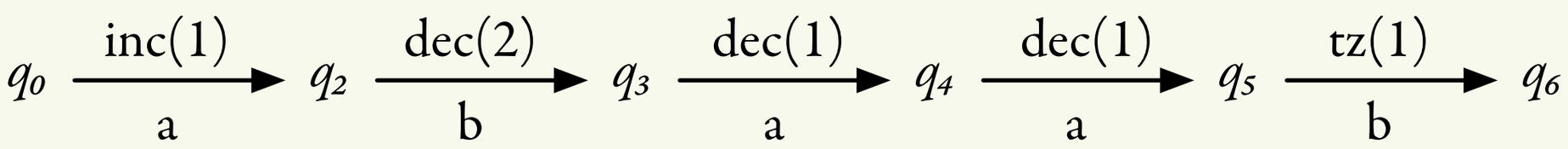
Incrementing faulty Counter Automata



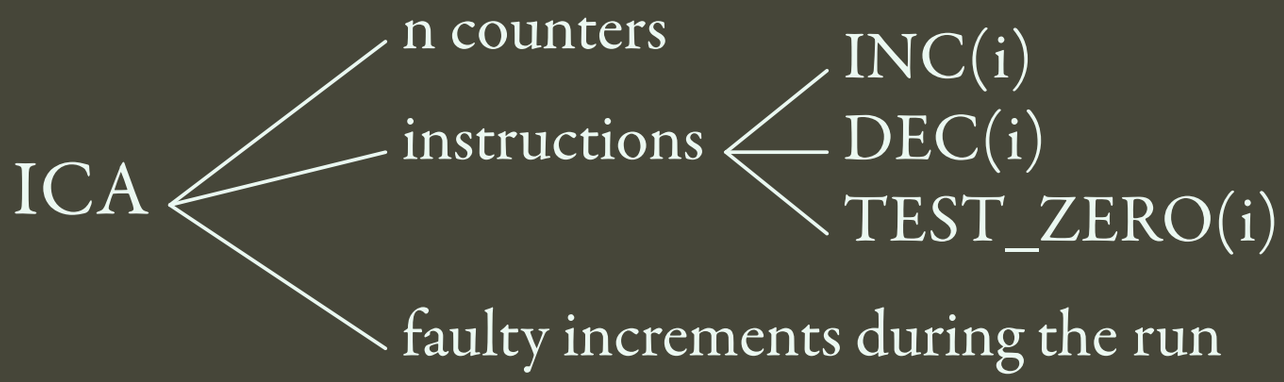
Non-emptiness of ICA is decidable, **non-primitive recursive** [Demri / Lazić / Schnoebelen]

e.g.

a run with 2 counters...



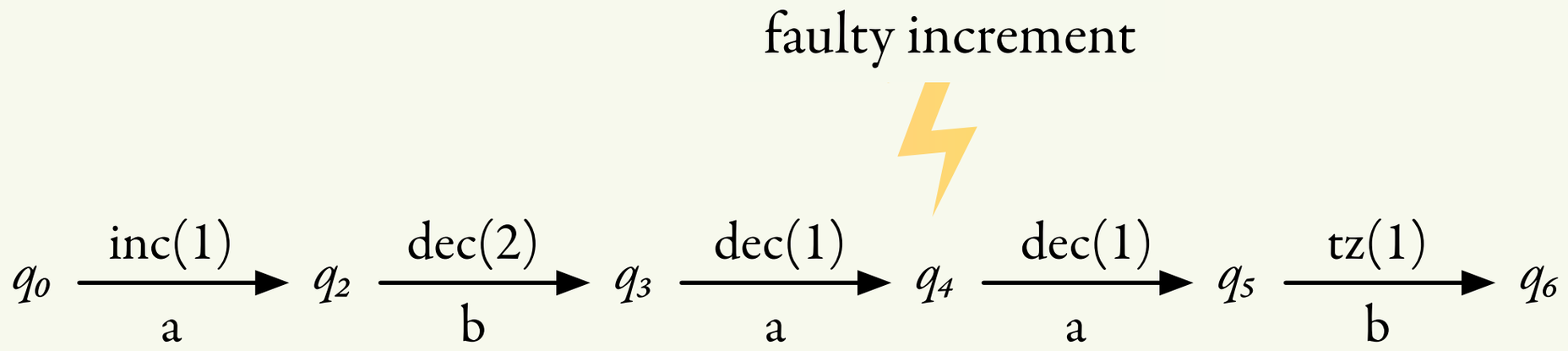
Incrementing faulty Counter Automata



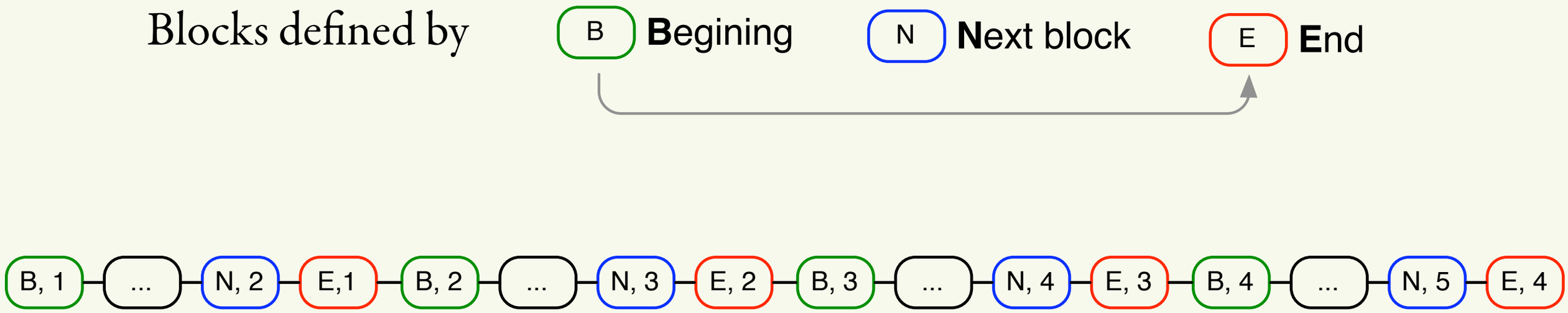
Non-emptiness of ICA is decidable, **non-primitive recursive** [Demri / Lazić / Schnoebelen]

e.g.

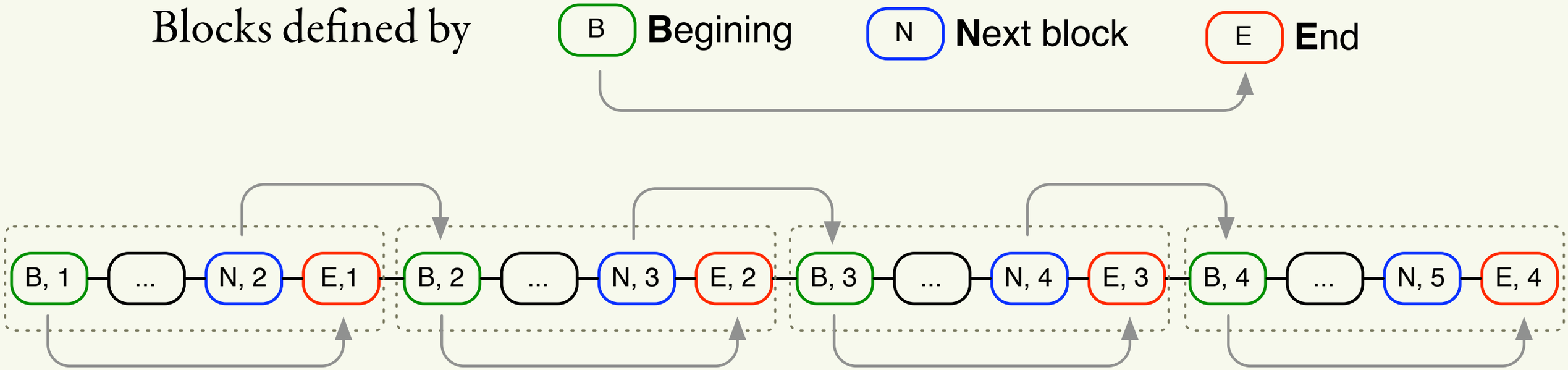
a run with 2 counters...



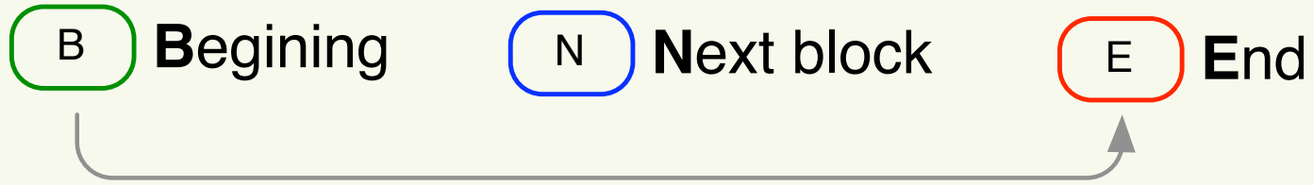
Technical idea of the coding

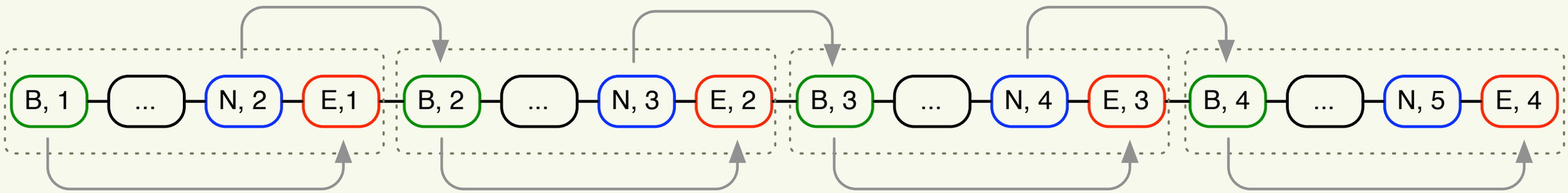


Technical idea of the coding

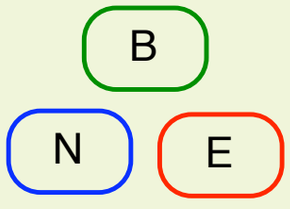


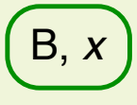
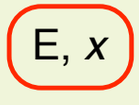
Technical idea of the coding

Blocks defined by 



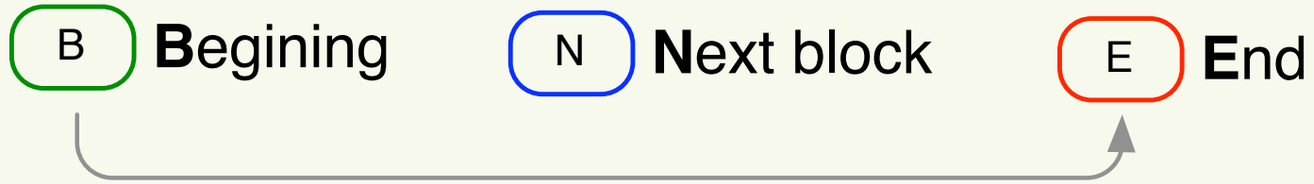
Using only F and G...

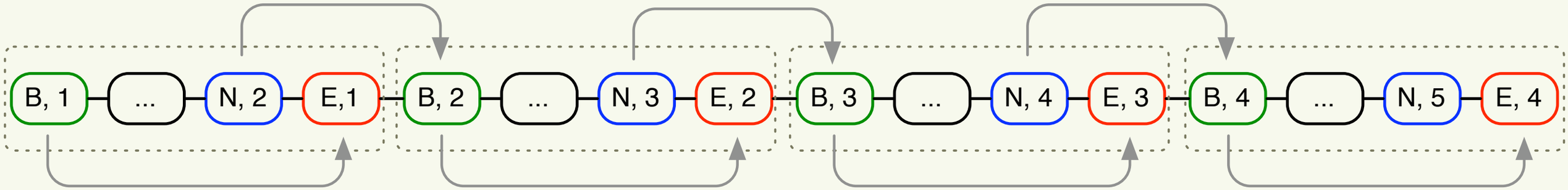
All  have different data

For every  there is a 
with the same data value

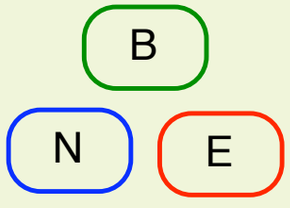
Technical idea of the coding

Blocks defined by





Using only F and G...

All  have different data

For every B, *x* there is a E, *x* with the same data value

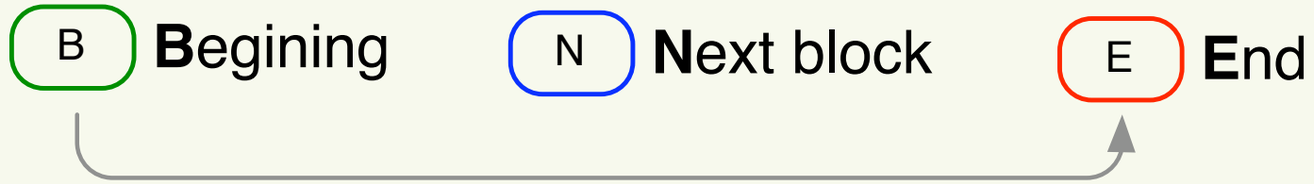
N points to next block

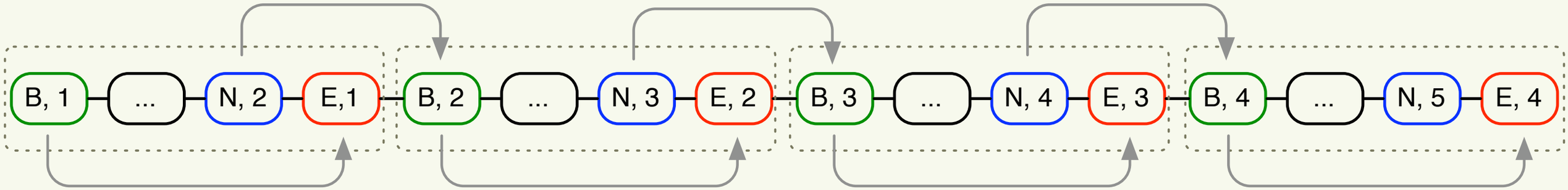
No more than one B N E per block

Order B, *x* ... N E, *x*

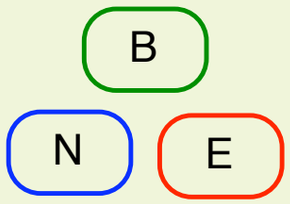
Technical idea of the coding

Blocks defined by





Using only F and G...

All  have different data

For every B, *x* there is a E, *x* with the same data value

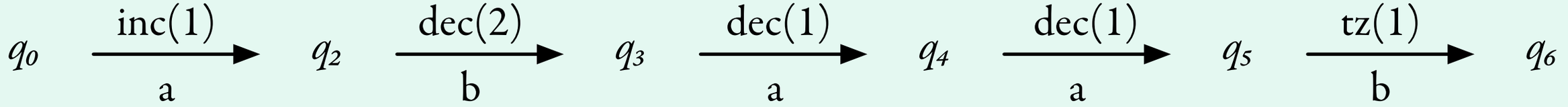
N points to next block

No more than one B N E per block

Order B, *x* ... N E, *x*

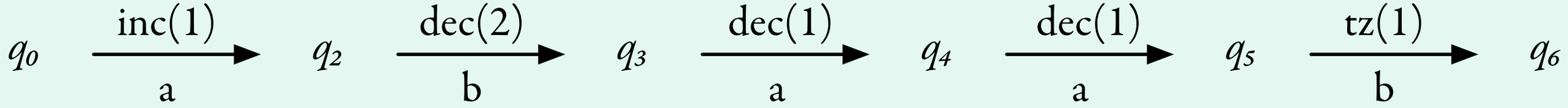
$$\text{next-block}(\phi) := \langle \varepsilon = \longrightarrow^*[\text{N} \wedge \langle \varepsilon = \longrightarrow^*[\text{B} \wedge \phi] \rangle] \longrightarrow^*[\text{E}] \rangle$$

we code:



as: $(q_0, a, inc(1), q_2)$ $(q_2, b, dec(2), q_3)$ $(q_3, a, dec(1), q_4)$ $(q_4, a, dec(1), q_5)$ $(q_5, b, tz(1), q_6)$

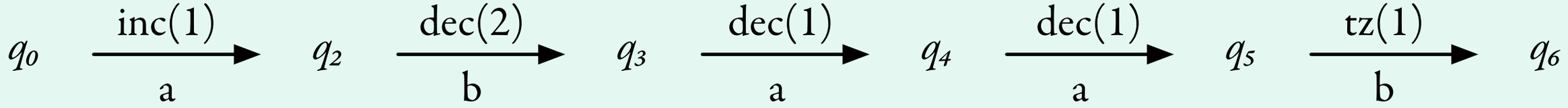
we code:



as: $(q_0, a, inc(1), q_2)$ $(q_2, b, dec(2), q_3)$ $(q_3, a, dec(1), q_4)$ $(q_4, a, dec(1), q_5)$ $(q_5, b, tz(1), q_6)$

the blocks are:

we code:



as: $(q_0, \text{a}, \text{inc}(1), q_2)$ $(q_2, \text{b}, \text{dec}(2), q_3)$ $(q_3, \text{a}, \text{dec}(1), q_4)$ $(q_4, \text{a}, \text{dec}(1), q_5)$ $(q_5, \text{b}, \text{tz}(1), q_6)$

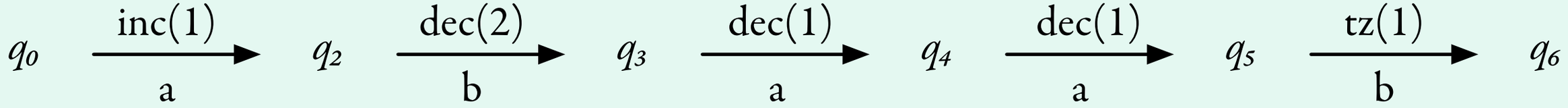
the blocks are:

check:

(ends) start with initial state, end with final state

(tran) every $(q_0, \text{a}, \text{inc}(1), q_2)$ is in δ.

we code:



as: $(q_0, \text{a}, \text{inc}(1), q_2)$ $(q_2, \text{b}, \text{dec}(2), q_3)$ $(q_3, \text{a}, \text{dec}(1), q_4)$ $(q_4, \text{a}, \text{dec}(1), q_5)$ $(q_5, \text{b}, \text{tz}(1), q_6)$

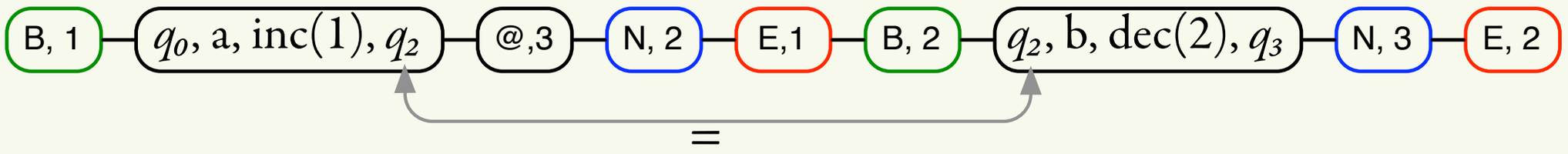
the blocks are:

check:

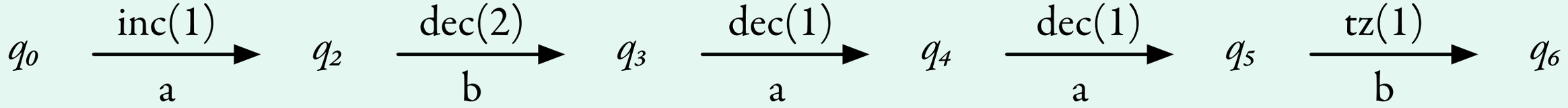
(ends) start with initial state, end with final state

(tran) every $(q_0, \text{a}, \text{inc}(1), q_2)$ is in δ. next-block(ϕ)

(chain)



we code:



as: $(q_0, \text{a}, \text{inc}(1), q_2)$ $(q_2, \text{b}, \text{dec}(2), q_3)$ $(q_3, \text{a}, \text{dec}(1), q_4)$ $(q_4, \text{a}, \text{dec}(1), q_5)$ $(q_5, \text{b}, \text{tz}(1), q_6)$

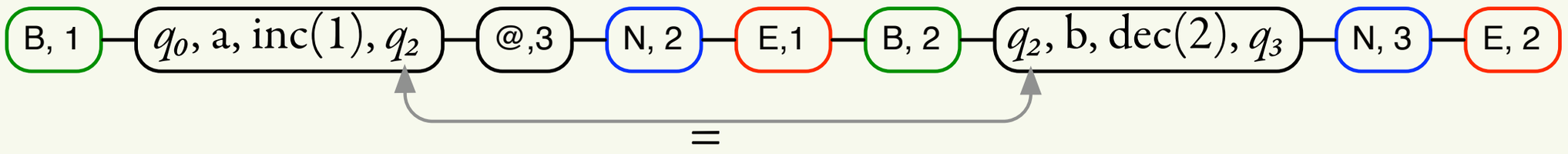
the blocks are:

check:

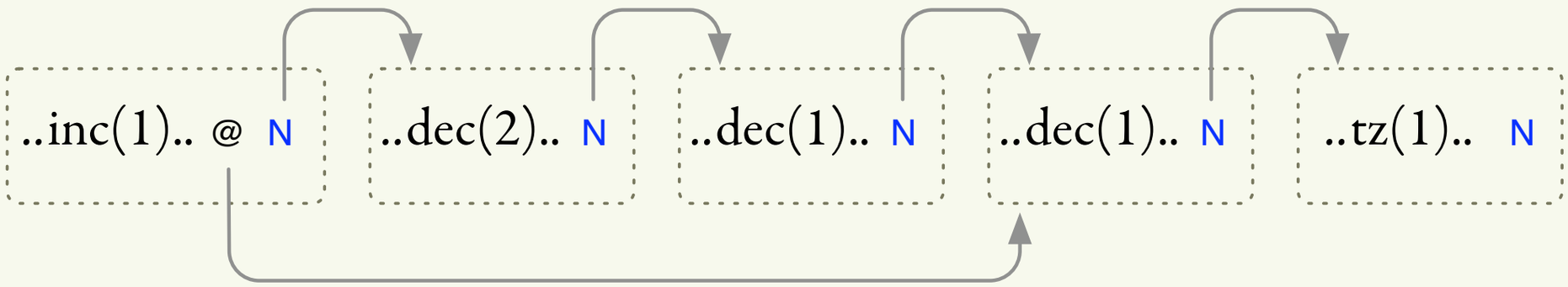
(ends) start with initial state, end with final state

(tran) every $(q_0, \text{a}, \text{inc}(1), q_2)$ is in δ. next-block(ϕ)

(chain)



(pair) every increment is decremented before the first test_zero to its right



Complexity

Therefore, we have:

- XPath($\longrightarrow^+$) is (decidable and) non-primitive recursive
- XPath($^+\longleftarrow$, $\longrightarrow^+$) and XPath($^*\longleftarrow$, $\longrightarrow^+$) are undecidable
- LTL_1(F) is (decidable and) non-primitive recursive
- LTL_1(F,F^{-1}) is undecidable
- LTL_1(F,F^{-1})

Horizontal

XPath($\rightarrow, \rightarrow^*, \leftarrow, {}^*\leftarrow$)

XPath($\rightarrow^+, {}^+\leftarrow$) : undecidable ♣

XPath($\rightarrow^+, {}^*\leftarrow$) : undecidable ♣

XPath($\rightarrow, \rightarrow^*, {}^*\leftarrow$) : undecidable ♣

XPath($\rightarrow^+$) : decidable, non-PR ♠♣

In particular, any fragment with $\rightarrow^+$ or ${}^+\leftarrow$ is **undecidable** or has a **non-PR complexity**

♠ [Demri, Lazić, LICS'06]

♣ [F., Segoufin, MFCS'09]

Horizontal

XPath($\rightarrow, \rightarrow^*, \leftarrow, {}^*\leftarrow$)

XPath($\rightarrow^+, {}^+\leftarrow$) : undecidable ♣

XPath($\rightarrow^+, {}^*\leftarrow$) : undecidable ♣

XPath($\rightarrow, \rightarrow^*, {}^*\leftarrow$) : undecidable ♣

XPath($\rightarrow^+$) : decidable, non-PR ♠♣

In particular, any fragment with $\rightarrow^+$ or ${}^+\leftarrow$ is **undecidable** or has a **non-PR complexity**

What about XPath($\rightarrow^*$)?

♠ [Demri, Lazić, LICS'06]

♣ [F., Segoufin, MFCS'09]

Horizontal

$\mathsf{XPath}(\rightarrow, \rightarrow^*, \leftarrow, {}^*\leftarrow)$

$\mathsf{XPath}(\rightarrow^+, {}^+\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow^+, {}^*\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow, \rightarrow^*, {}^*\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow^+)$: decidable, non-PR ♠♣

In particular, any fragment with $\rightarrow^+$ or ${}^+\leftarrow$ is **undecidable** or has a **non-PR complexity**

What about $\mathsf{XPath}(\rightarrow^*)$?

$\mathsf{XPath}(\rightarrow^*)$ is decidable in 2ExpSpace ♦

$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace ♦

♦ [F., LICS'11]

♠ [Demri, Lazić, LICS'06]

♣ [F., Segoufin, MFCS'09]

Horizontal

$\mathsf{XPath}(\rightarrow, \rightarrow^*, \leftarrow, {}^*\leftarrow)$

$\mathsf{XPath}(\rightarrow^+, {}^+\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow^+, {}^*\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow, \rightarrow^*, {}^*\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow^+)$: decidable, non-PR ♠♣

In particular, any fragment with $\rightarrow^+$ or ${}^+\leftarrow$ is **undecidable** or has a **non-PR complexity**

What about $\mathsf{XPath}(\rightarrow^*)$?

$\mathsf{XPath}(\rightarrow^*)$ is decidable in 2ExpSpace ♦

$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace ♦

$\mathsf{XPath}(\rightarrow^*, \downarrow_*, {}^*\leftarrow)$ is decidable ♥

♥ [F., PODS'13]

♦ [F., LICS'11]

♠ [Demri, Lazić, LICS'06]

♣ [F., Segoufin, MFCS'09]

Horizontal

$\mathsf{XPath}(\rightarrow, \rightarrow^*, \leftarrow, {}^*\leftarrow)$

$\mathsf{XPath}(\rightarrow^+, {}^+\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow^+, {}^*\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow, \rightarrow^*, {}^*\leftarrow)$: undecidable ♣

$\mathsf{XPath}(\rightarrow^+)$: decidable, non-PR ♠♣

In particular, any fragment with $\rightarrow^+$ or ${}^+\leftarrow$ is **undecidable** or has a **non-PR complexity**

What about $\mathsf{XPath}(\rightarrow^*)$?

$\mathsf{XPath}(\rightarrow^*)$ is decidable in 2ExpSpace ♦

$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace ♦

$\mathsf{XPath}(\rightarrow^*, \downarrow_*, {}^*\leftarrow)$ is decidable ♥

♥ [F., PODS'13]

♦ [F., LICS'11]

♠ [Demri, Lazić, LICS'06]

♣ [F., Segoufin, MFCS'09]

$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace

Proof idea

ϕ :
- there is only one dv under a **c**
- for every **a**, there is a **b** accessible via a **c** with the same dv
- there is a **c** with the same dv as the current position

e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace

Proof idea

ϕ : { there is only one dv under a **c**
{ for every **a**, there is a **b** accessible via a **c** with the same dv
{ there is a **c** with the same dv as the current position

e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

a 4	1 : { $\rightarrow^*$[c]$\rightarrow^*$[b], $\rightarrow^*$[b], $\rightarrow^*$[c] } 2 : { } 3 : { $\rightarrow^*$[b] } ⋮

mosaic
- a label
- a data value
- for every dv *d*, all the paths that can access *d*

restricted to subpaths of ϕ = { $\rightarrow^*$[c]$\rightarrow^*$[b], $\rightarrow^*$[c], $\rightarrow^*$[b] }

$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace

Proof idea

ϕ : { there is only one dv under a **c**
for every **a**, there is a **b** accessible via a **c** with the same dv
there is a **c** with the same dv as the current position

e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

a relation  if they can abstract consecutive positions in a word

$\rightsquigarrow$ An infinite transition system TS(ϕ) over mosaics.

ϕ is satisfiable $\Leftrightarrow$ there is 

in TS(ϕ) s.t. the first and last elements have certain conditions

XPath($\rightarrow^*, {}^*\leftarrow$) is decidable in 2ExpSpace

Proof idea

ϕ : { there is only one dv under a **c**
for every **a**, there is a **b** accessible via a **c** with the same dv
there is a **c** with the same dv as the current position

e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

We can add a dv that simulates 4

XPath($\rightarrow^*, {}^*\leftarrow$) is decidable in 2ExpSpace

Proof idea

ϕ :
- there is only one dv under a **c**
- for every **a**, there is a **b** accessible via a **c** with the same dv
- there is a **c** with the same dv as the current position

e.g.

a	b	b	b	a	a	b	c	a	b	c	c	a	a	a	a	b	c	b	b
1	2	4	9	4	9	3	1	5	1	1	1	4	9	4	9	5	1	4	9

$\models \phi$

We can add a dv that simulates 4

XPath($\rightarrow^*, {}^*\leftarrow$) is decidable in 2ExpSpace

Proof idea

ϕ :
- there is only one dv under a **c**
- for every **a**, there is a **b** accessible via a **c** with the same dv
- there is a **c** with the same dv as the current position

e.g.

a	b	b	b	a	a	b	c	a	b	c	c	a	a	a	a	b	c	b	b
1	2	4	9	4	9	3	1	5	1	1	1	4	9	4	9	5	1	4	9

$\models \phi$

We can add a dv that simulates 4

...but we cannot simulate 1.

$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace

Proof idea

ϕ :
- there is only one dv under a **c**
- for every **a**, there is a **b** accessible via a **c** with the same dv
- there is a **c** with the same dv as the current position

e.g.

a	b	b	b	a	a	b	c	a	b	c	c	a	a	a	a	b	c	b	b
1	2	4	9	4	9	3	1	5	1	1	1	4	9	4	9	5	1	4	9

$\models \phi$

We can add a dv that simulates 4 → flexible value

...but we cannot simulate 1 → rigid value

satisfaction of ϕ is closed under simulation of flexible values

rigid value : from some position it is the only dv accessed with a path α

XPath($\rightarrow^*, {}^*\leftarrow$) is decidable in 2ExpSpace

Proof idea

ϕ : { there is only one dv under a **c**
for every **a**, there is a **b** accessible via a **c** with the same dv
there is a **c** with the same dv as the current position

e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

adding simulated values of flexible values: ≤ ☐ ≤ ☐

XPath($\rightarrow^*, {}^*\leftarrow$) is decidable in 2ExpSpace

Proof idea

ϕ : { there is only one dv under a **c**
for every **a**, there is a **b** accessible via a **c** with the same dv
there is a **c** with the same dv as the current position

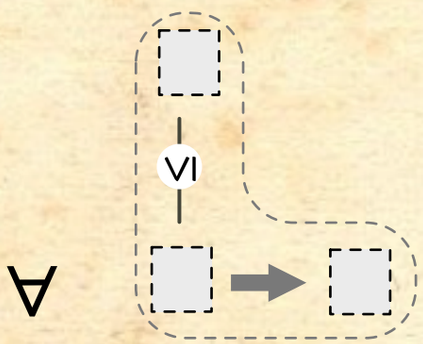
e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

adding simulated values of flexible values: $\leq$ $\square \leq \square$

A monotonicity property:



XPath($\rightarrow^*, {}^*\leftarrow$) is decidable in 2ExpSpace

Proof idea

ϕ :
- there is only one dv under a **c**
- for every **a**, there is a **b** accessible via a **c** with the same dv
- there is a **c** with the same dv as the current position

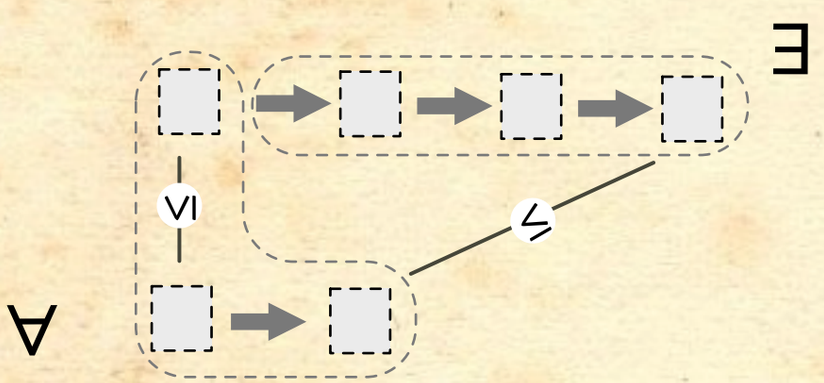
e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

adding simulated values of flexible values: ≤ ▢ ≤ ▢

A monotonicity property:



$\mathsf{XPath}(\rightarrow^*, {}^*\leftarrow)$ is decidable in 2ExpSpace

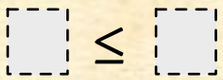
Proof idea

ϕ : { there is only one dv under a **c**
for every **a**, there is a **b** accessible via a **c** with the same dv
there is a **c** with the same dv as the current position

e.g.

a	b	b	a	b	c	a	b	c	c	a	a	b	c	b
1	2	4	4	3	1	5	1	1	1	4	4	5	1	4

$\models \phi$

adding simulated values of flexible values: $\leq$ 

we only need to consider $\leq$-minimal mosaics

there are boundedly many (since there are boundedly many rigid values)

we reduce to a derivation problem for a *finite* transition system