On the tree-transformation power of XSLT

Wim Janssen Jan Van den Bussche Alexandr Korlyukov
{wim.janssen, jan.vandenbussche}@uhasselt.be

Hasselt University - Grodno State University

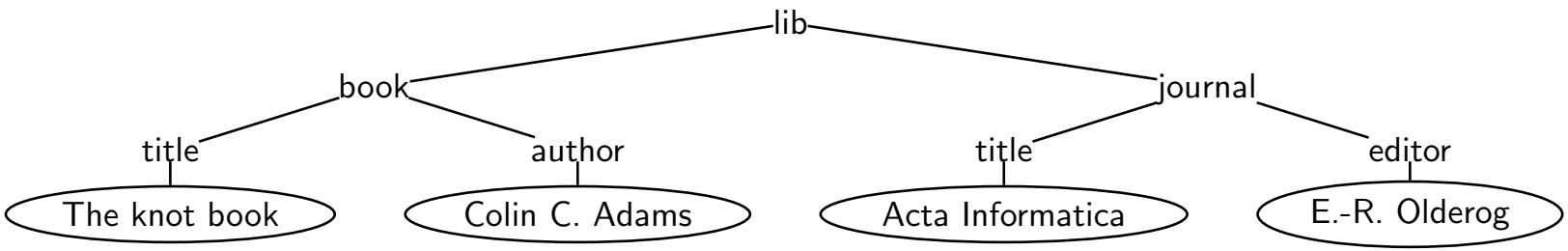
Dutch Belgian Database Day - 15 November 2006

arXiv:cs.PL/0603028

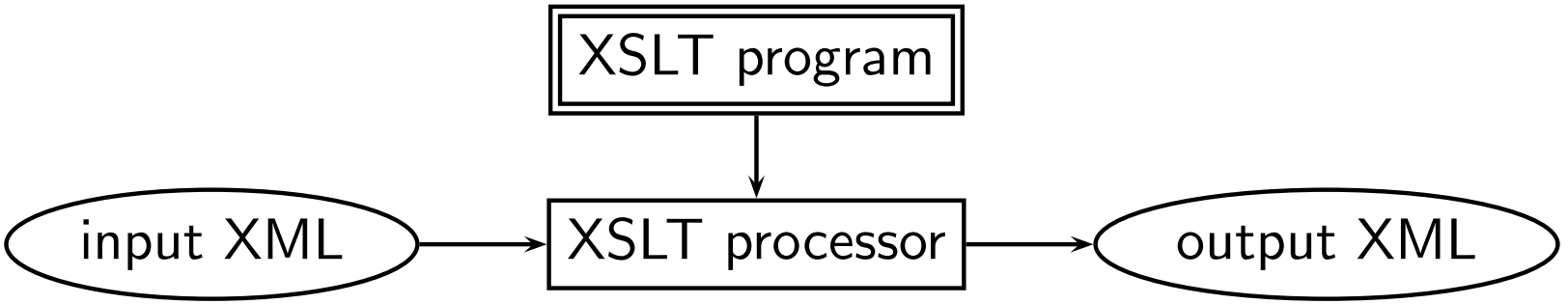
Acta Informatica, online, 26 October 2006

- XML: eXtensible Markup Language (W3C Recommendation)

```
<lib>
   <book>
      <title>The knot book</title>
      <author>Colin C. Adams</author>
   </book>
   <journal>
      <title>Acta Informatica</title>
      <editor>E.-R. Olderog</editor>
   </journal>
</lib>
```



XSLT



- XSLT: eXtensible Stylesheet Language - Transformations
- rule-based programming language for expressing transformations of XML data.
- version 1.0: W3C Recommendation (1999)
- version 2.0: W3C Candidate Recommendation (2005)

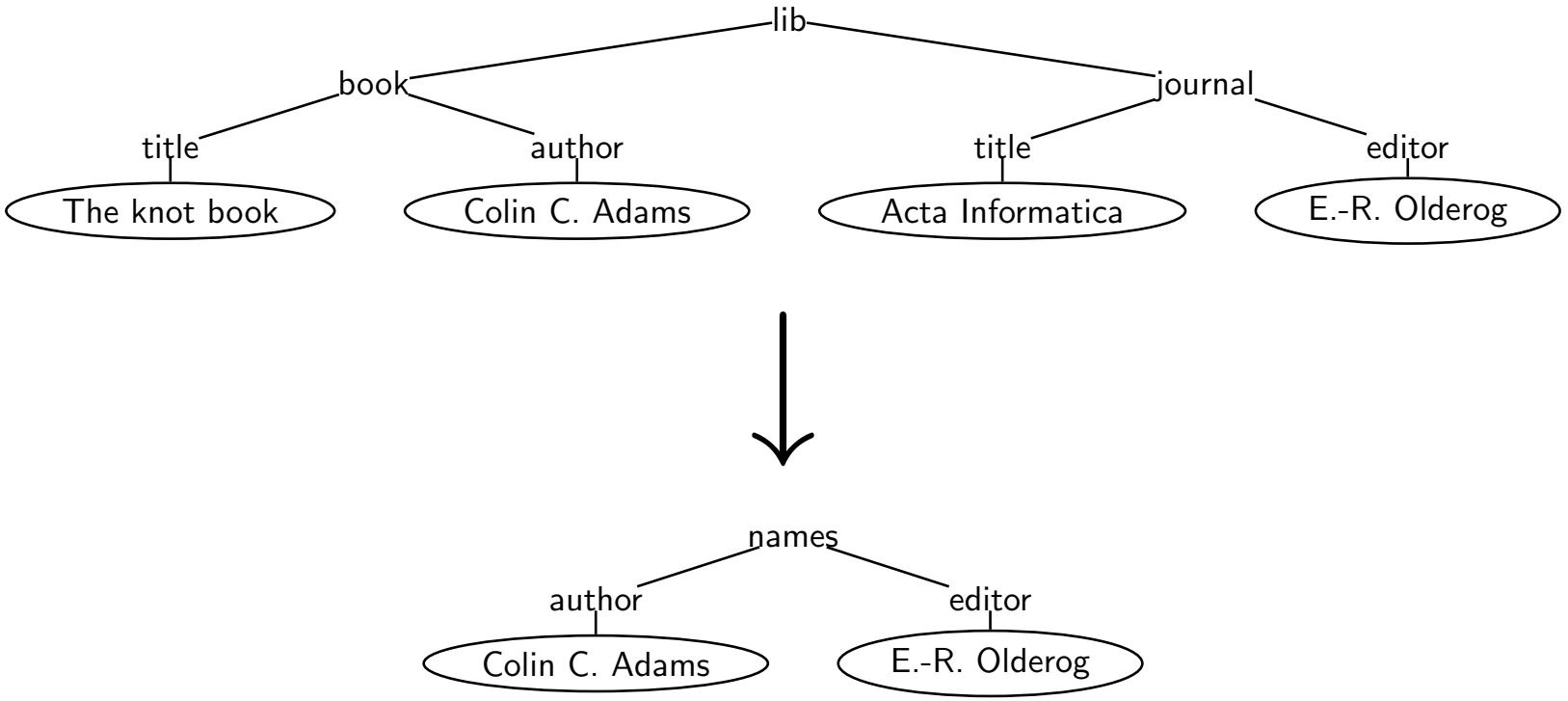
XSLT example

```
<xsl:stylesheet>
   <xsl:template match="/lib">
      <names>
         <xsl:apply-templates select="book"/>
         <xsl:apply-templates select="journal"/>
      </names>
   </xsl:template>

   <xsl:template match="book">
      <xsl:copy-of select="./child::author"/>
   </xsl:template>

   <xsl:template match="journal">
      <xsl:copy-of select="./child::editor"/>
   </xsl:template>
</xsl:stylesheet>
```

XSLT example



goal

What is the computational power of XSLT?

- String-manipulation functions: obviously computational complete!
- Structural tree transformations
  - Fixed tag-alphabet (node labels)
  - Ignore text nodes (data)
  - No use of string-manipulation functions and arithmetics

Outline

1. Formal semantics
2. Computational completeness of XSLT 2.0
   - Temporary Trees
3. Exponential time upper bound for XSLT 1.0

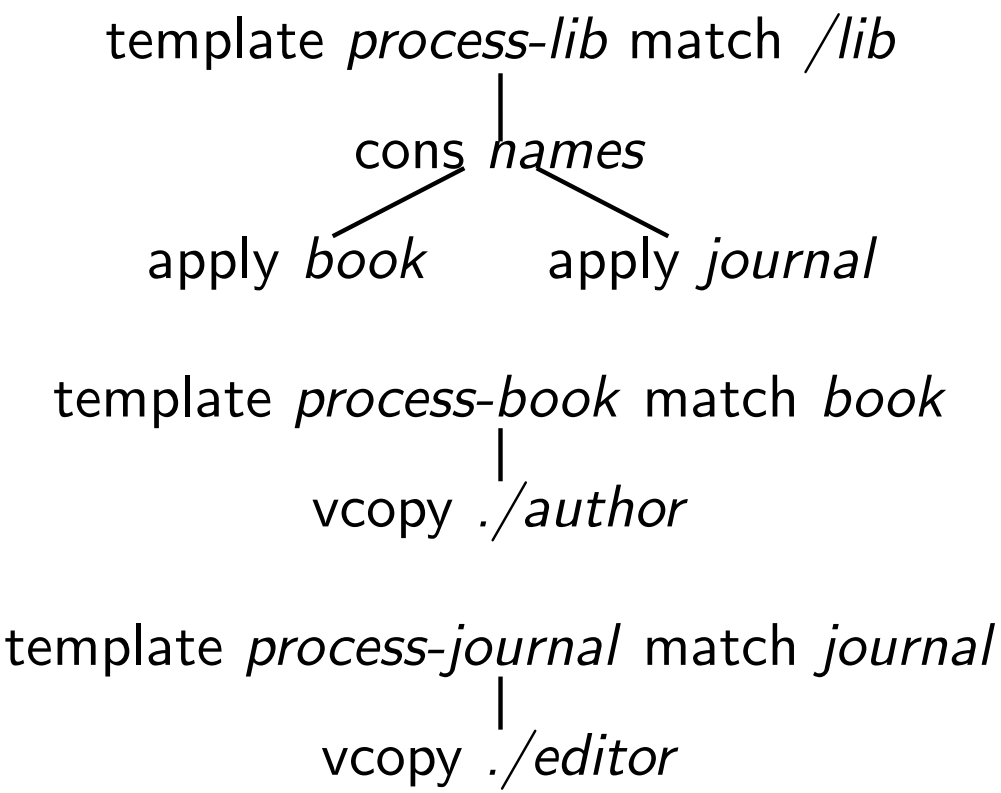
Formal syntax

Program → Rule*
Rule → template *name* match *expr* (mode *name*)? { Template }
Template → Statement*
Statement → cons *label* { Template }
| apply *expr* (mode *name*)?
| call *name*
| foreach *expr* { Template }
| val *value_variable expr*
| tree *tree_variable* { Template }
| vcopy *expr*
| tcopy *tree_variable*
| if *expr* { Template } else { Template }

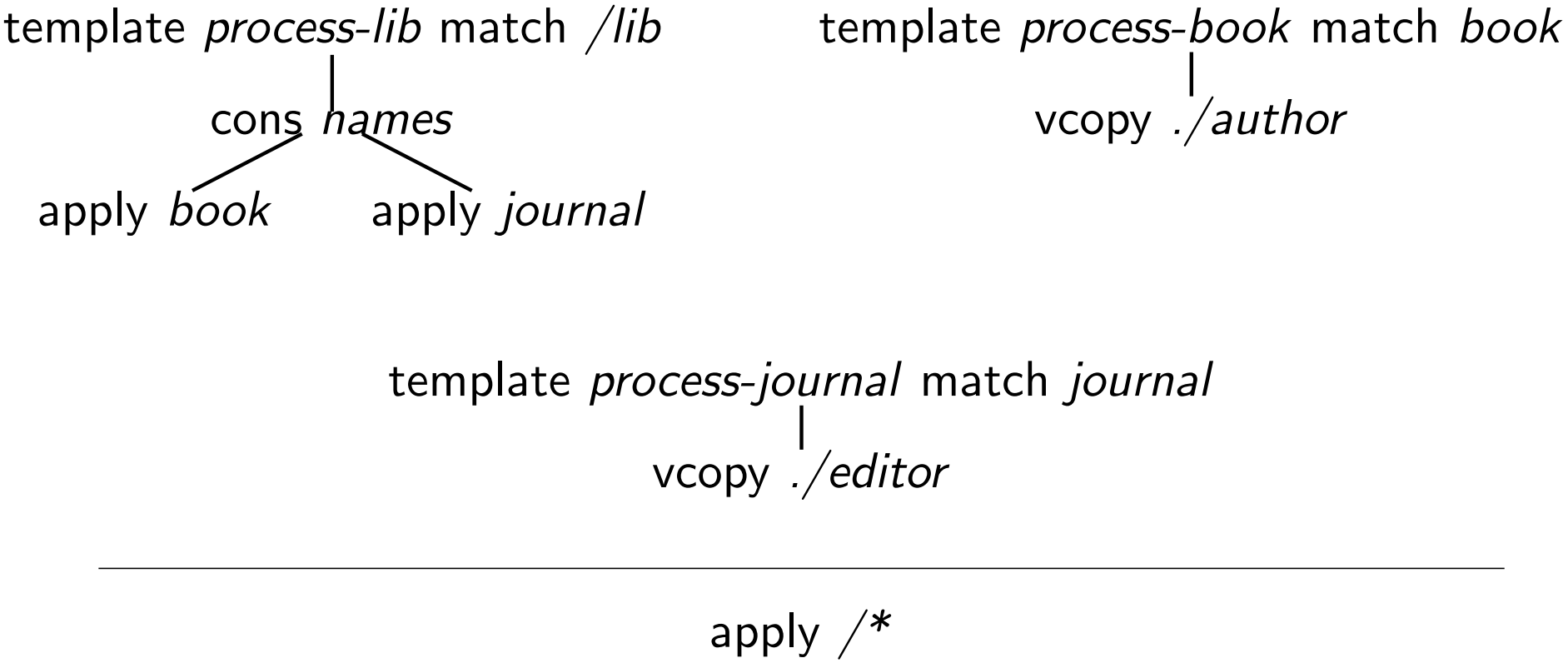
Formal semantics: rewrite relation $\Rightarrow$

- $$\frac{S = \text{if } e \text{ \{ } M_{\text{true}} \text{ \} else \{ } M_{\text{false}} \text{ \} } \lhd \gamma \quad \gamma(S) = C \quad \mathit{eval}(e, C) \neq \emptyset}{\gamma \overset{\text{if}}{\Rightarrow} \gamma(S \leftarrow M_{\text{true}})}$$
- $$\frac{S = \text{if } e \text{ \{ } M_{\text{true}} \text{ \} else \{ } M_{\text{false}} \text{ \} } \lhd \gamma \quad \gamma(S) = C \quad \mathit{eval}(e, C) = \emptyset}{\gamma \overset{\text{if}}{\Rightarrow} \gamma(S \leftarrow M_{\text{false}})}$$
- $$\frac{S = \text{vcopy } e \lhd \gamma \quad \gamma(S) = C = (\mathbf{S}, \mathbf{E}, c) \quad \mathit{eval}(e, C) = (\mathbf{n}_1, \ldots, \mathbf{n}_k) \quad \mathit{ttemp}\ \ \mathit{forest}((\mathbf{n}_1, \ldots, \mathbf{n}_k), \mathbf{S}) \ \ = M}{\gamma \Rightarrow \gamma(S \leftarrow M)}$$
- $$\frac{S = \text{val } x\ e \lhd \gamma \quad \gamma(S) = C \quad C(x\colon \mathit{eval}(e, C)) = C' \quad \mathit{updateset}(\gamma, S) = M \quad \mathit{init}(M, C') = \gamma_1 \quad \gamma(SM \leftarrow \gamma_1) \overset{\text{if}}{\Rightarrow} ! \ \gamma'}{\gamma \Rightarrow \gamma'}$$
- $$\frac{S = \text{tcopy } y \lhd \gamma \quad \gamma(S) = (\mathbf{S}, \mathbf{E}, c) \quad (y, \mathbf{t}) \in \mathbf{S} \quad \mathit{ttemp}(\mathit{choproot}(\mathbf{t})) = M}{\gamma \Rightarrow \gamma(S \leftarrow M)}$$
- $$\frac{S = \text{call } \mathit{name} \lhd \gamma \quad \gamma(S) = C \quad \mathit{rulewithname}(\mathit{name}) = M \quad \mathit{init}(M, C) = \gamma_1 \quad \gamma(S \leftarrow \gamma_1) \overset{\text{if}}{\Rightarrow} ! \ \gamma'}{\gamma \Rightarrow \gamma'}$$
- $$\frac{S = \text{tree } y \text{ \{ } M \text{ \} } \lhd \gamma \quad M \text{ is terminal} \quad \gamma(S) = C \quad C(y\colon \mathit{maketree}(M)) = C' \quad \mathit{updateset}(\gamma, S) = M' \quad \mathit{init}(M', C') = \gamma_3 \quad \gamma(SM' \leftarrow \gamma_3) \overset{\text{if}}{\Rightarrow} ! \ \gamma'}{\gamma \Rightarrow \gamma'}$$
- $$\frac{S = \text{apply } e \lhd \gamma \quad \gamma(S) = C = (\mathbf{S}, \mathbf{E}, c) \quad \mathit{eval}(e, C) = (\mathbf{n}_1, \ldots, \mathbf{n}_k) \quad \mathit{ruletoapply}(\mathbf{n}_i, C) = M_i \quad \text{for } i = 1, \ldots, k \quad \mathit{init}(M_i, (\mathbf{S}, \mathbf{E}, (\mathbf{n}_i, i, k))) = \gamma_i \quad \text{for } i = 1, \ldots, k \quad \gamma(S \leftarrow \gamma_1 \ldots \gamma_k) \overset{\text{if}}{\Rightarrow} ! \ \gamma'}{\gamma \Rightarrow \gamma'}$$
- $$\frac{S = \text{foreach } e \text{ \{ } M \text{ \} } \lhd \gamma \quad \gamma(S) = C = (\mathbf{S}, \mathbf{E}, c) \quad \mathit{eval}(e, C) = (z_1, \ldots, z_k) \quad \mathit{init}(M, (\mathbf{S}, \mathbf{E}, (z_i, i, k))) = \gamma_i \quad \text{for } i = 1, \ldots, k \quad \gamma(S \leftarrow \gamma_1 \ldots \gamma_k) \overset{\text{if}}{\Rightarrow} ! \ \gamma'}{\gamma \Rightarrow \gamma'}$$

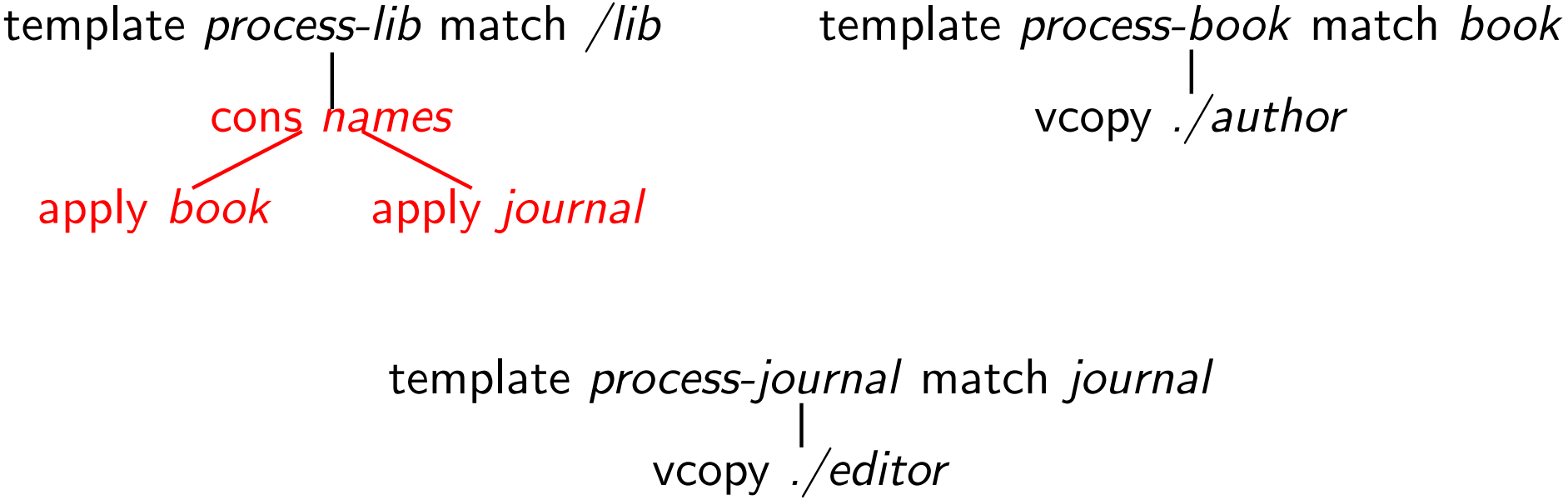
Example


template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor

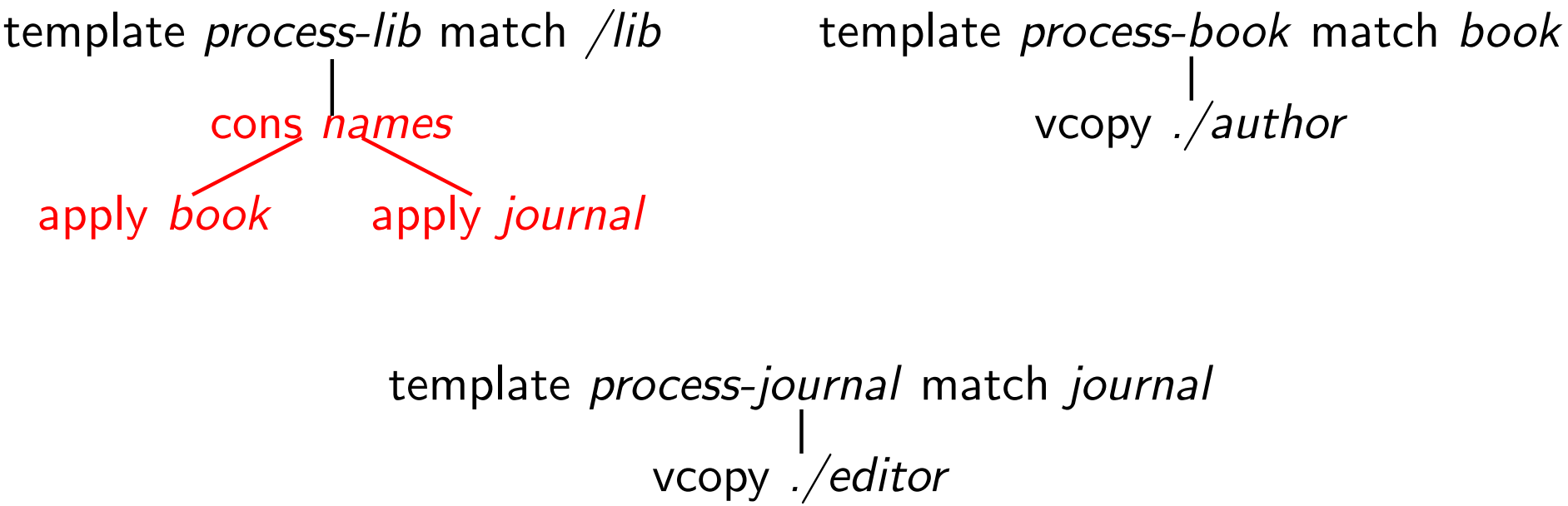
Example 1

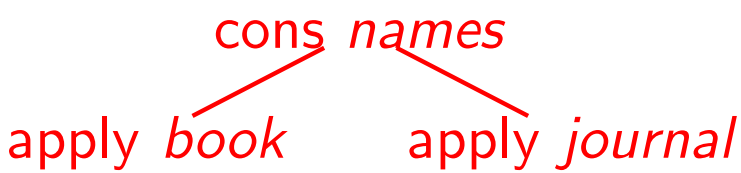

template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor
apply /*

Example 2

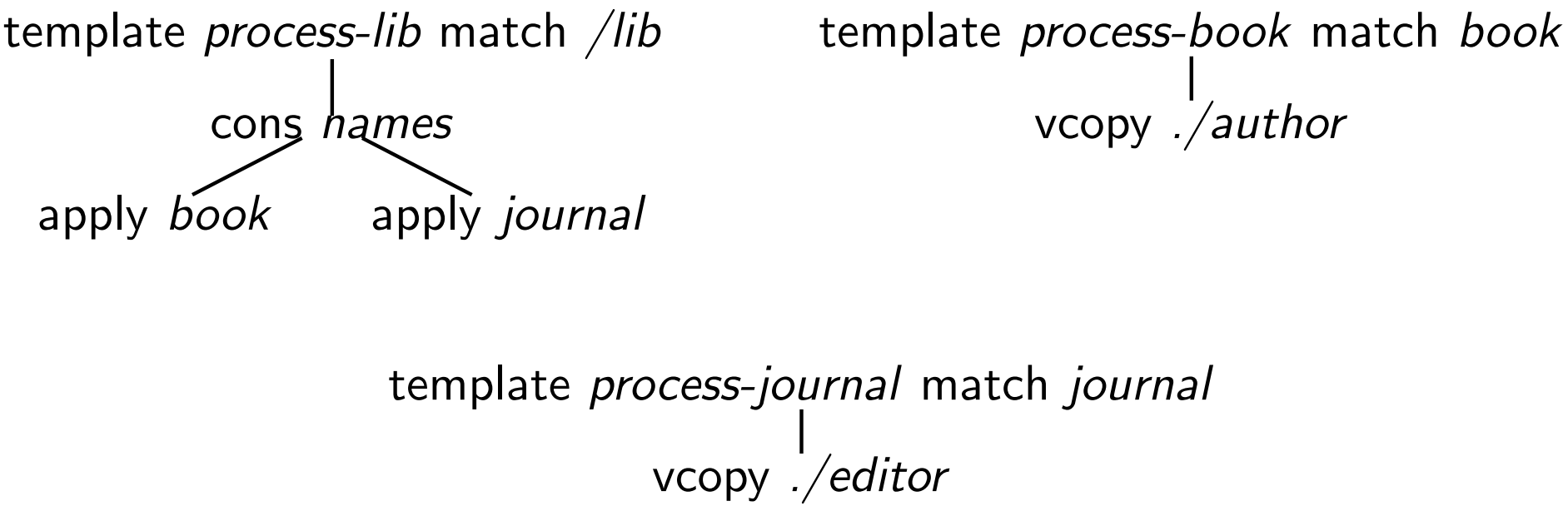

template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor
apply /*

Example 3

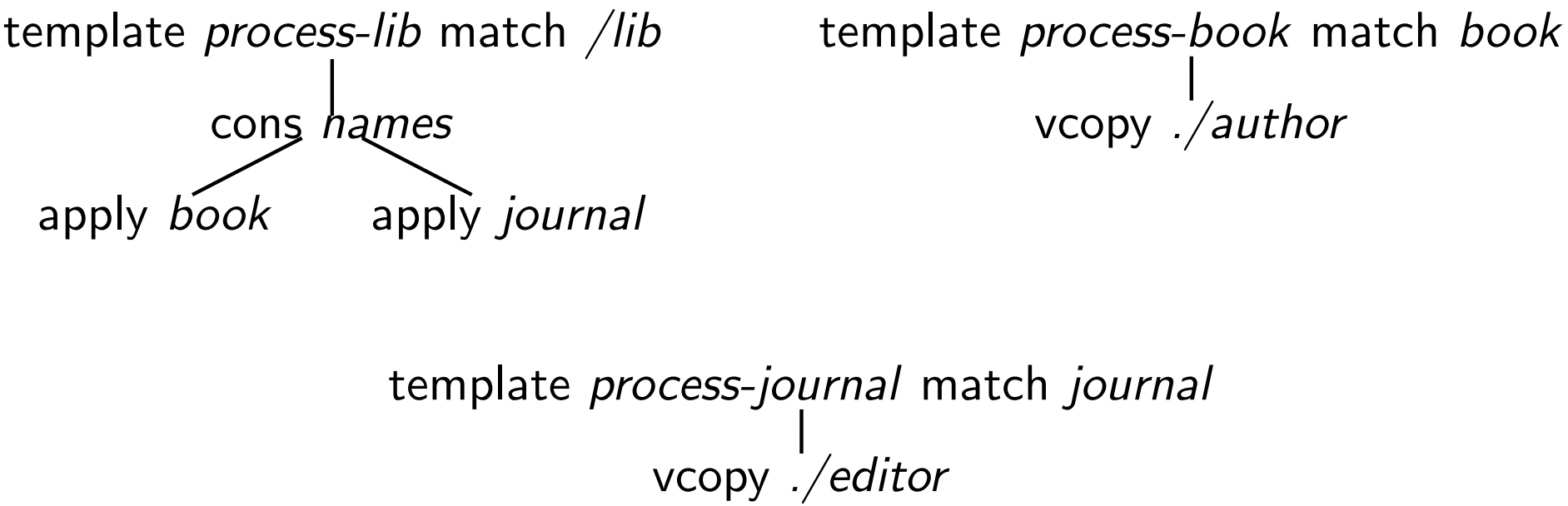

template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor

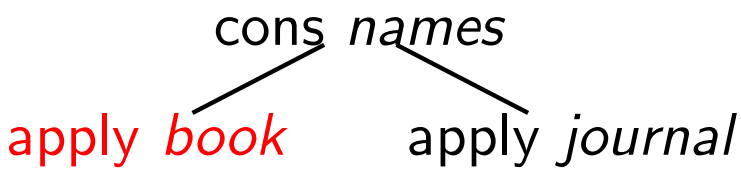

cons names
apply book
apply journal

Example 4

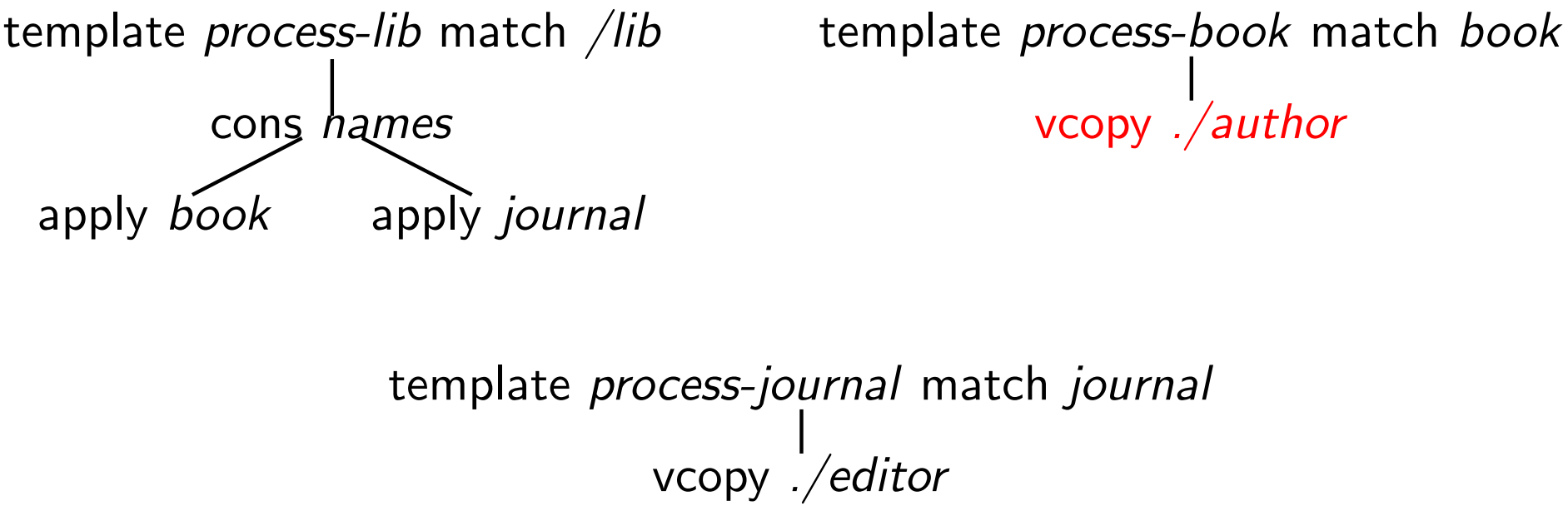

template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor
cons names
apply book
apply journal

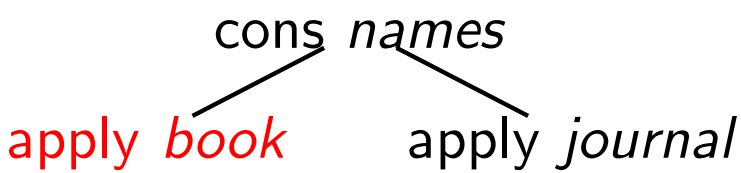
Example 5


template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor

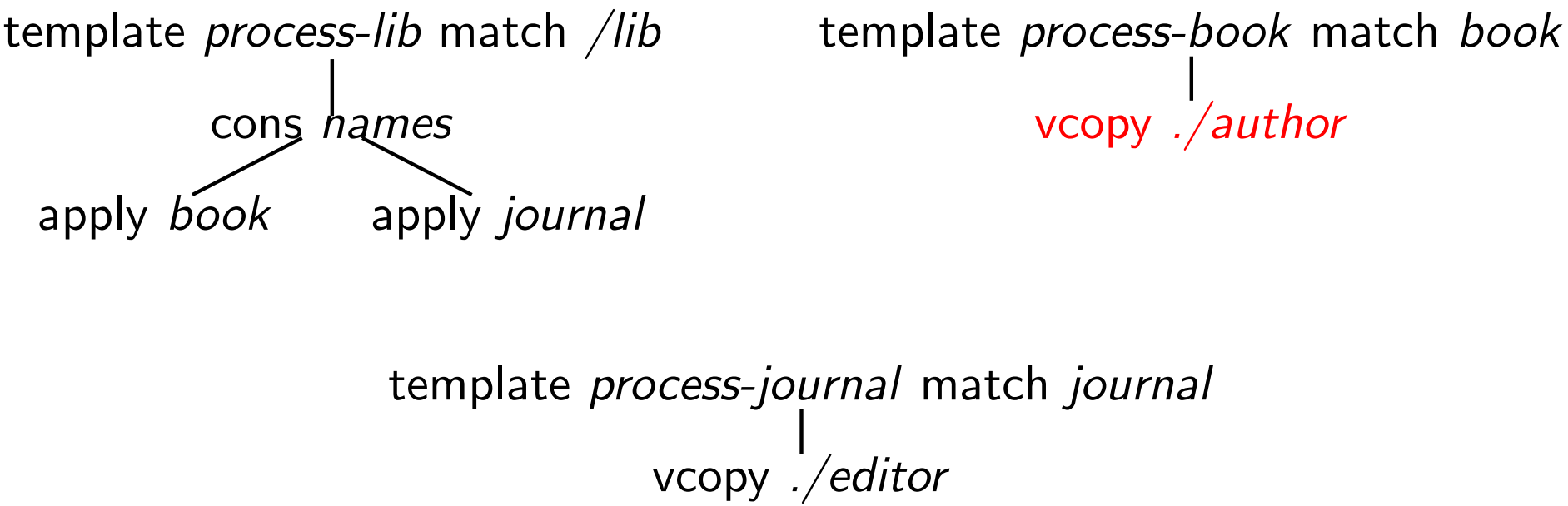

cons names
apply book
apply journal

Example 6

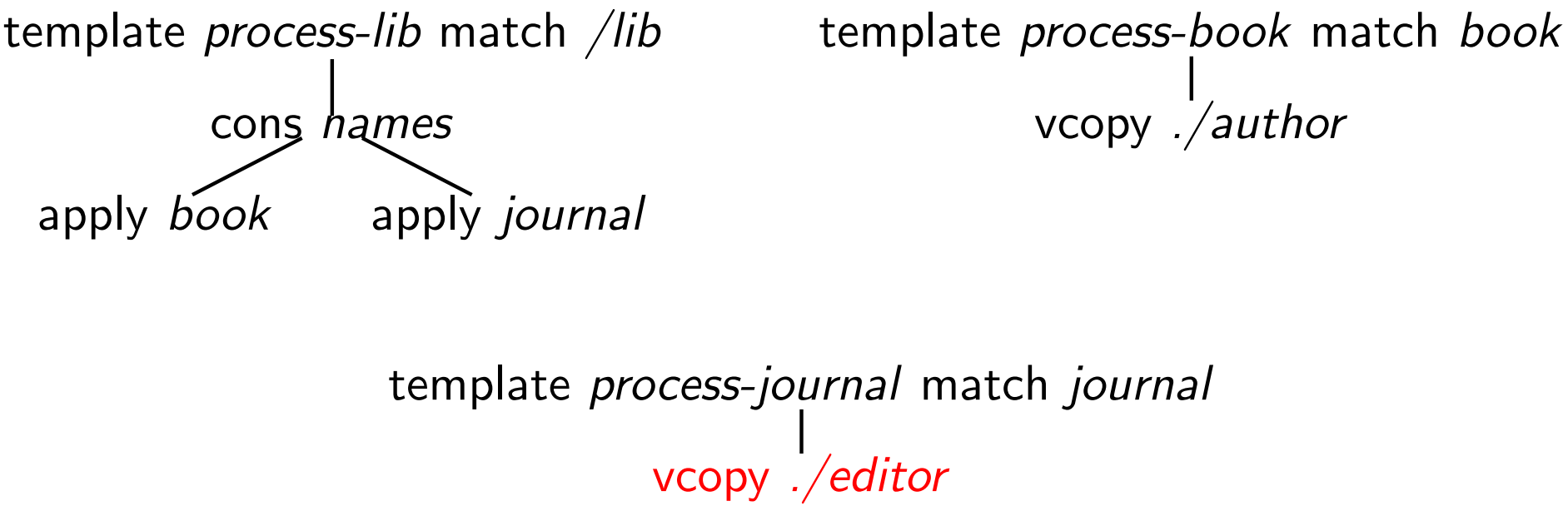

template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor


cons names
apply book
apply journal

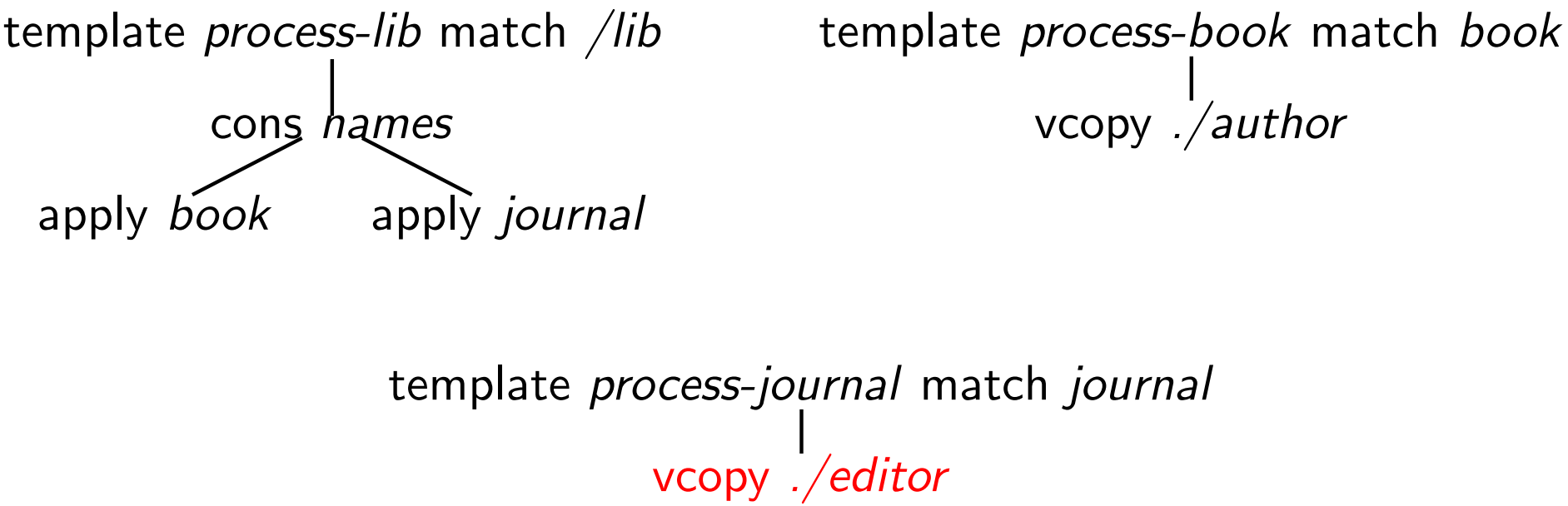
Example 7

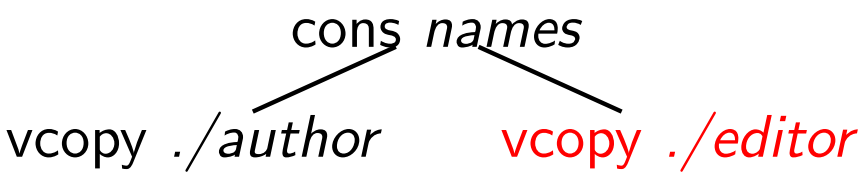

template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor
cons names
vcopy ./author
apply journal

Example 8

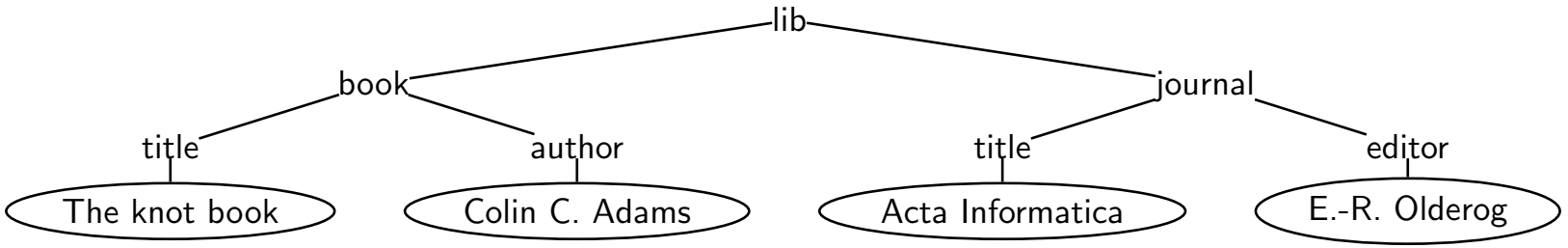

template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor
cons names
vcopy ./author
apply journal

Example 9


template process-lib match /lib
cons names
apply book
apply journal
template process-book match book
vcopy ./author
template process-journal match journal
vcopy ./editor


cons names
vcopy ./author
vcopy ./editor

Example 10

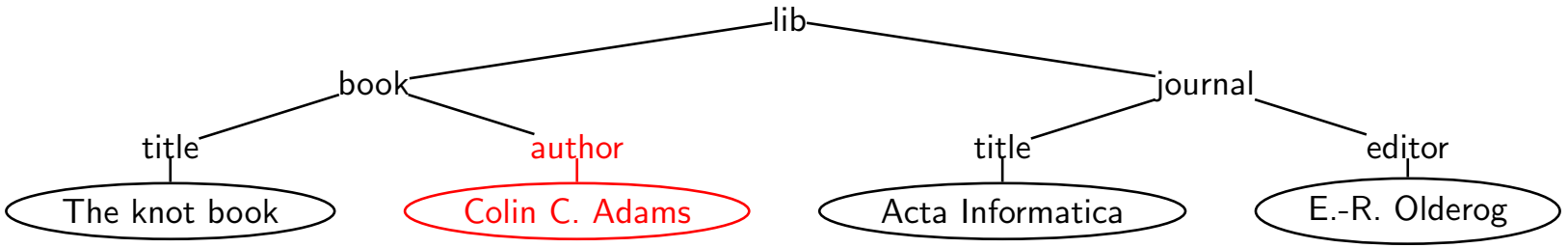


---

cons *names*

vcopy *./author* vcopy *./editor*

Example 11

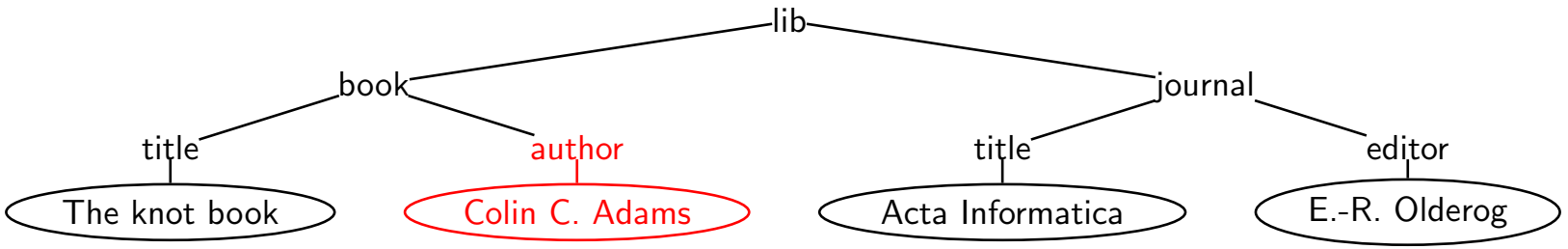


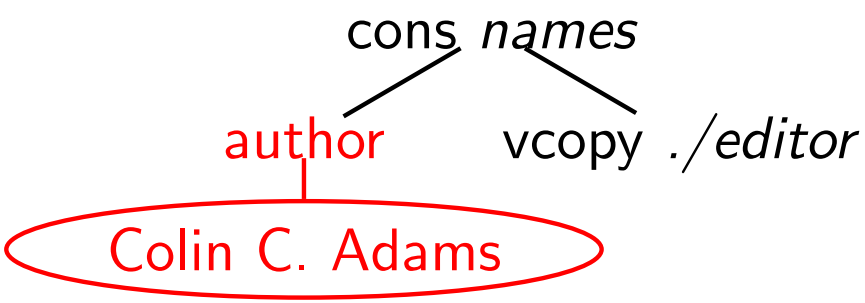
---

cons *names*

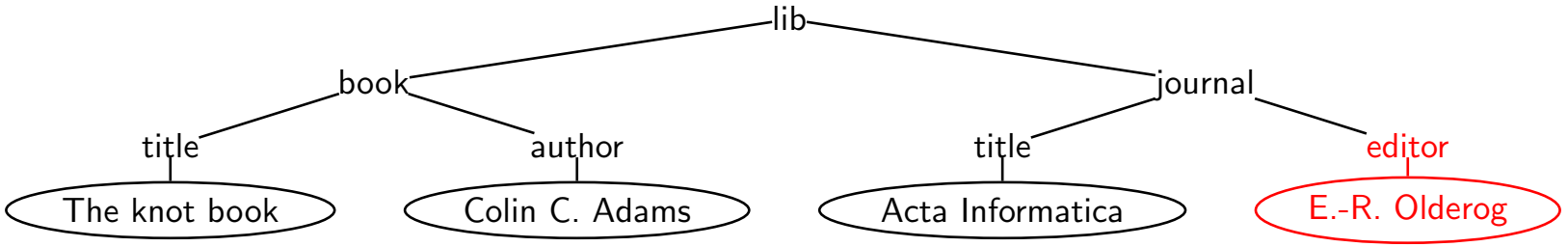
vcopy *./author* vcopy *./editor*

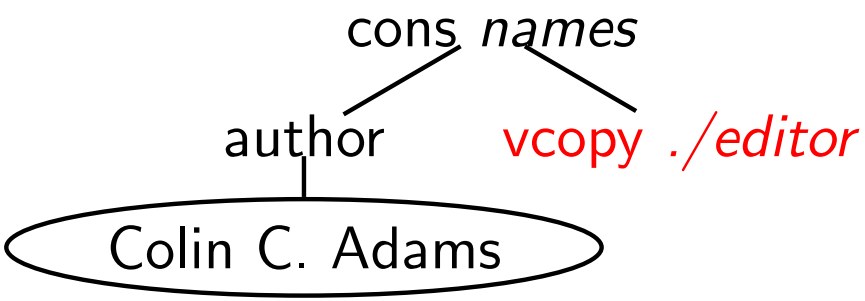
Example 12


lib
book
journal
title
author
title
editor
The knot book
Colin C. Adams
Acta Informatica
E.-R. Olderog

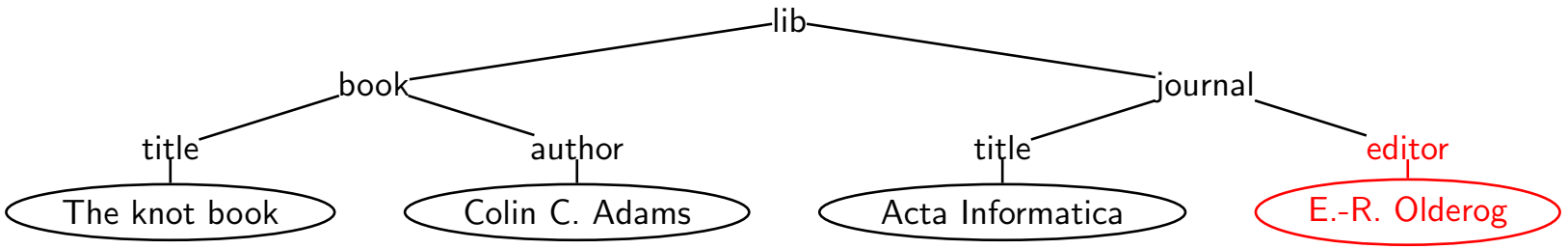

cons names
author
vcopy ./editor
Colin C. Adams

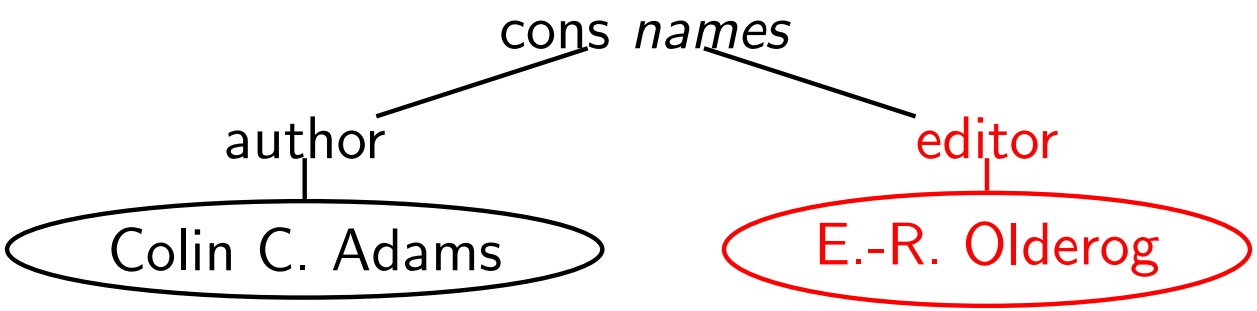
Example 13


lib
book
journal
title
author
title
editor
The knot book
Colin C. Adams
Acta Informatica
E.-R. Olderog

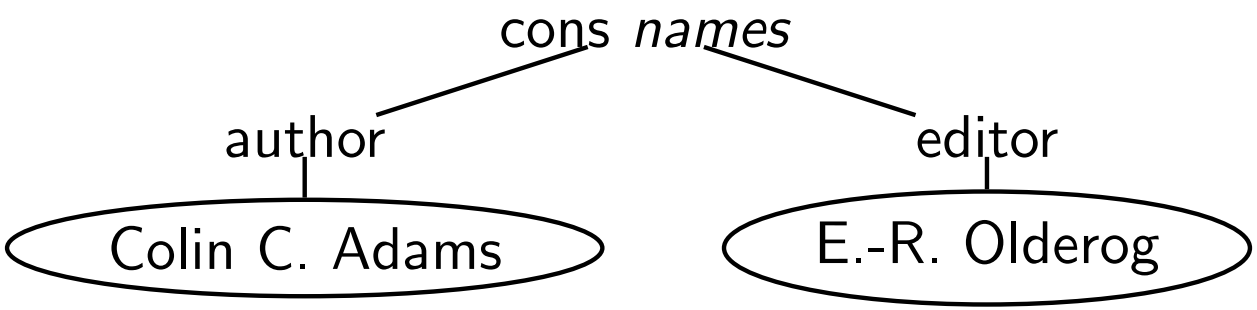

cons names
author
vcopy ./editor
Colin C. Adams

Example 14


lib
book
journal
title
author
title
editor
The knot book
Colin C. Adams
Acta Informatica
E.-R. Olderog


cons names
author
editor
Colin C. Adams
E.-R. Olderog

Example 15

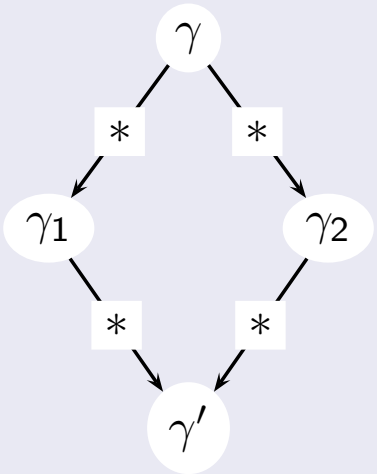


Confluence

Theorem

Our rewrite relation $\Rightarrow$ is confluent.

Definition of confluence



From the W3C Recommendation...

Implementations are free to process the source document in any way that produces the same result as if it were processed using this processing model.

Outline

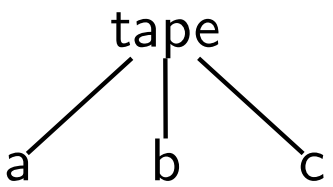
1. Formal semantics
2. Computational completeness of XSLT 2.0
   - Temporary Trees
3. Exponential time upper bound for XSLT 1.0

Computational power of XSLT 2.0

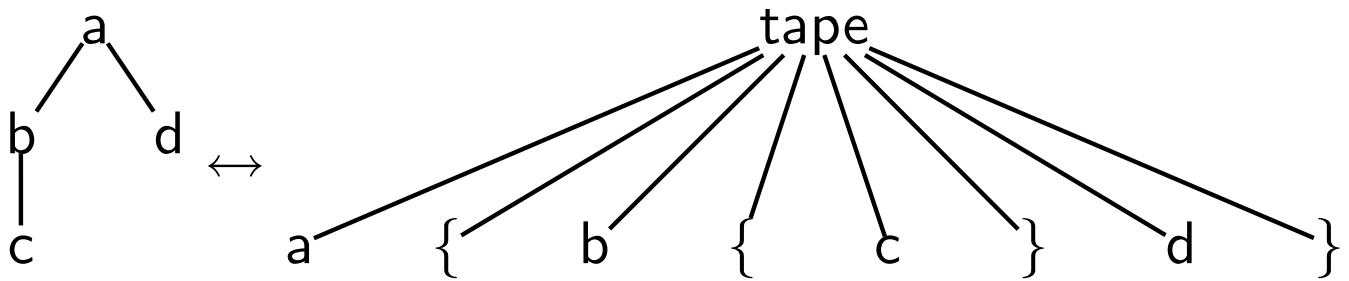
Theorem

Every computable tree transformation can be realised by a XSLT 2.0 program.

- Show that every turing machine (working on strings) can be simulated by some program.



- Show the transformations tree $\leftrightarrow$ tape (and vice versa)



XSLT 2.0 Requirements

From the XSLT 2.0 requirements...

"In addition, the following are explicitly not goals:

- ...
- Turning XSLT into a general-purpose programming language
- ...

"

Temporary Trees: a crucial feature of XSLT 2.0

```
<xsl:stylesheet version="2.0">
   <xsl:template match="/*">
      <xsl:variable name="var">
         <xsl:call-template name="t"/>
      </xsl:variable>

      <xsl:apply-templates select="$var"/>
   </xsl:template>

   <xsl:template name="t">
      ...
   </xsl:template>

   ...

</xsl:stylesheet>
```

- not in XSLT 1.0!

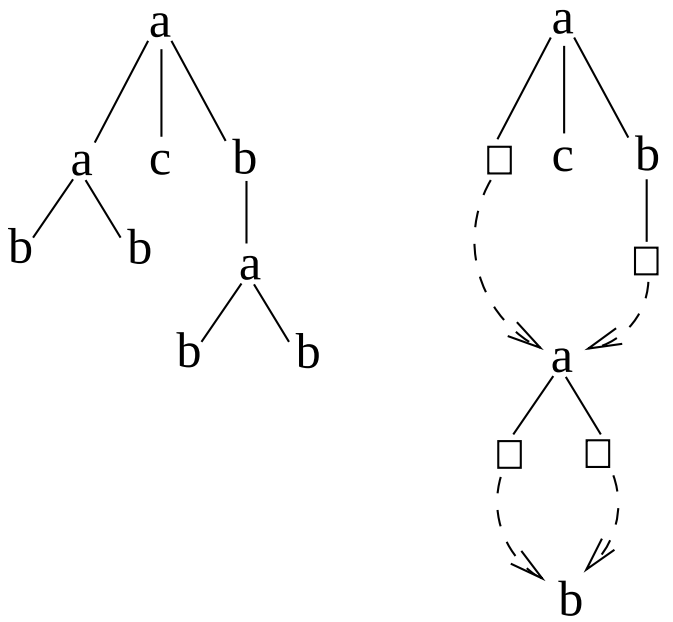
Outline

1. Formal semantics
2. Computational completeness of XSLT 2.0
   - Temporary Trees
3. Exponential time upper bound for XSLT 1.0

Computational power of XSLT 1.0

Theorem

All tree transformations computable in XSLT 1.0 are computable in at most (singly) exponential time. (Using a DAG-representation of the output tree.)



Corollary

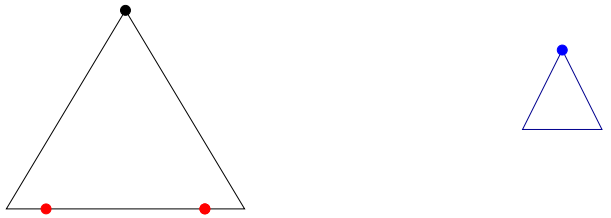
XSLT 1.0 is not computationally complete!

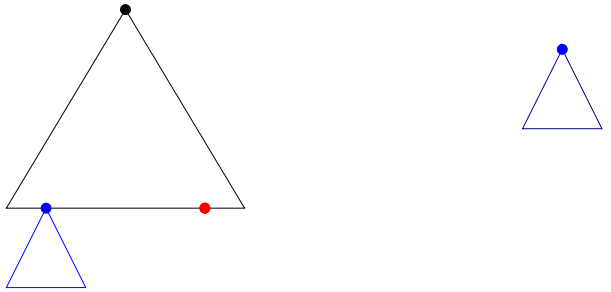
Proof.

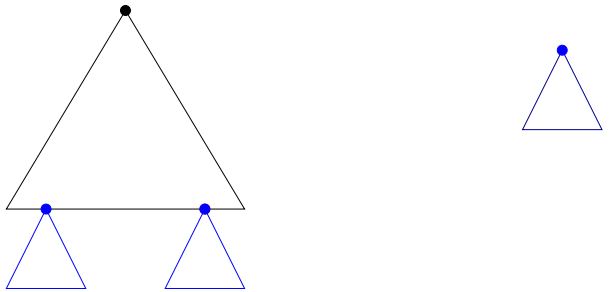
- A program has a *fixed* number of templates
- Each template can be called with different contexts
- Given input of size n, the number of contexts is $2^{n^{O(1)}}$
- Create a *configuration* for each combination of a template and a context
- Implement template calls by pointers to configurations

□

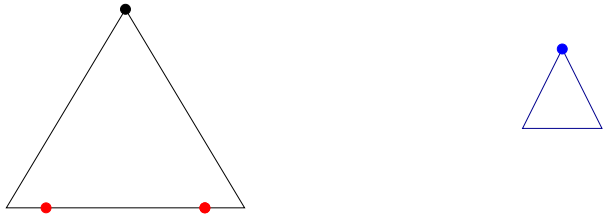
DAG

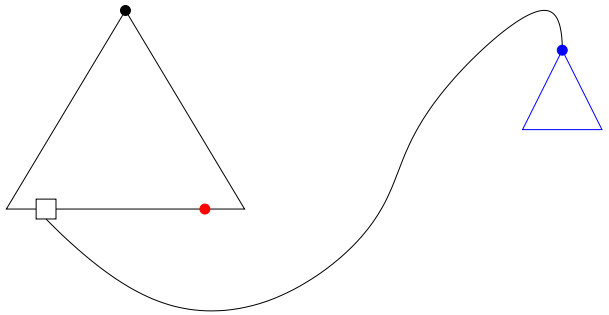


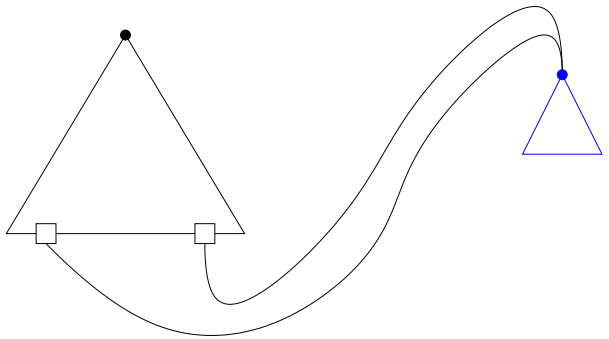




DAG







Corollary

XSLT 1.0 is not closed under composition.

Proof.

- Program 1 creates a tree of doubly exponential size
- Program 2 creates a tree of exponential size
- Composition creates a tree of triply exponential size

$\Rightarrow$ DAG representation of triply exponential large tree computable in exponential time!
(contradiction)

□

Conclusions

- Formal syntax and semantics for XSLT
  - Confluence
- Computational completeness for XSLT 2.0
- Exponential time upper bound for XSLT 1.0
  - there exists XSLT 1.0 transformations that are PSPACE-complete!
    $\Rightarrow$ time upper bound unlikely to be improved
  - open question: is each XSLT 1.0 transformation in PSPACE?