IBM Zurich Research Laboratory | Business Integration Technologies

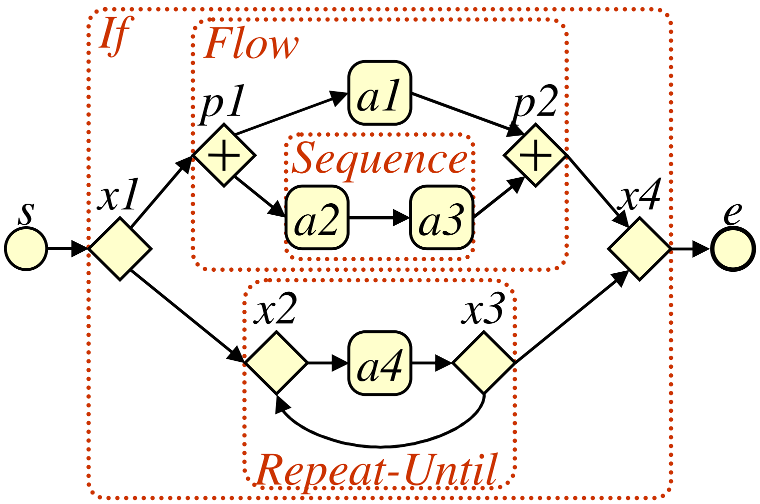
The Refined Process Structure Tree

Jussi Vanhatalo juv@zurich.ibm.com
Hagen Völzer hvo@zurich.ibm.com
Jana Koehler koe@zurich.ibm.com

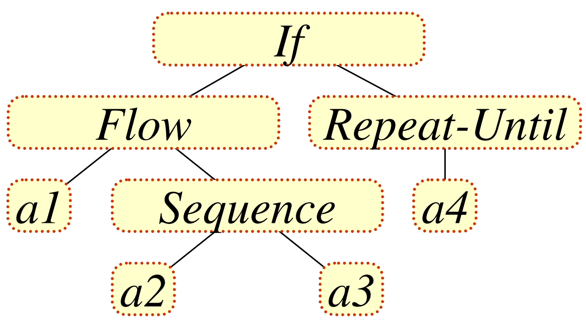
The 6th International Conference on Business Process Management (BPM 2008)

September 2008

Motivation: BPMN to BPEL Translation



Business Process Modeling Notation (*BPMN*)



Parse tree

```
<process ...>
  ...
  <if>                                                If
    <condition>...</condition>
    <flow>                                            Flow
      <invoke name="a1" ... />                        a1
      <sequence>                                      Sequence
        <invoke name="a2" ... />  a2
        <invoke name="a3" ... />  a3
      </sequence>
    </flow>
    <else>
      <repeatUntil>                                   Repeat-Until
        <invoke name="a4" ... />    a4
        <condition>...</condition>
      </repeatUntil>
    </else>
  </if>
</process>
```

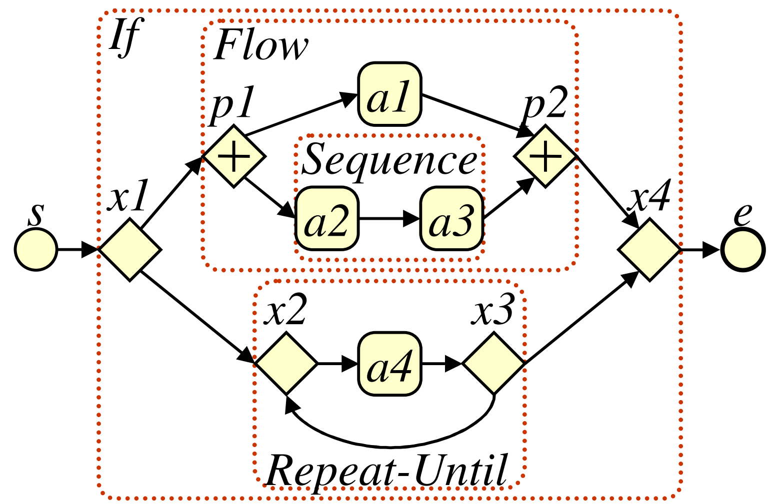
Business Process Execution Language (*BPEL*)

Research Problem: Parsing a Business Process Model

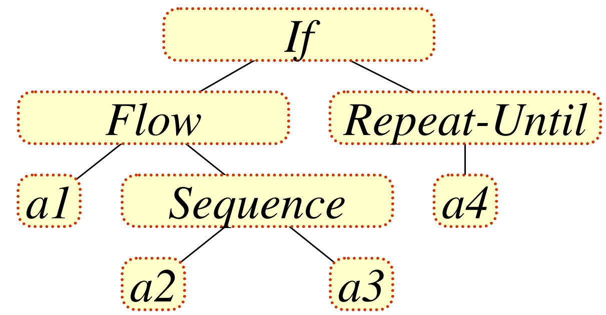
- ***Parsing***
  1) Decomposition into *fragments*
  2) Categorization of the fragments

  → *Parse tree*

- Our contribution is a new parsing technique
  – ***Refined process structure tree*** (*RPST*)
  – Improves existing techniques by providing a more fine-grained decomposition



Process model in BPMN



Parse tree

Outline

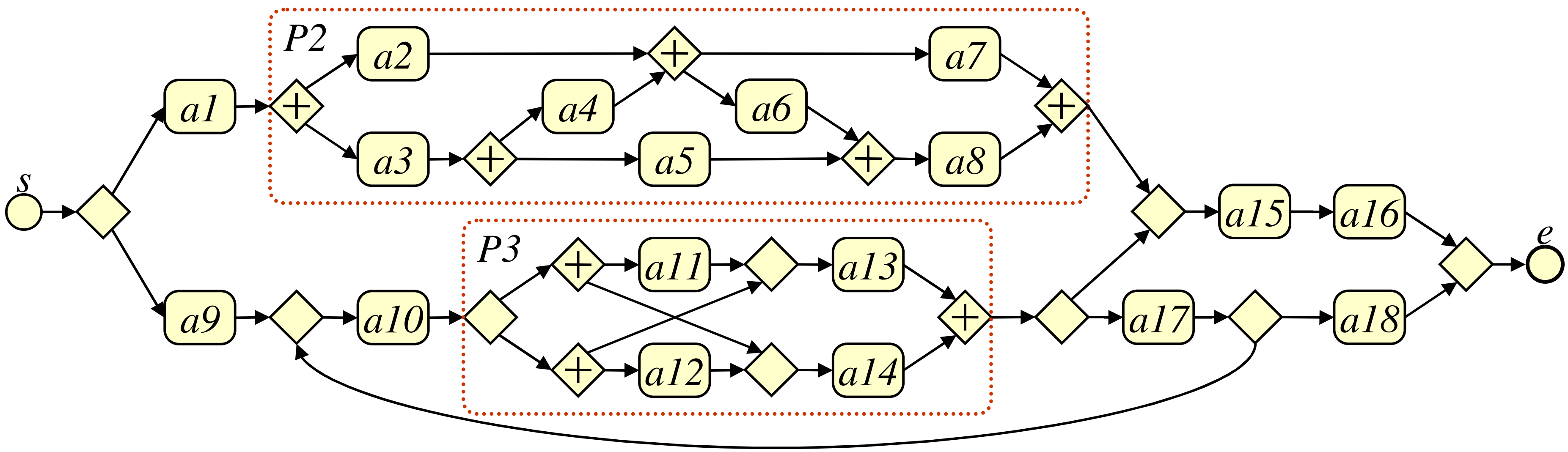
- Research Problem: Parsing a Business Process Model
- Use Cases for Parsing
- Requirements for Parsing and the Related Work
- Our Solution: The Refined Process Structure Tree

Use Cases for Parsing

- Translating a graph-based process model (e.g. BPMN) into a block-based process model (e.g. BPEL)
- Speeding up control-flow analysis [Vanhatalo et al., 2007]
- Pattern-based editing [Gschwind et al., 2008; Today 11:00 am]
- Process merging [Küster et al., 2008; Tomorrow 16:00 am]
- Understanding large process models
- Subprocess detection

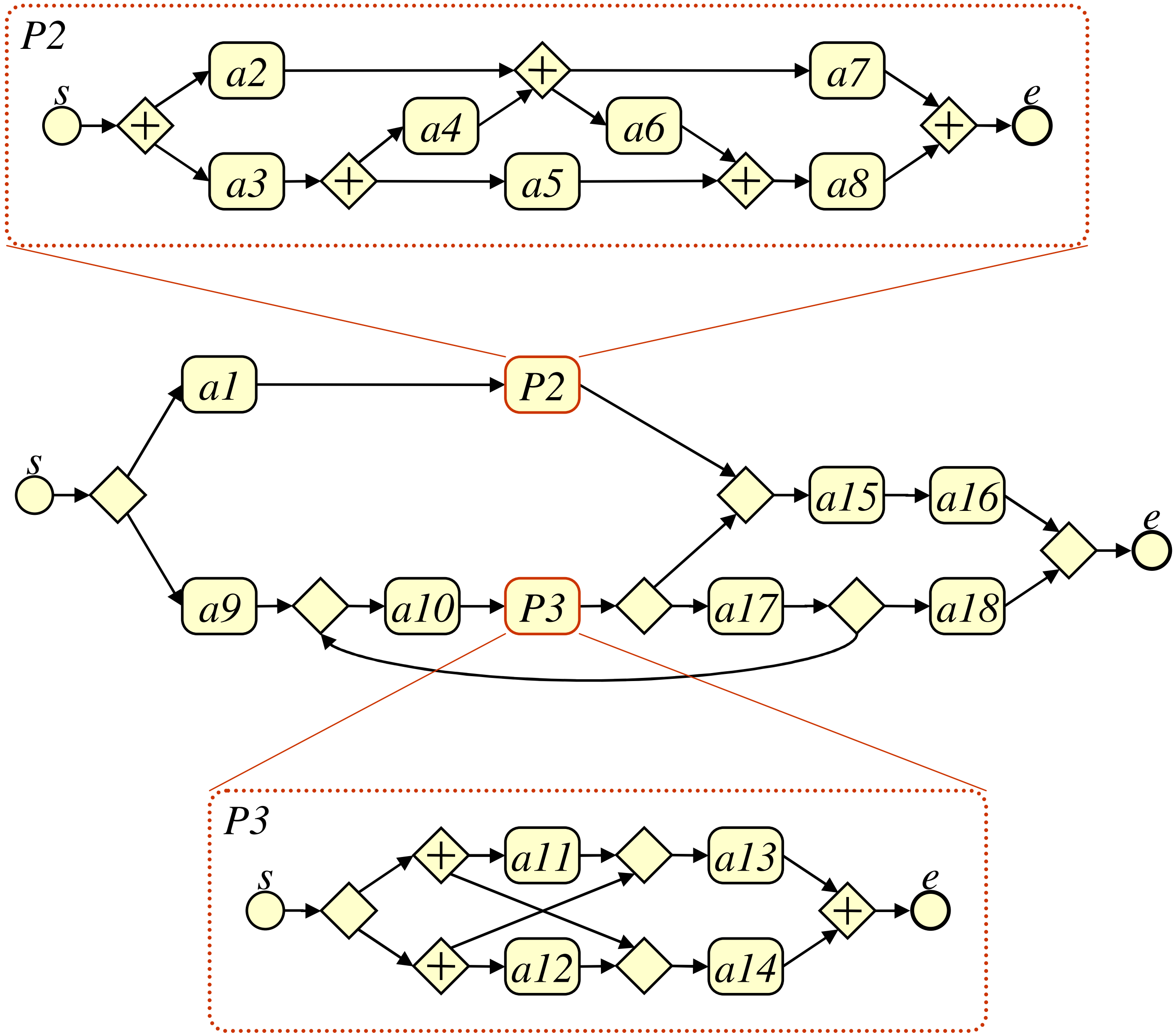
Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Subprocess Detection



Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Subprocess Detection

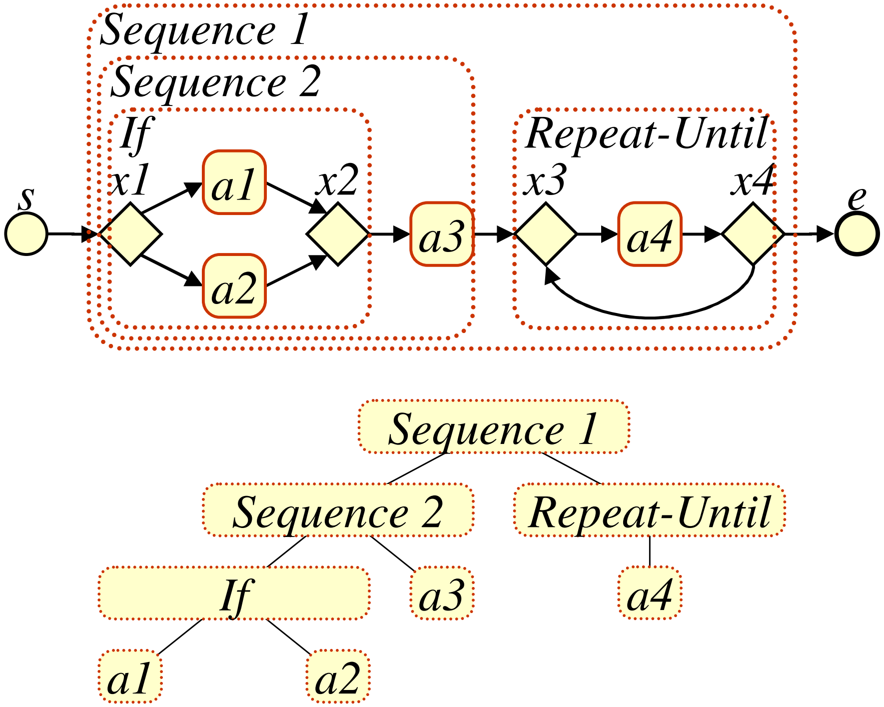


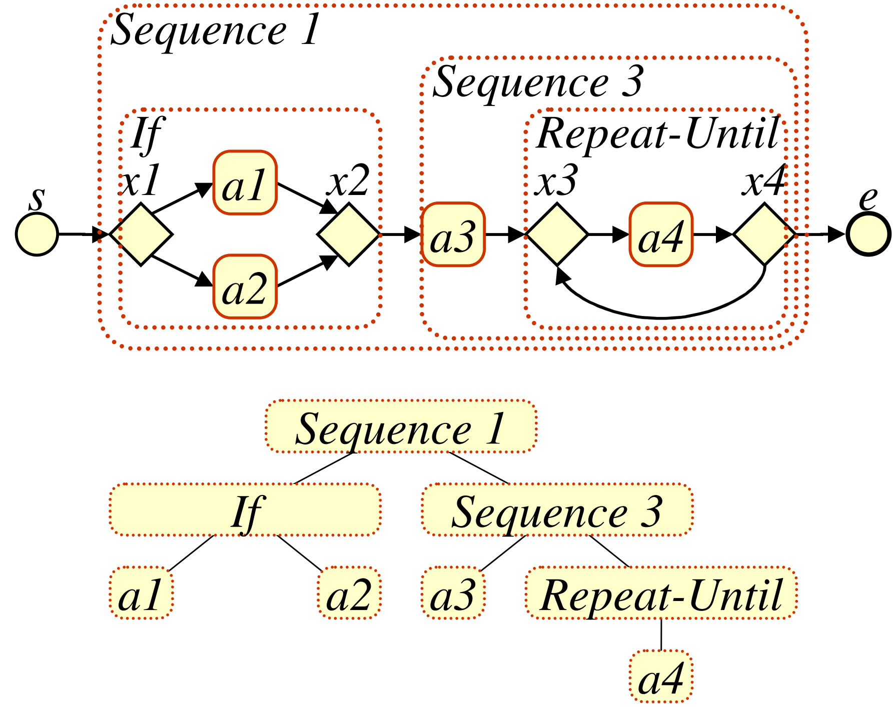
Outline

- Problem: Parsing a Business Process Model
- Use Cases for Parsing
- Requirements for Parsing and the Related Work
  - Uniqueness
  - Modularity
  - Computing the Parse Tree Fast
  - Granularity
- Our Solution: The Refined Process Structure Tree

Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Requirement: Uniqueness

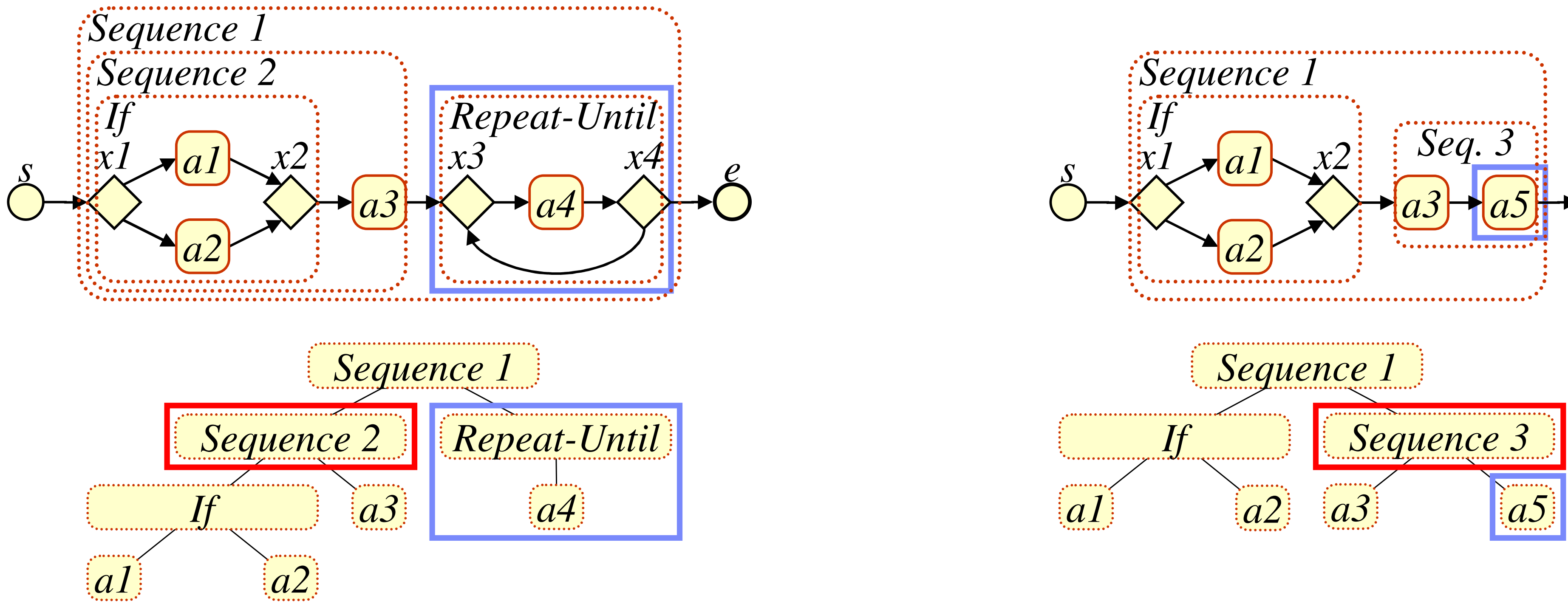




- The parse tree should be **unique**
  - Motivation: The same BPMN diagram is always translated to the same BPEL process
- Parsing techniques presented for BPMN to BPEL translations are not unique
  - Nondeterministic pattern-matching approach

Jussi Vanhatalo, Hagen Völzer, Jana Koehler

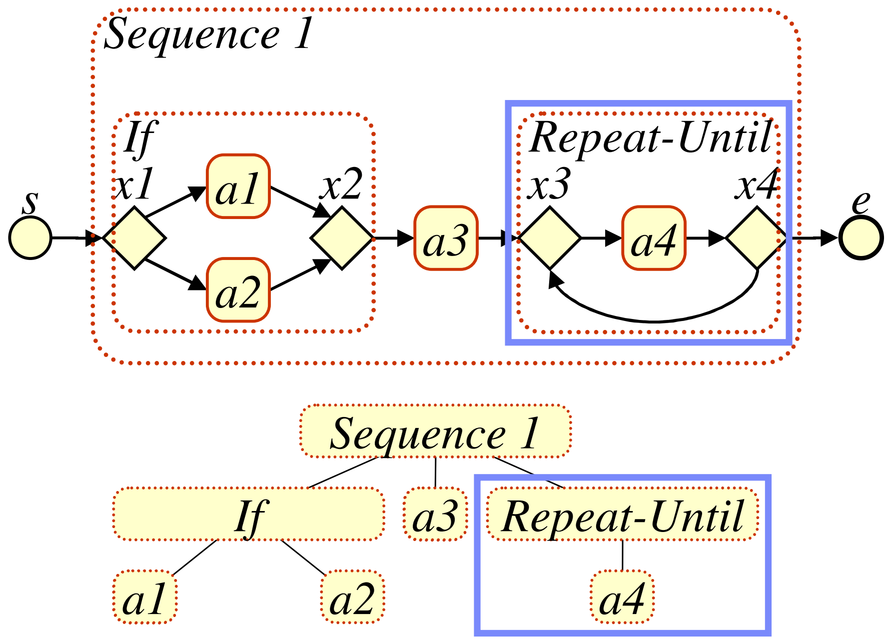
Requirement: Modularity

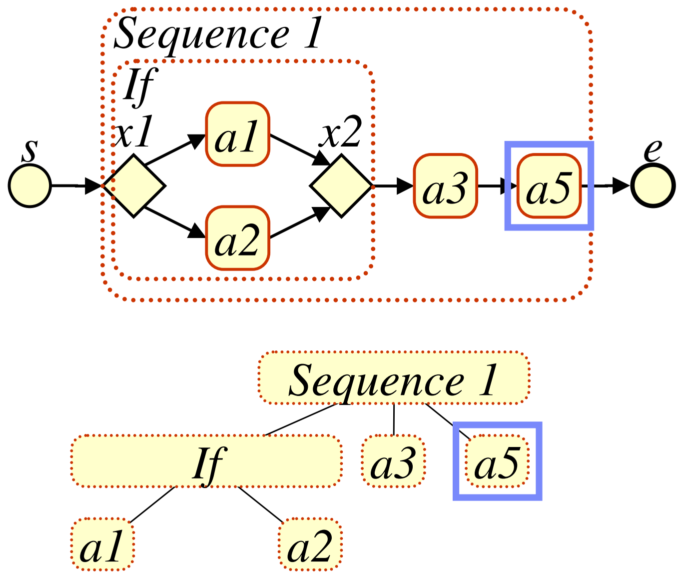


- Motivation: A local change in BPMN translates into a local change in BPEL
- *Modular*:
  - Replacing a fragment with another fragment changes only the respective subtree in the parse tree
- Parsing techniques presented for BPMN to BPEL translations are not modular

Jussi Vanhatalo, Hagen Völzer, Jana Koehler

The Normal Process Structure Tree (NPST)

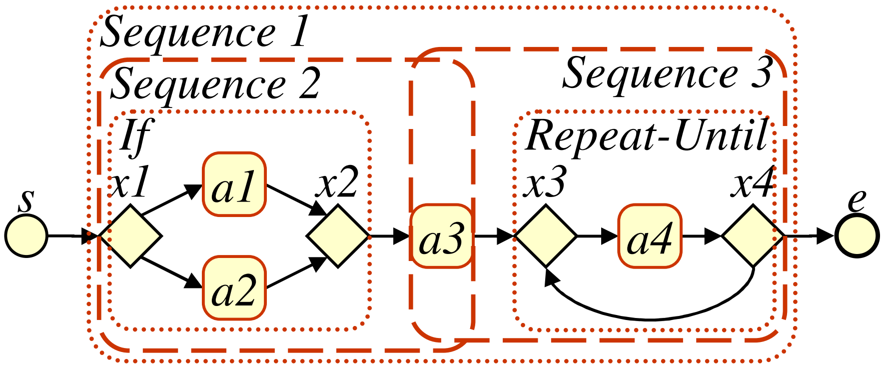


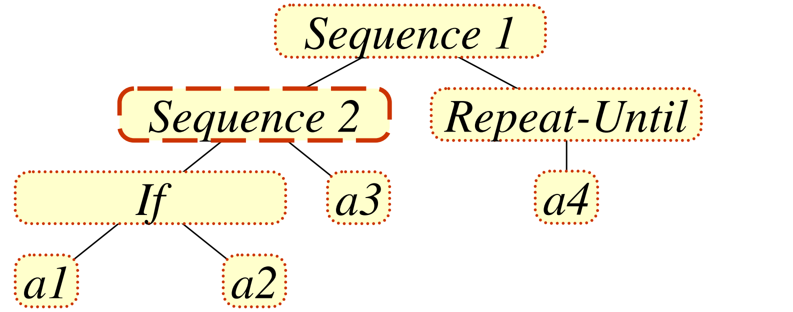


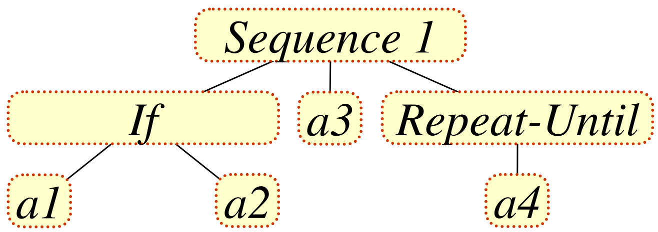
- The **NPST** is unique and modular
  - Extends work on the *program structure tree* [Johnson et al., 1994]
  - Adapted for process models [Vanhatalo, Völzer and Leymann, 2007]

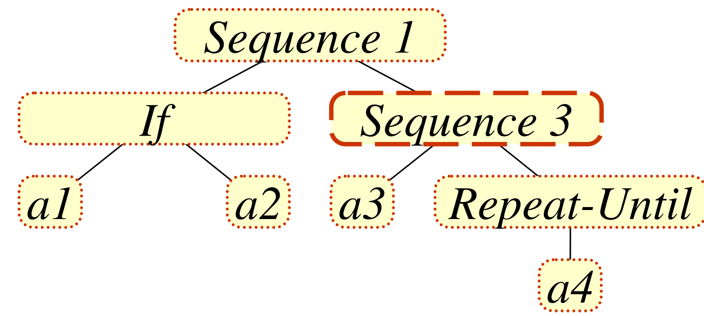
Jussi Vanhatalo, Hagen Völzer, Jana Koehler

The NPST is the Hierarchy of the Canonical Fragments









- Parse tree is a hierarchy of fragments in which any two fragments **do not overlap**

  → Some fragments must be excluded a parse tree
- What makes the NPST different from the non-deterministic parse trees?
  - Each fragment that overlaps some other fragment is excluded from the NPST
    - Such a fragment is called **non-canonical**
    - The non-maximal sequences are the non-canonical fragments

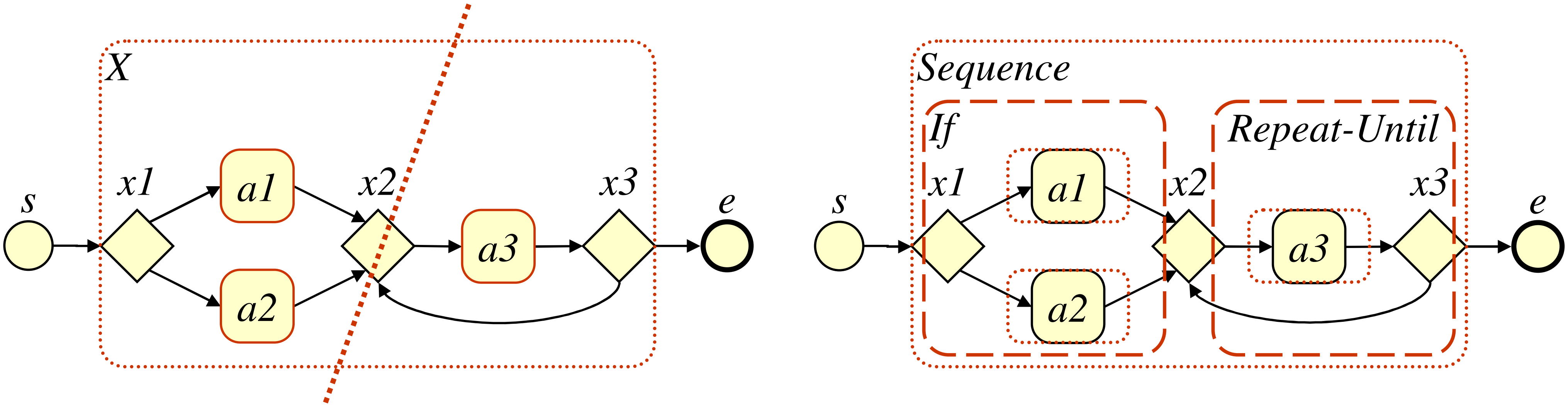
Requirement: Computing the Parse Tree Fast

- Some use cases require a fast algorithm for computing the parse tree
  - **Process version merging**
    - Process models are compared based on their parse trees
    - Change operations are applied to merge the process models
    - Each time a process model changes, the parse tree is recomputed
  - **Pattern-based editing**
    - Some editing operations are applicable/prevented based on the information in the parse tree
  - **Speeding up control-flow analysis**

- The NPST can be computed in linear time

Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Requirement: Granularity



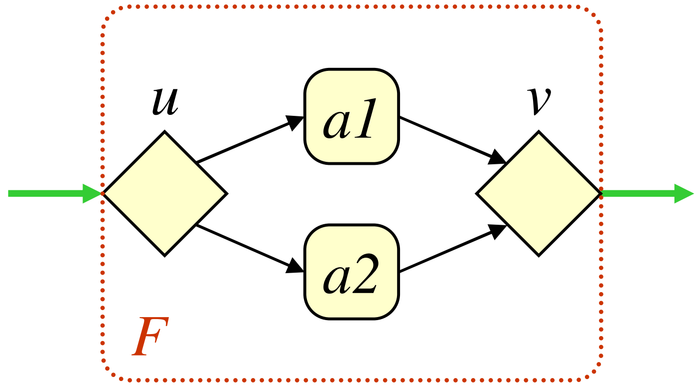
- Motivation: Translate more BPMN diagrams into BPEL
- Our new contribution is the **refined process structure tree**
  - Extends work on a *parse tree* for sequential programs [Tarjan and Valdes, 1980]
  - More fine-grained than any previous technique

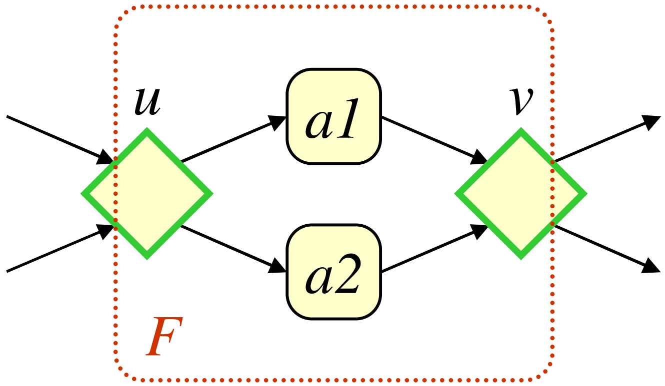
Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Outline

- Problem: Parsing a Business Process Model
- Use Cases for Parsing
- Requirements for Parsing and Related Work
- Our Solution: The Refined Process Structure Tree
  - Relaxed Notion of a Fragment
  - Canonical Fragments
  - The Refined Process Structure Tree
  - Uniqueness, Modularity, Granularity
  - A Linear Time Algorithm

Relaxed Notion of a Fragment





The commonly used notion:

- A *fragment* is a connected subgraph that has
  - exactly one entry edge, and
  - exactly one exit edge.

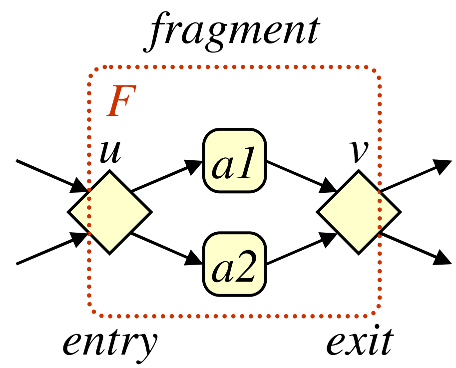
Relaxed notion:

- A *fragment* is a connected subgraph that has
  - exactly one entry node, and
  - exactly one exit node.

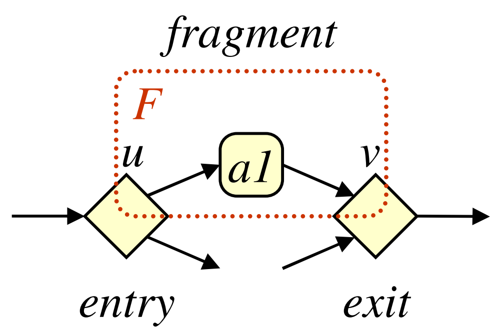
Jussi Vanhatalo, Hagen Völzer, Jana Koehler

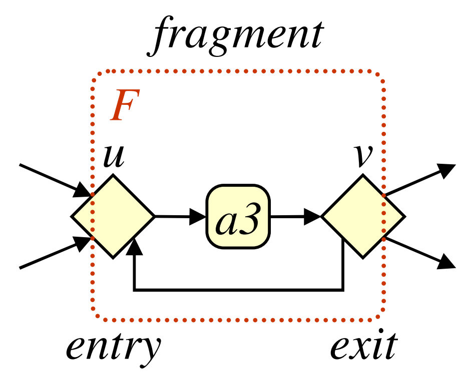
More Precisely:

- If anything inside a fragment F is executed, then
  - the entry node was executed before, and
  - the exit node will be executed afterwards



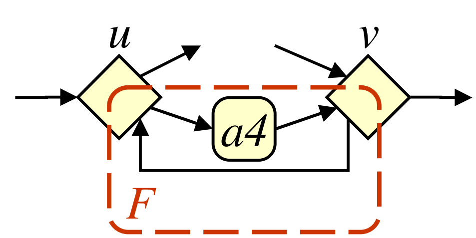
- A boundary node is an ***entry*** if
  - all incoming edges are outside F, or
  - all outgoing edges are inside F
- A boundary node is an ***exit*** if
  - all incoming edges are inside F, or
  - all outgoing edges are outside F





- A ***fragment*** F is a connected subgraph that has
  - exactly two boundary nodes,
  - one entry, and one exit
- [Tarjan and Valdes, 1980]

Not a fragment!

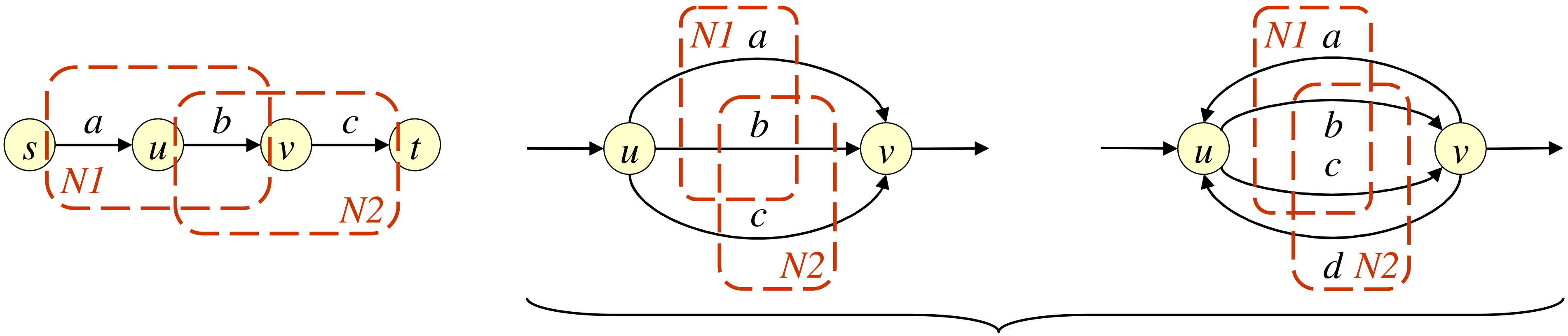


These boundary nodes are neither entries nor exits

Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Non-Canonical and Canonical Fragments

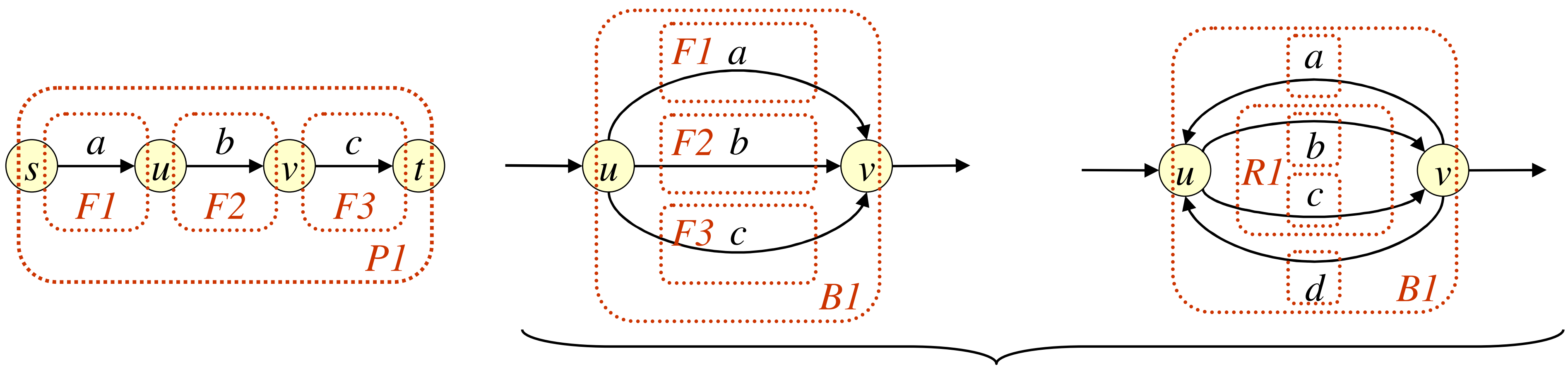
- ***Non-canonical*** fragments overlap with some fragment



non-maximal sequences

non-canonical bond fragments

- ***Canonical*** fragments do not overlap and thus they form a hierarchy



maximal sequence

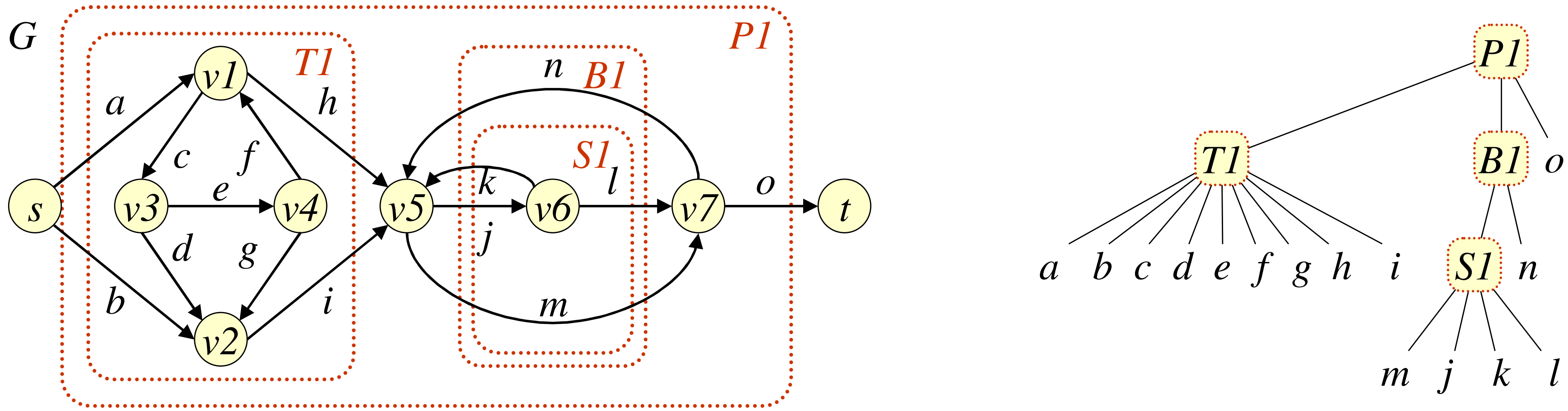
canonical bond fragments

Jussi Vanhatalo, Hagen Völzer, Jana Koehler

The Refined Process Structure Tree

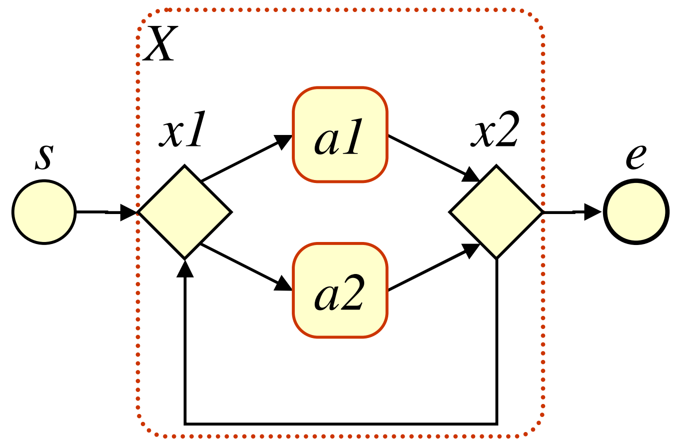
- As the canonical fragments do not overlap, they form a hierarchy.

- The ***refined process structure tree*** is the tree of canonical fragments of a process model G, such that the parent of a canonical fragment F is the smallest canonical fragment of G that properly contains F.

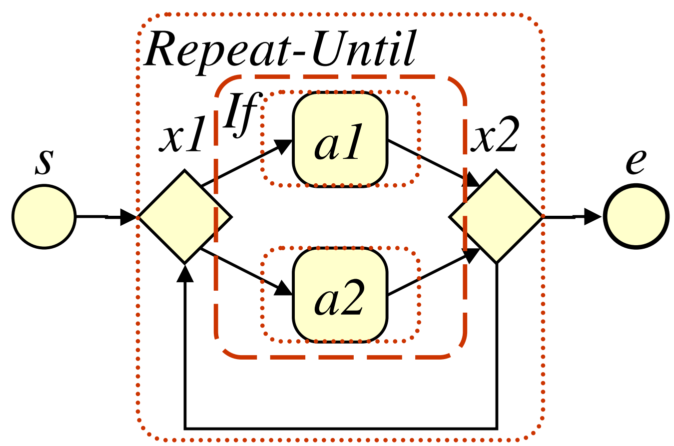


Properties of the Refined Process Structure Tree

- The RPST is:
  - Unique
  - Modular
  - More fine-grained than
    - the NPST
    - the parse tree by Tarjan and Valdes
- It can be computed in linear time

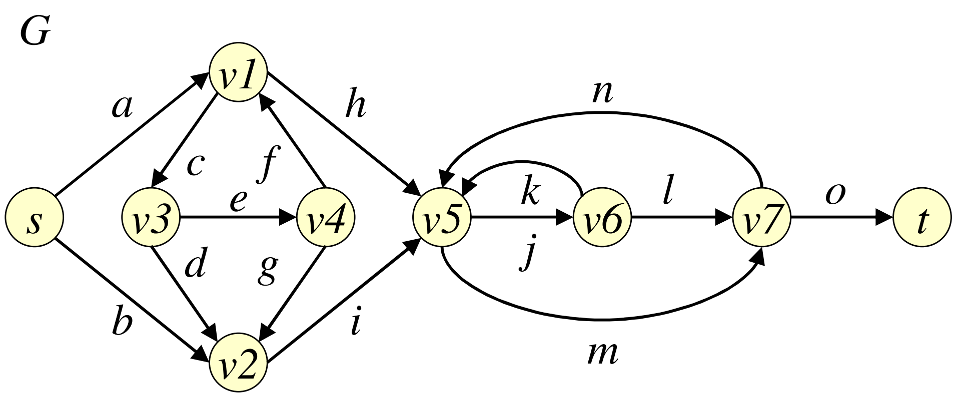


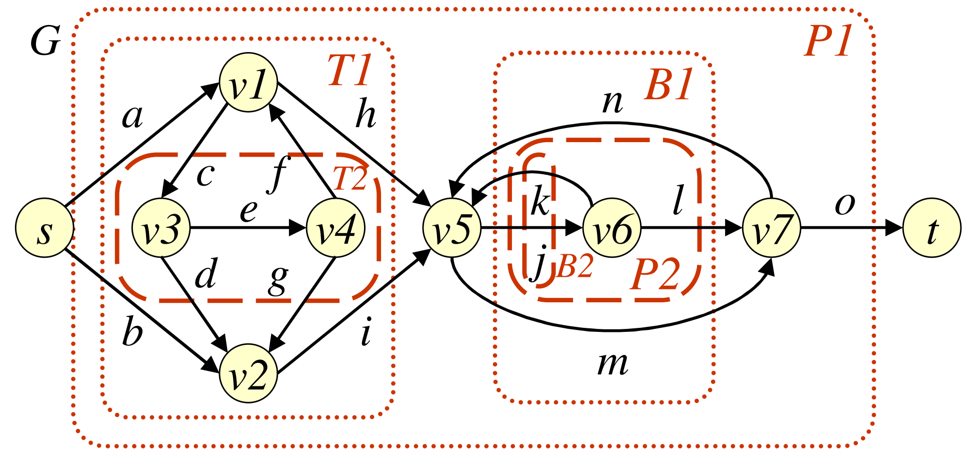
Fragments in the NPST



Fragments in the RPST

A Linear Time Algorithm for Computing the RPST

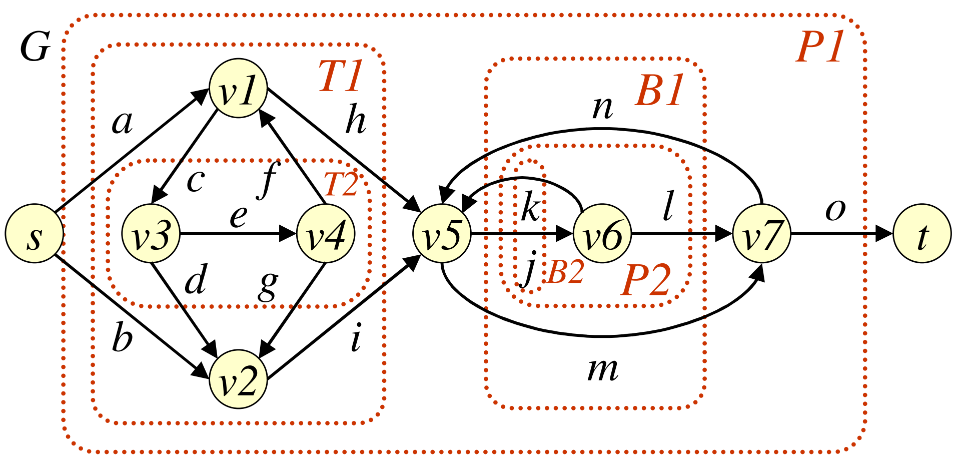


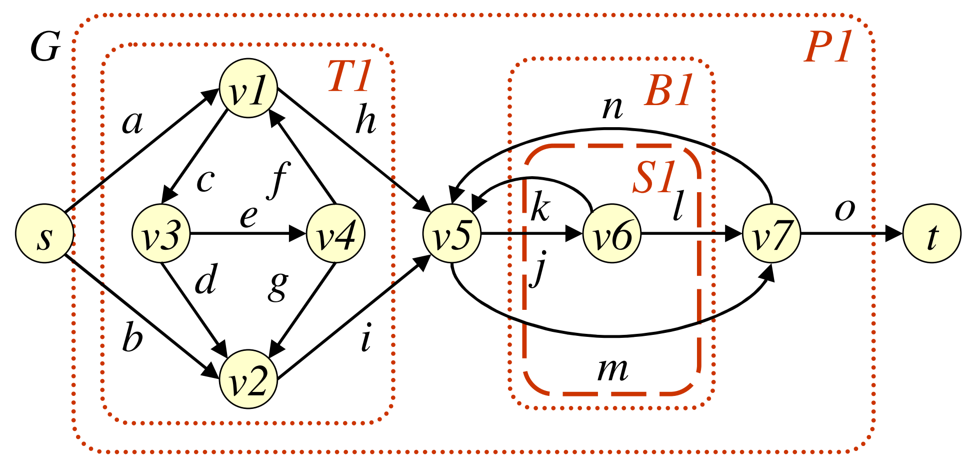


Step 1: Detect the triconnected components.

Step 2: Check whether each triconnected component is a fragment.

Step 3: Restructure the tree into the RPST.

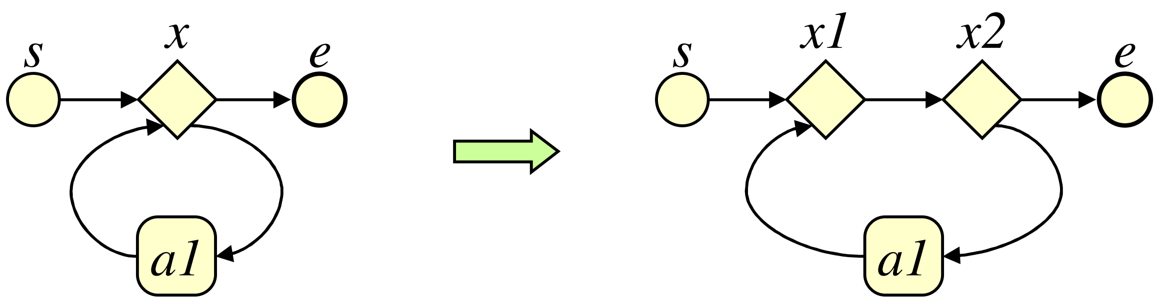




Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Generalized Theory

- In this paper, we assumed two restrictions for process models to simplify the presented theory
  - Exactly one start node and exactly one end node
  - Loops must have separate entry and exit node



- We have generalized this theory for arbitrary process models
  - This will published in an extended version of this paper

Jussi Vanhatalo, Hagen Völzer, Jana Koehler

Conclusions

- Parsing business process models
  - Many interesting use cases
  - Requirements for a parsing technique
    - Uniqueness, modularity, granularity, fast computation
- A new parsing technique called the ***refined process structure tree***
  - Improves existing techniques by providing a more fine-grained decomposition
  - Unique, and modular
  - Can be computed in linear time
- Future work: Applying the RPST for different use cases

References

- [HT73] J. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2:135–158, 1973.
- [Val78] Jacobo Valdes Ayesta. *Parsing flowcharts and series-parallel graphs.* PhD thesis, Stanford, CA, USA, 1978.
- [TV80] Robert E. Tarjan and Jacobo Valdes. Prime subprogram parsing of a program. In *POPL '80: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 95–105, New York, NY, USA, 1980. ACM.
- [JJP94] Richard Johnson, David Pearson, and Keshav Pingali. The program structure tree: Computing control regions in linear time. In *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI)*, pages 171–185, 1994.
- [Joh95] Richard Craig Johnson. *Ecient program analysis using dependence flow graphs*. PhD thesis, Ithaca, NY, USA, 1995.
- [GM00] Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In Joe Marks, editor, *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2000.
- [VVL07] Jussi Vanhatalo, Hagen Völzer, and Frank Leymann. Faster and more focused control-flow analysis for business process models though SESE decomposition. In *5th International Conference on Service-Oriented Computing (ICSOC)*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer-Verlag Berlin Heidelberg, September 2007.