Data-flow based Model Analysis

Christian Saad and Bernhard Bauer

Distributed Systems Programming

Christian Saad
(*saad@informatik.uni-augsburg.de*)

Bernhard Bauer
(*bauer@informatik.uni-augsburg.de*)

Motivation

(Meta) Modeling allows to formally capture the syntactical structure of application domains

but

Validation of (static) semantics is still a challenge

Motivation

(Meta) Modeling allows to formally capture the syntactical structure of application domains

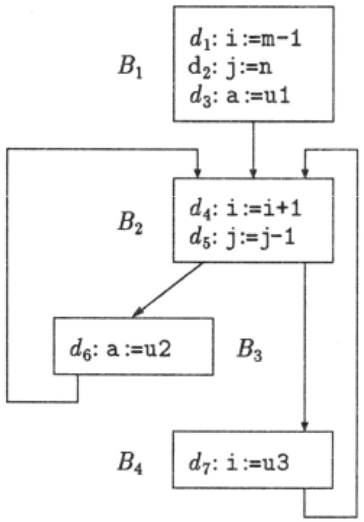
but

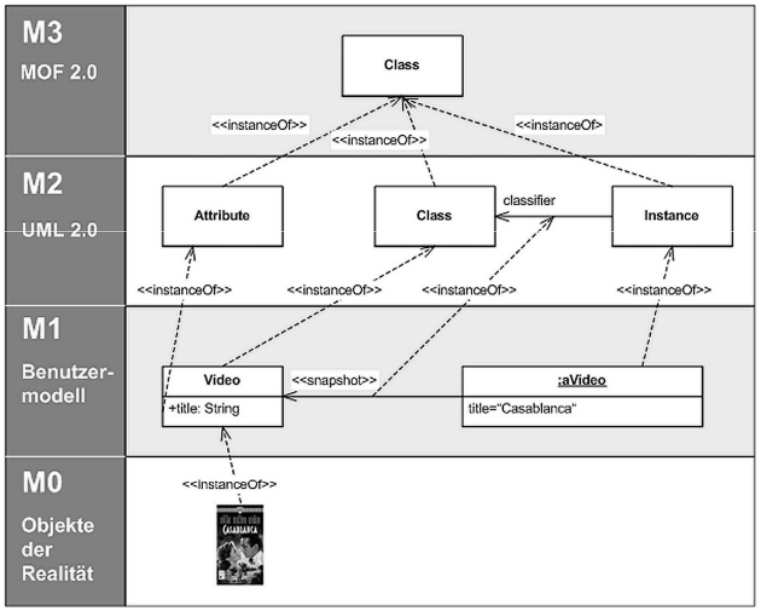
Validation of (static) semantics is still a challenge

$\Rightarrow$ Use Data-flow Analysis to enable context-sensitive model analysis

- Data-flow Analysis
  - Used in Compiler Construction
  - Calculates information flow in directed graphs
  - Approximate min/max flow for cyclic paths

```
/* d1 */  i := m-1;
/* d2 */  j := n;
/* d3 */  a := u1;
          do
/* d4 */        i := i+1;
/* d5 */        j := j-1;
              if e1 then
/* d6 */              a := u2
              else begin
/* d7 */                    i := u3;

                  end
          while e2
```

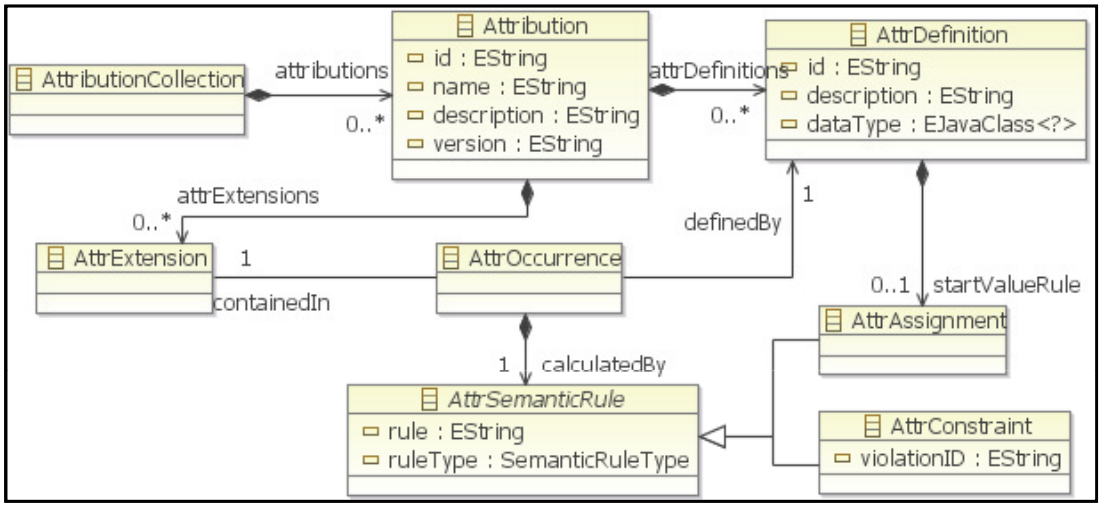


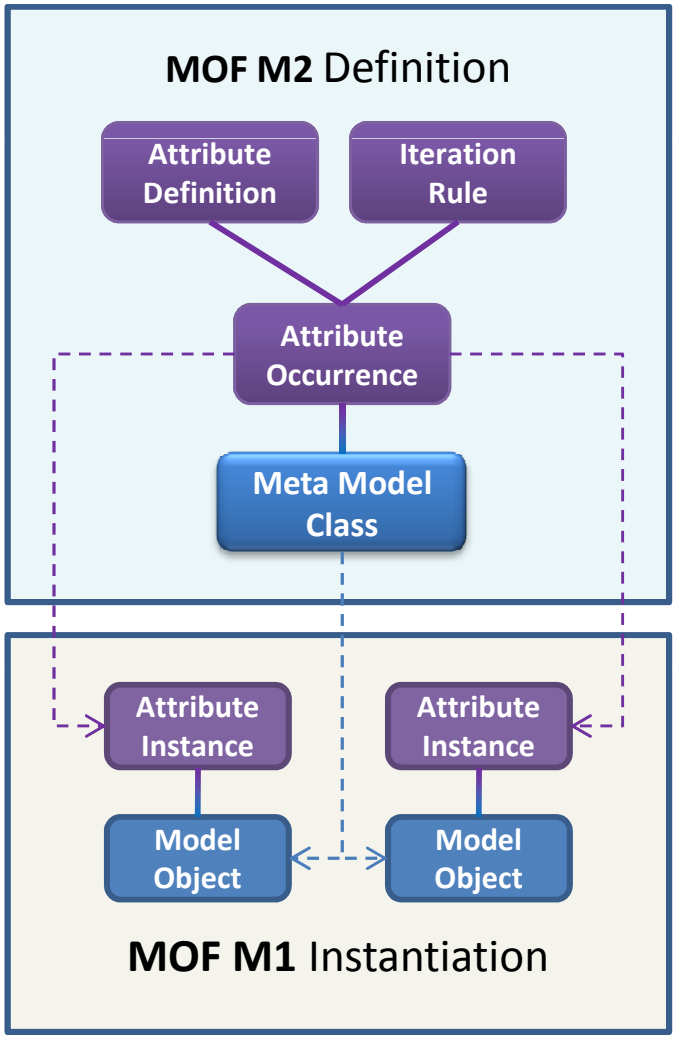


Definition

Stay in the **Modeling Domain**!

- DFA definition **based on Meta Model**
- Respect concepts like **Generalization** etc.
- Inspired by **Attribute Grammars**

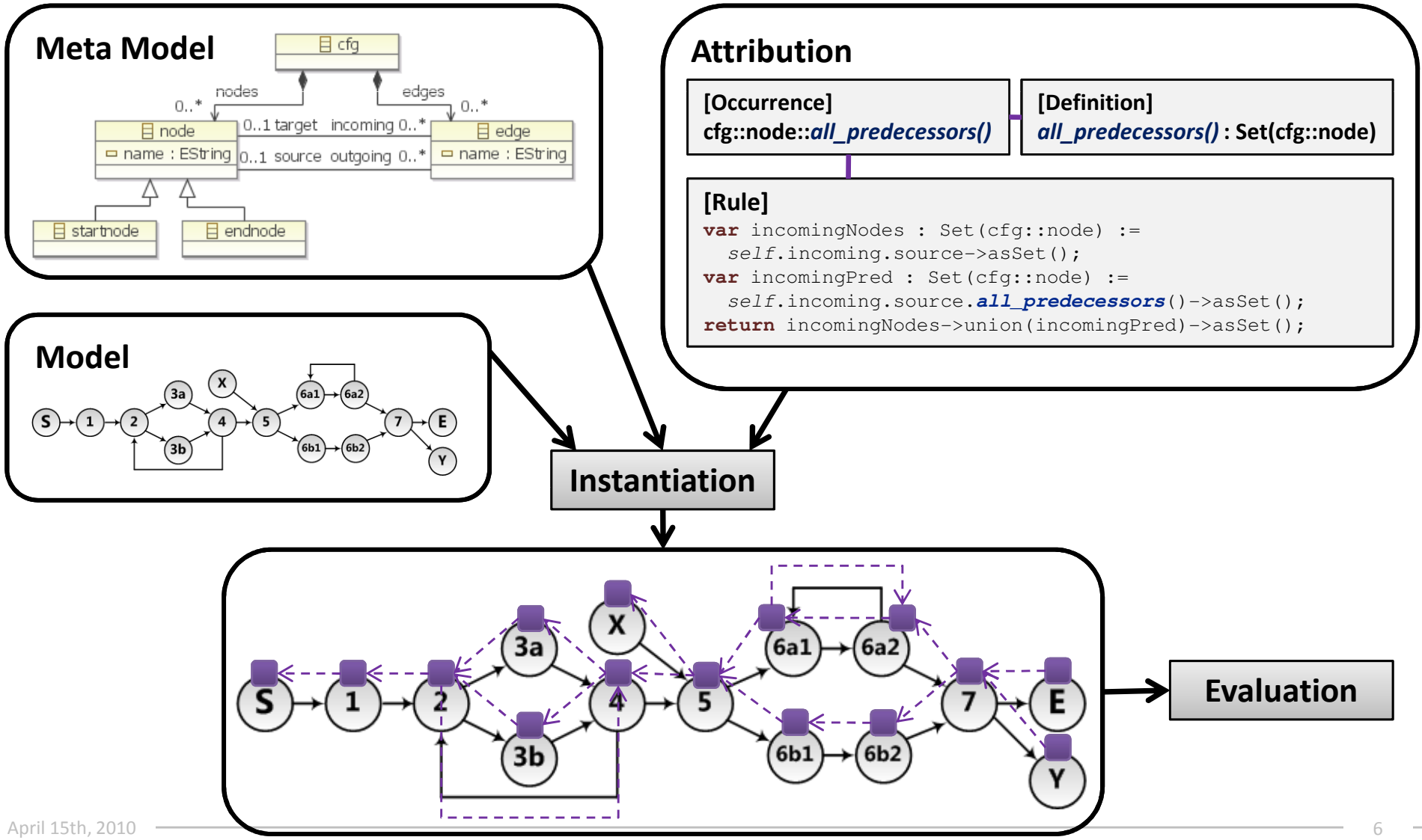




DFA equations/rules

- **Java**-based or
- **Extended (Imperative) OCL**

Control-Flow Graph - Calculate *transitive predecessors*



Evaluation

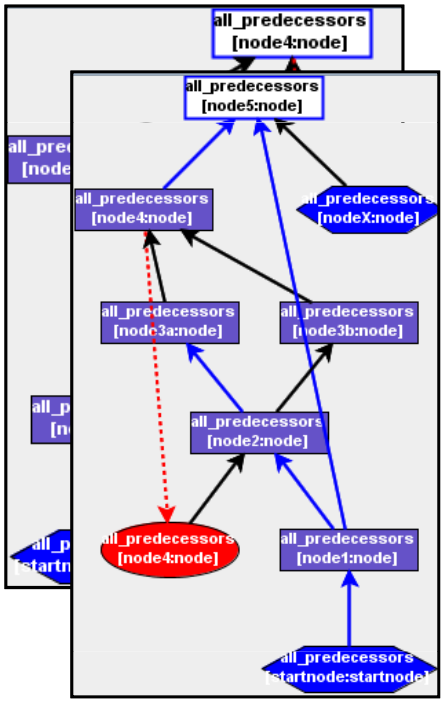
- **Traditional DFA Evaluation**
  - Worklist algorithm (execute rule, add depending rules to worklist)
  - Problem: Requests dynamic, output dependencies not known!

- **Dependency Chain Algorithm**
  - Uses Tree-like Directed-Acyclic Graph structure

  1. Build Chain
     - **Record dependencies** and **execute** requested rule recursively
     - Replace **cyclic dependencies** with virtual nodes
     - **Merge chains** if necessary
  2. Evaluate Chain
     - **Identify unstable** virtual nodes
     - Perform **bottom-up reevaluation** until stable
     - Adjust dependency chain if necessary

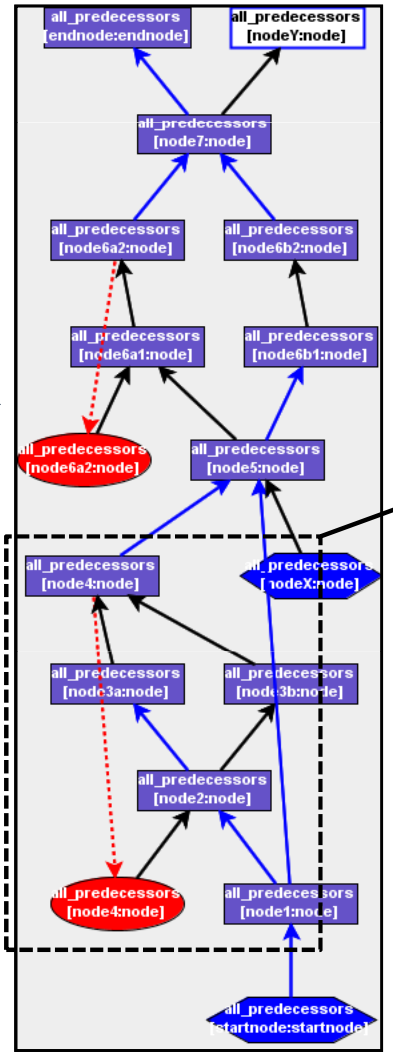
Evaluation

Build and Merge

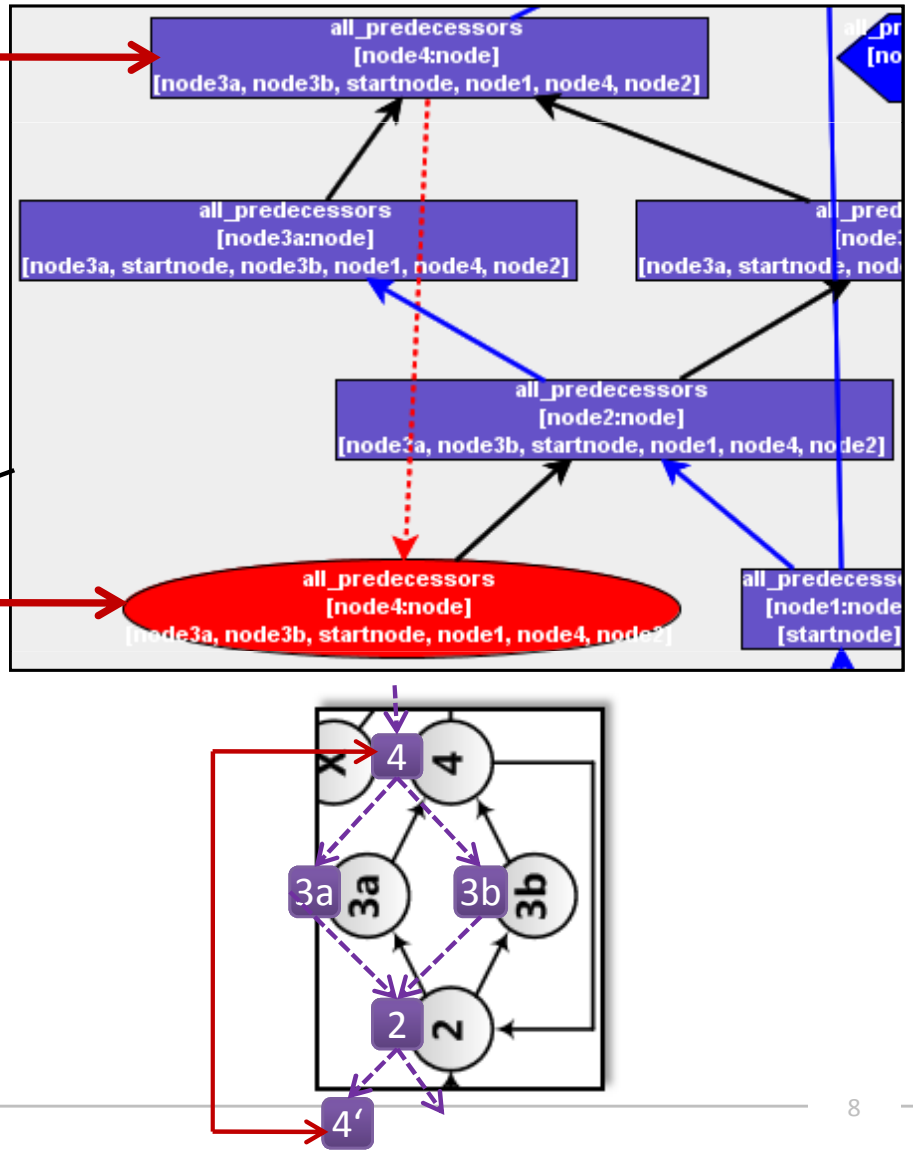


...

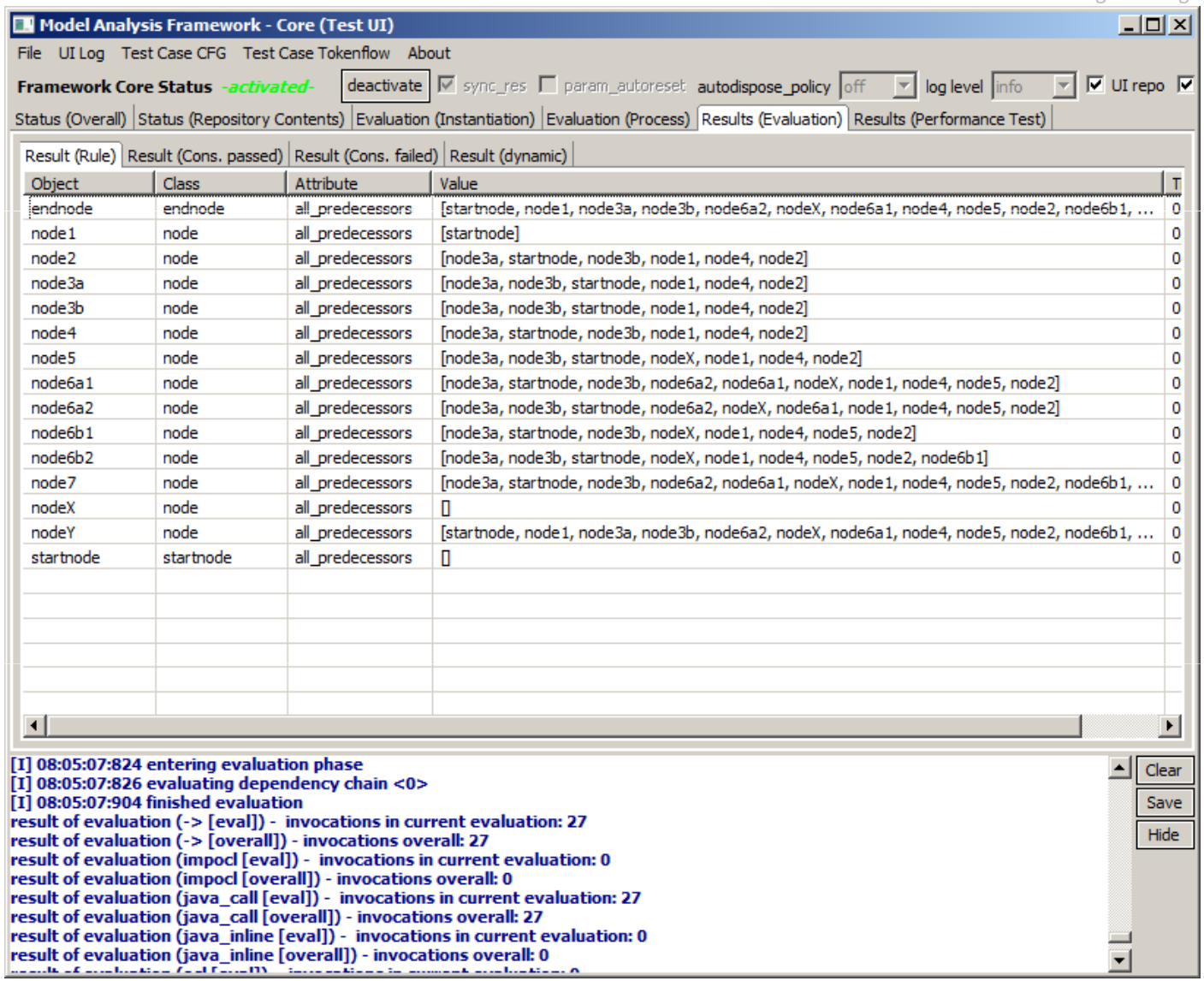
Built



Reevaluate until values are stable



Evaluation



http://code.google.com/p/model-analysis-framework/

Control-Flow Graph Analysis

Reachability Analysis (compare traditional DFA)

cfg::node::*is_reachable()* : Boolean

```
return self.incoming.source.is_reachable()->includes(true)
```

cfg::node::*is_live()* : Boolean

```
return self.outgoing.target.is_live()->includes(true)
```

Strongly Connected Components (SCC)

cfg::node::*all_predecessors()* : Set(tokenflow::Token)

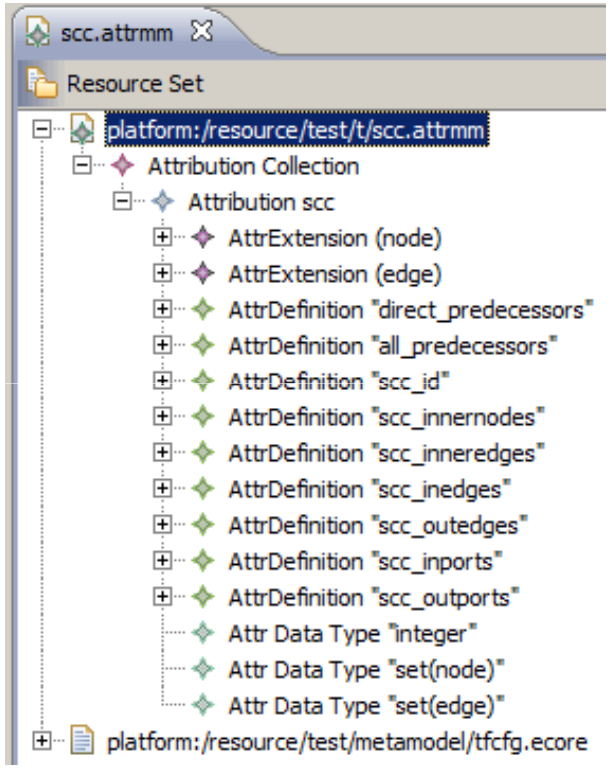
```
var directPred : Set(node) := self.direct_predecessors();
var incomingPred : Set(node) :=
    self.direct_predecessors().all_predecessors()->asSet();
return directPred->union(incomingPred)
```

cfg::edge::*scc_id()*: Integer

```
var self_pred : Set(node) := self.all_predecessors()->including(self);
var all_pred : Set(node) := self.direct_predecessors().all_predecessors()->asSet();

if (all_pred=self_pred) then return self_pred->hashCode() else return 0 endif;
```

cfg::edge::*scc_inedges()*: Set(cfg::edge)

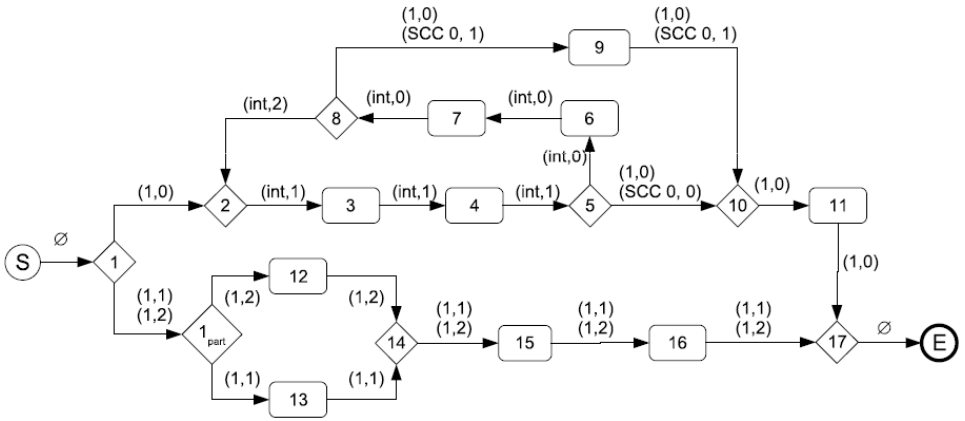
```
if (self.source = null or self.target = null) then return Set{} endif;

var inConnectors : Set(edge) := Set{};
var inSCC : Boolean := self.source.scc_id() != 0 and self.target.scc_id() != 0;

if (inSCC) then self.source.incoming->forEach(incomingEdge) {
    if (incomingEdge.source.scc_id() = self.source.scc_id())
    then inConnectors := inConnectors->union(incomingEdge.scc_inedges())->asSet()
    else inConnectors := inConnectors->union(Set{incomingEdge})->asSet() endif
} endif;

return inConnectors-self.scc_inneredges()
```



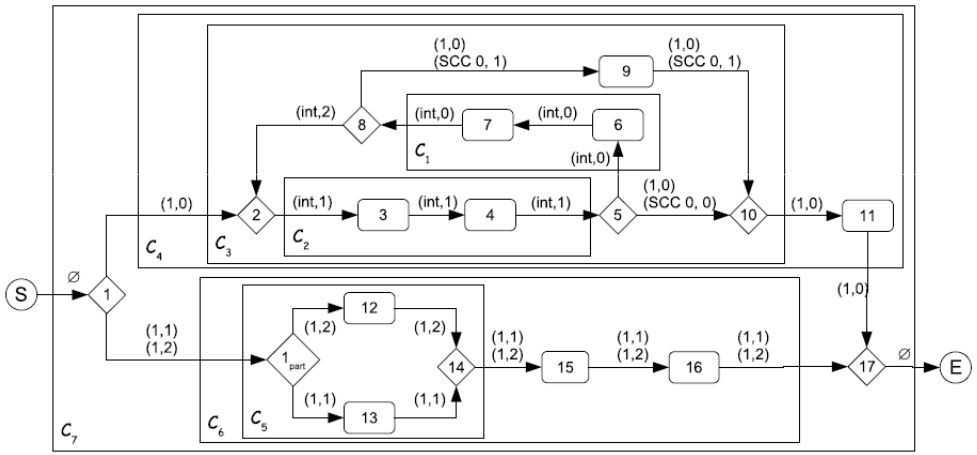
Token-flow Analysis [1]

- Extract **Single-Entry-Single-Exit** (SESE) components
- Required for validation, transformation, …

1. Create and Merge Tokens



2. Identify components with identical token sets



[1] Mathias Götz, Stephan Roser, Florian Lautenbacher, and Bernhard Bauer. *Token Analysis of Graph-Oriented Process Models*. 13th IEEE International EDOC Conference,September 2009.

Use Case

1. cfg::edge::*tokens_initial*() : Set(tokenflow::Token)

```
if (self.source = null) then return Set{} endif;

var tokens : Set(tokenflow::Token) := Set{};

self.source.incoming->forEach(incomingEdge)
{
    tokens := tokens->union(incomingEdge.tokens_initial())
};

var sccID : Integer := self.scc_id();
var sourceOutgoing : Integer = self.source.outgoing->size();
if (sccID=0 and sourceOutgoing>1)
then
{
  va
  ne

  va
  if
  th
  {

  }
  el
  {

  }
  en

  to
}
endi

retu
```

2. cfg::edge::*tokens*() : Set(tokenflow::Token)

```
var tokens : Set(tokenflow::Token) := Set{}->union(self.tokens_initial());
var originMap : Dict(String, Set(tokenflow::Token)) := Dict{};

self.tokens_initial()->forEach(token)
{
  var entry : Set(tokenflow::Token) := originMap->get(token.splitNode);

  if (entry = null) then entry := Set{} endif;

  entry := entry->including(token);
  originMap->put(token.splitNode, entry);

  if (entry->size()=token.outgoingCount)
  then
    tokens := tokens-entry
  endif;
};

return tokens;
```

3. cfg::edge::*tokenstack*() : JavaObject

Java-based DFA rule

Result (tokenstack at end node)

```
<?xml version="1.0" encoding="UTF-8"?>
<component>
    <type> ACYCLIC </type>
    <tokenset> [] </tokenset>
    <inputedge> S->1 </inputedge>
    <outputedge> 17->E </outputedge>
    <inneredges>
        <edge> 11->17 </edge>
        <edge> 1->2 </edge>
        <edge> 16->17 </edge>
        <edge> 1->1part </edge>
    </inneredges>
    <trivialnodes>
        <node> 17 </node>
        <node> 1 </node>
    </trivialnodes>
    <subcomponents>
        <component>
            <type> SEQUENCE </type>
            <tokenset> [(1,2,2)] </tokenset>
            <inputedge> 1->2 </inputedge>
            <outputedge> 11->17 </outputedge>
            <inneredges>
                <edge> 10->11 </edge>
            </inneredges>
            <trivialnodes>
                <node> 11 </node>
            </trivialnodes>
                <subcomponents>
                <component>
                    <type> CYCLIC </type>
                    <tokenset> [(1,2,2)] </tokenset>
                    <inputedge> 1->2 </inputedge>
                    <outputedge> 10->11 </outputedge>
                    <inneredges>
                        <edge> 5->10 </edge>
                        <edge> 9->10 </edge>
                        <edge> 8->9 </edge>
                    </inneredges>
                    <trivialnodes>
                        <node> 10 </node>
                        <node> 9 </node>
                    </trivialnodes>
                    <allinneredges>
                        <edge> 5->10 </edge>
                        <edge> 9->10 </edge>
...
```

Outlook

- Concept
  - Improve evaluation algorithm
  - Practical and theoretical evaluation of performance
- Implementation (MAF)
  - Provide graphical IDE
  - Run as server
- Additional Use Cases
  - Compute model metrics / check modeling guidelines
  - „Traditional DFA“, e.g. on model of Java code
  - Validation of (UML) models

Thank you for your attention!

Questions?