

The OSATE Slicer: Graph-Based Reachability for Architectural Models

JULY 20, 2023

Sam Procter



Document Markings

Copyright 2023 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM23-0721

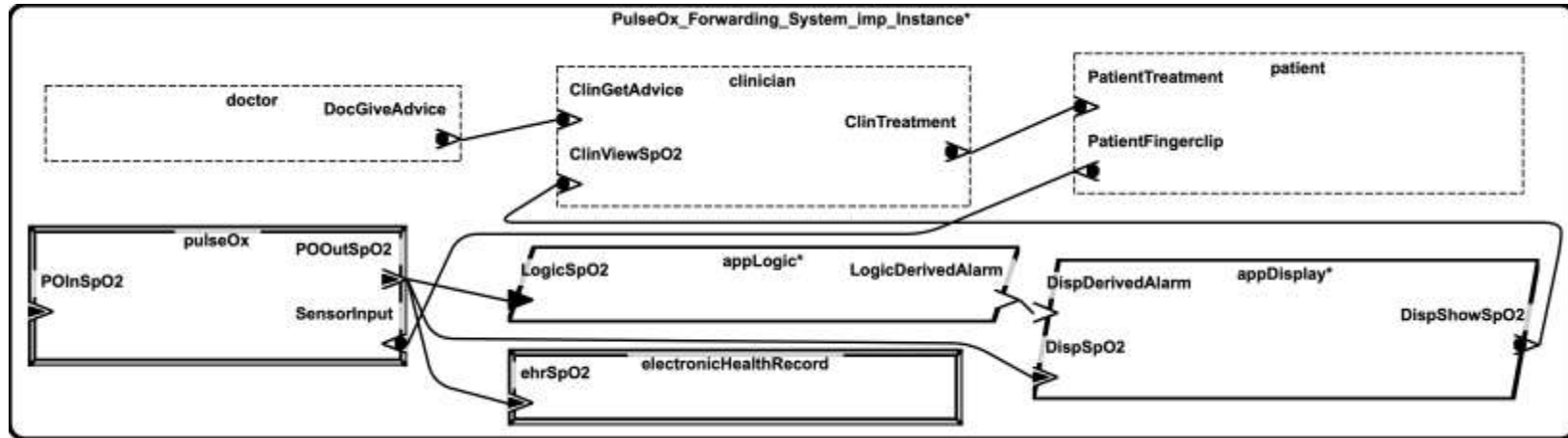
Agenda

- **Introduction & Background**
 - **Problem**
 - **Context**
 - **Solution**
- The OSATE Slicer
- Evaluation

Problem: Models are hard to comprehend

For both manual and automated analyses

- For humans: High cognitive burden – “unwieldy far quicker” than programs [1]
- For automated analyses: Traversal of model elements not easily converted to data- or control-flow ordering

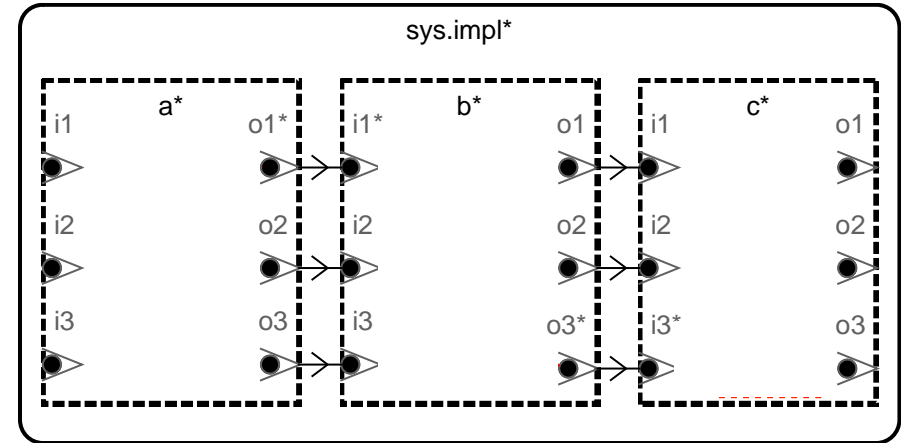
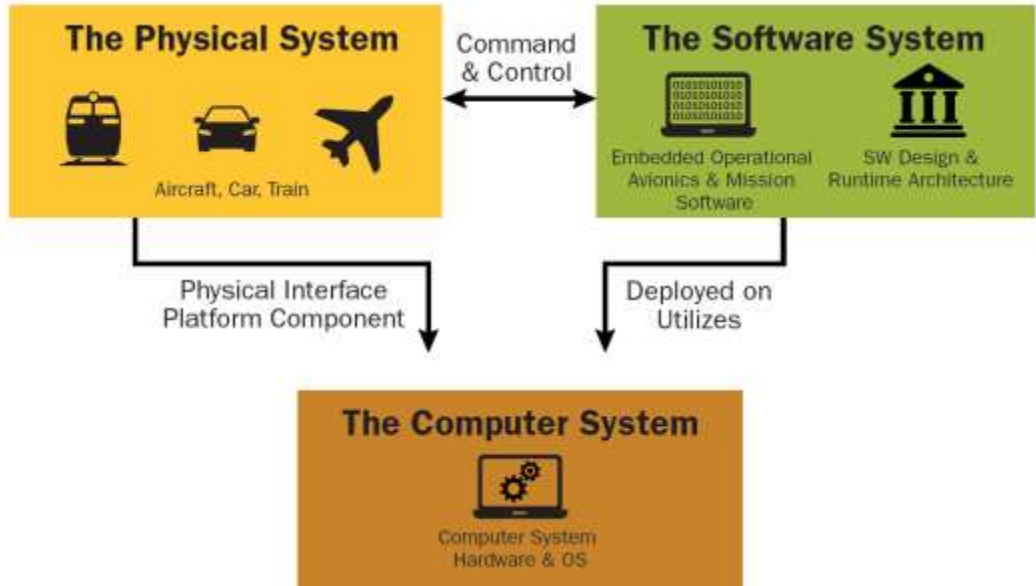


[1] “State-Based ModelSlicing: A Survey.” K. Androutsopoulos, D. Clark, M. Harman, J. Krinke, L. Tratt. *ACM Computing Surveys*, 2013.

Context (1): Architecture Analysis & Design Language

AADL

AADL focuses on interaction between the three elements of a software-reliant mission and safety-critical systems



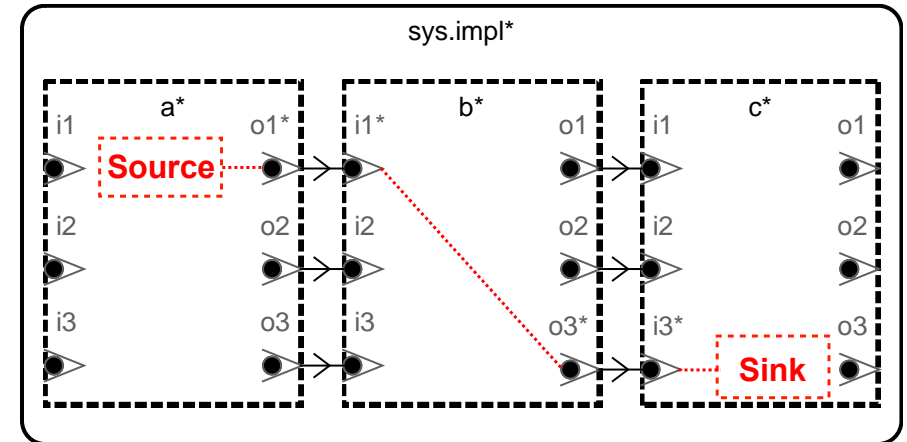
Context (2): AADL Error Modeling Annex

EMV2

Extension of core AADL for modeling off-nominal behavior

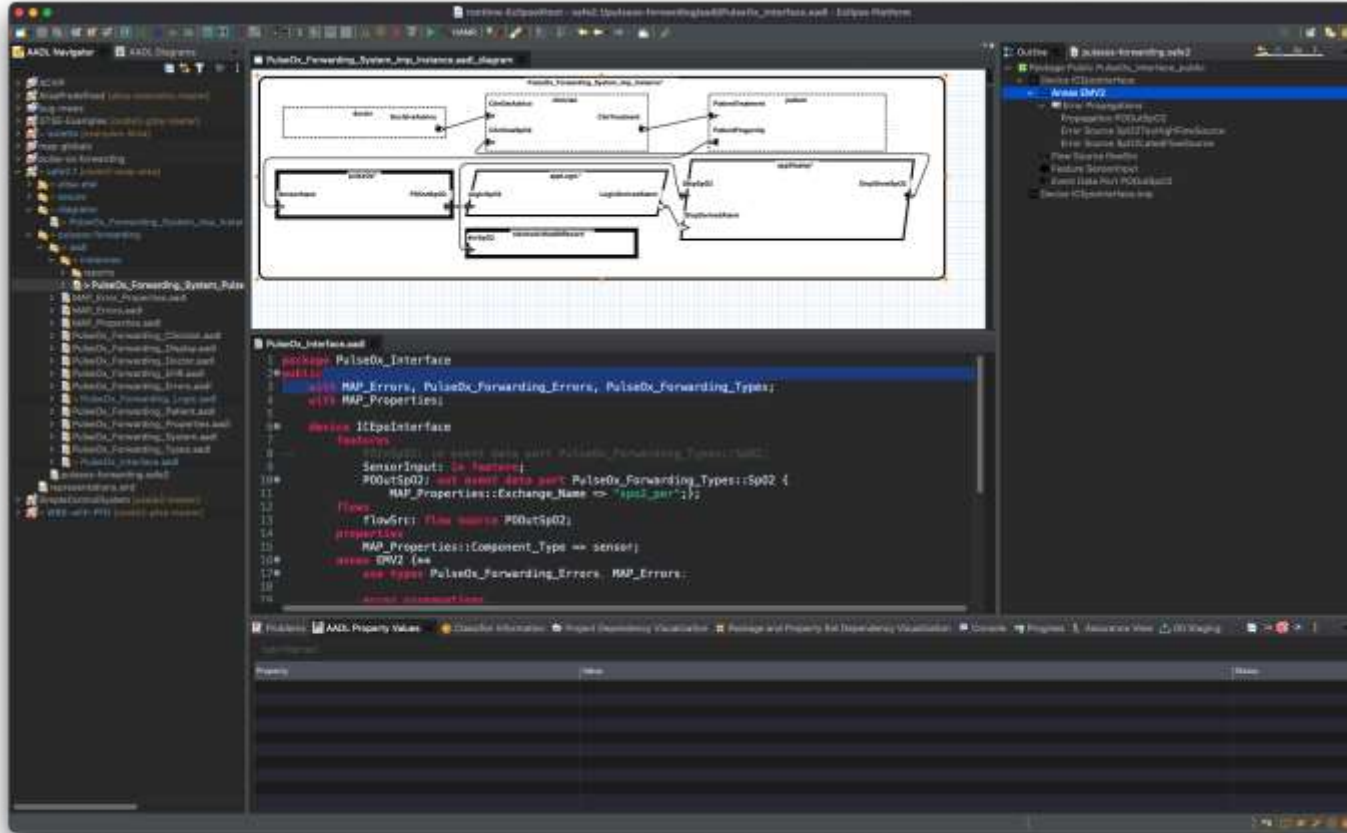
We rely heavily on:

- Error Types
- Propagation Paths
- Error Flows
 - Creation
 - Propagation
 - Transformation
 - Consumption



Context (3): Open Source AADL Tool Environment

OSATE



Context (4): Program and Model Slicing

1. Weiser's Original Concept: Reduced, but still executable, version of program [1]
2. Ahmadi's Slicer: Reduced model (UML-RT) preserving structural and behavioral aspects [2]
3. SafeSlice: Requirement traceability across multiple levels of safety-critical system models (SysML) [3]
4. Kompren: Combines metamodel and conformant model to generate a slicer [4]
5. Awas: OSATE plugin for calculating reachability queries across AADL models [5]

[1] "Program Slicing." Mark Weiser. *IEEE Transactions on Software Engineering*, 1984.

[2] "Slicing UML-based Models of Real-time Embedded Systems." Reza Ahmadi, Ernesto Posse, Juergen Dingel. *MODELS*, 2018.

[3] "Traceability and SysML Design Slices to Support Safety Inspections: A Controlled Experiment." Lionel Briand, Davide Falessi, Shiva Nejati, Mehrdad Sabetzadeh, Tao Yue. *ACM Trans. on SW Eng and Methodology*, 2014.

[4] "Kompren: Modeling and Generating Model Slicers." Arnaud Blouin, Benoît Combemale, Benoit Baudry, Olivier Beaudoux. *Software & Systems Modeling*, 2012.

[5] "Awas: AADL Information Flow and Error Propagation Analysis Framework." Hariharan Thiagarajan, John Hatcliff, Robby. *Innovations in Systems and Software Engineering*, 2022.

Solution: The *OSATE Slicer*

Goal: Reachability calculations built into OSATE that are...

- Usable
 - Data and control flow are used in a number of analyses
 - ... so the Slicer should be too
- Maintainable
 - Align with existing OSATE design principles, tooling, and infrastructure
- Performant
 - Reduced installation complexity and execution time

Agenda

- Introduction & Background
- **The OSATE Slicer**
 - Representation
 - Generation
 - Queries
- Evaluation

Graph Representation

Slicer generates and queries two graphs

- Nominal (Core AADL)
- Off-Nominal (Core AADL + EMV2)

Nominal:

1. $\mathcal{G}_N = (V_N, \rightarrow_{e_N})$
 1. $V_N = F \cup A_{used}$
 2. $\rightarrow_{e_N} \subseteq V_N \times V_N$

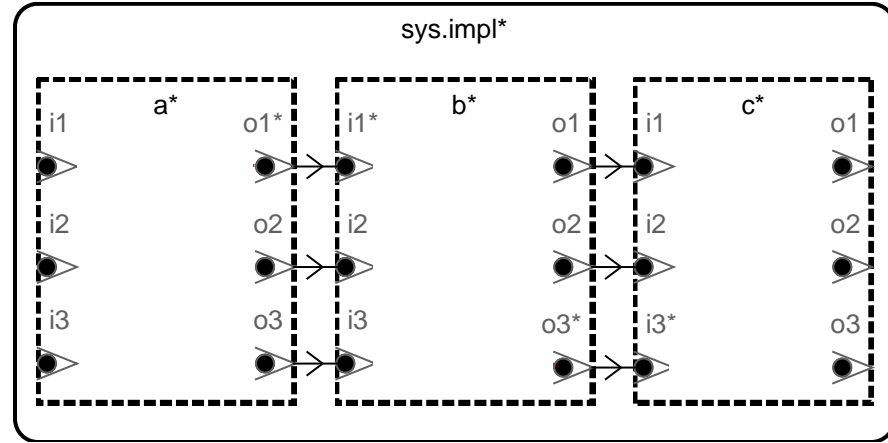
Off-Nominal:

2. $\mathcal{G}_O = (V_O, \rightarrow_{e_O})$
 1. $V_O = L \times T$
 2. $L = P_F \times P_B \times P_A \times P_P \times R_{Src} \times R_{Snk}$
 3. $\rightarrow_{e_O} \subseteq V_O \times V_O$

Graph Generation

Nominal

(a) Nominal Reachability



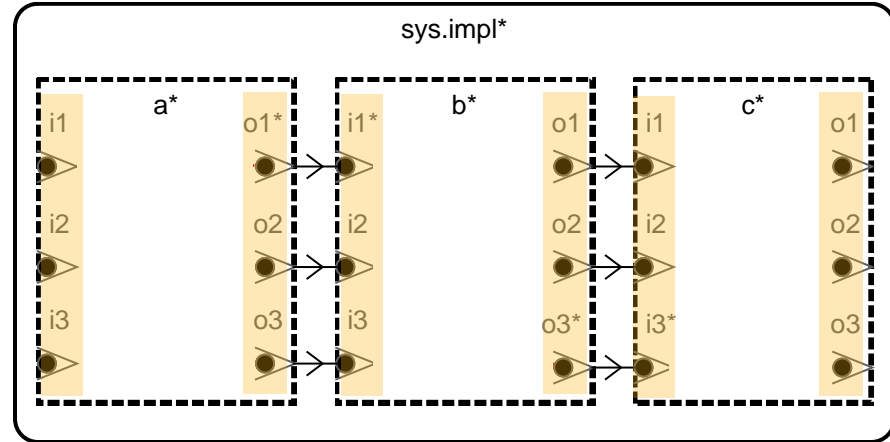
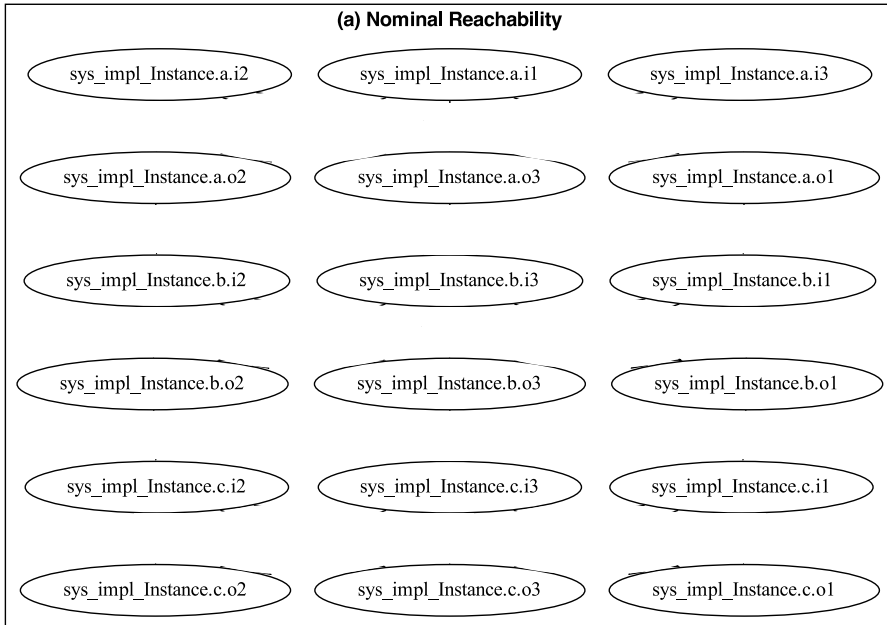
```

1: function CONSTRUCTNOMINAL(m)
2:   ED := ∅ ▷ Components with explicit decompositions
3:   GN = (VN, →eN) := (∅, ∅) ▷ Initialize GN
4:   for feat ∈ m do ▷ feat is a feature
5:     VN := VN ∪ VERT(feat)
6:   for conn ∈ m do ▷ conn is a connection
7:     VN := VN ∪ VERT(connSrc) ∪ VERT(connDst)
8:     →eN := →eN ∪ EDGE(connSrc, connDst)
9:     ED := ED ∪ CONTAINER(conn)
10:  for flow ∈ m do ▷ flow is a end to end flow
11:    →eN := →eN ∪ EDGE(flowSrc, flowDst)
12:    ED := ED ∪ CONTAINER(flow)
13:  for comp ∈ (m \ ED) do ▷ comp is a component
14:    for featin ∈ comp do
15:      for featout ∈ comp do
16:        →eN := →eN ∪ EDGE(featin, featout)
  
```

Graph Generation

Nominal

(a) Nominal Reachability



```

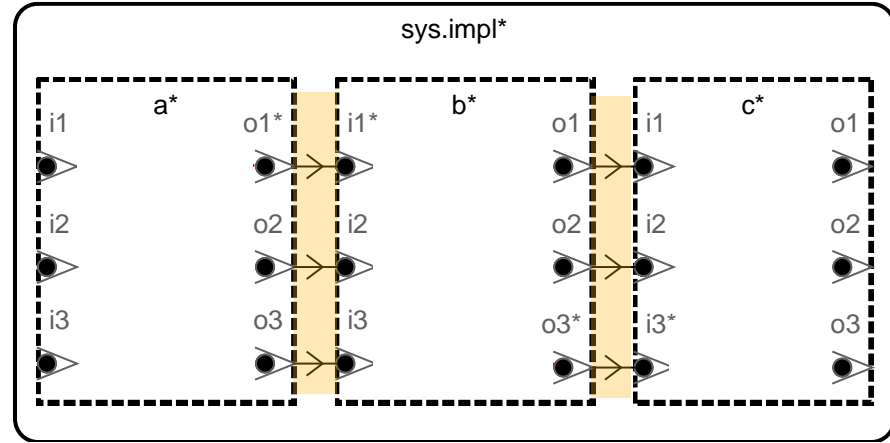
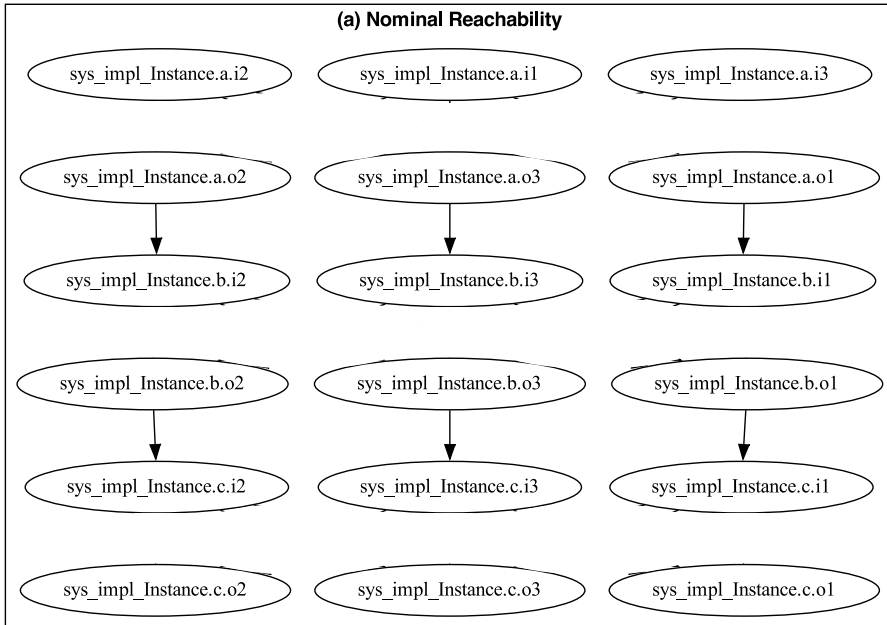
1: function CONSTRUCTNOMINAL(m)
2:    $ED := \emptyset$   $\triangleright$  Components with explicit decompositions
3:    $\mathcal{G}_N = (V_N, \rightarrow_{e_N}) := (\emptyset, \emptyset)$   $\triangleright$  Initialize  $\mathcal{G}_N$ 
4:   for feat  $\in$  m do  $\triangleright$  feat is a feature
5:      $V_N := V_N \cup \text{VERT}(\text{feat})$ 
6:   for conn  $\in$  m do  $\triangleright$  conn is a connection
7:      $V_N := V_N \cup \text{VERT}(\text{conn}_{Src}) \cup \text{VERT}(\text{conn}_{Dst})$ 
8:      $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{conn}_{Src}, \text{conn}_{Dst})$ 
9:      $ED := ED \cup \text{CONTAINER}(\text{conn})$ 
10:  for flow  $\in$  m do  $\triangleright$  flow is a end to end flow
11:     $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{flow}_{Src}, \text{flow}_{Dst})$ 
12:     $ED := ED \cup \text{CONTAINER}(\text{flow})$ 
13:  for comp  $\in$  (m  $\setminus$   $ED$ ) do  $\triangleright$  comp is a component
14:    for featin  $\in$  comp do
15:      for featout  $\in$  comp do
16:         $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{feat}_{in}, \text{feat}_{out})$ 

```

Graph Generation

Nominal

(a) Nominal Reachability



```

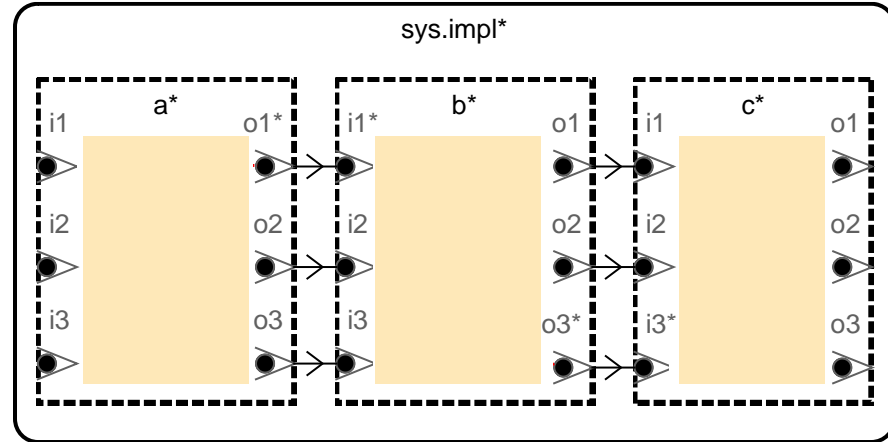
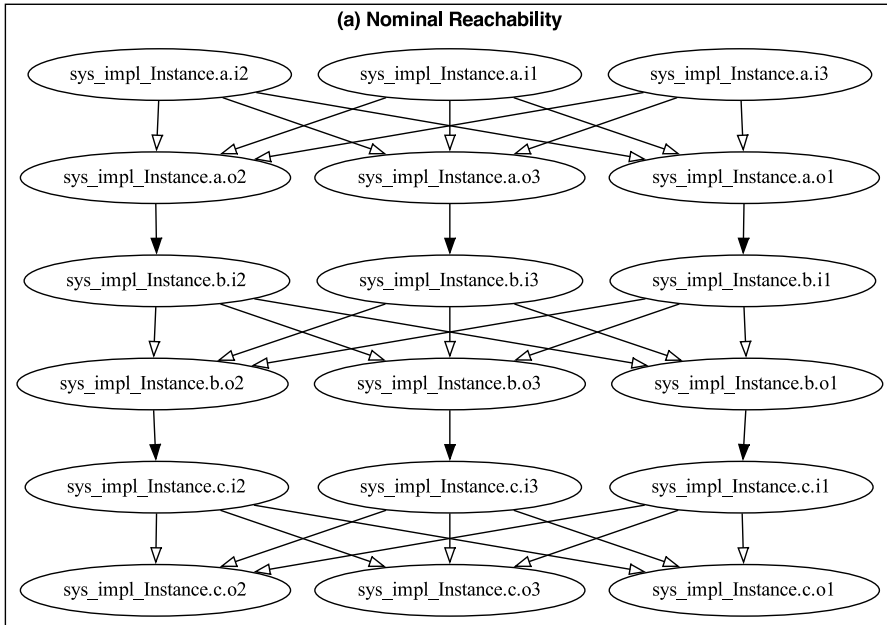
1: function CONSTRUCTNOMINAL(m)
2:    $ED := \emptyset$   $\triangleright$  Components with explicit decompositions
3:    $\mathcal{G}_N = (V_N, \rightarrow_{e_N}) := (\emptyset, \emptyset)$   $\triangleright$  Initialize  $\mathcal{G}_N$ 
4:   for feat  $\in$  m do  $\triangleright$  feat is a feature
5:      $V_N := V_N \cup \text{VERT}(\text{feat})$ 
6:     for conn  $\in$  m do  $\triangleright$  conn is a connection
7:        $V_N := V_N \cup \text{VERT}(\text{conn}_{Src}) \cup \text{VERT}(\text{conn}_{Dst})$ 
8:        $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{conn}_{Src}, \text{conn}_{Dst})$ 
9:        $ED := ED \cup \text{CONTAINER}(\text{conn})$ 
10:    for flow  $\in$  m do  $\triangleright$  flow is a end to end flow
11:       $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{flow}_{Src}, \text{flow}_{Dst})$ 
12:       $ED := ED \cup \text{CONTAINER}(\text{flow})$ 
13:    for comp  $\in$  (m  $\setminus$   $ED$ ) do  $\triangleright$  comp is a component
14:      for featin  $\in$  comp do
15:        for featout  $\in$  comp do
16:           $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{feat}_{in}, \text{feat}_{out})$ 

```

Graph Generation

Nominal

(a) Nominal Reachability



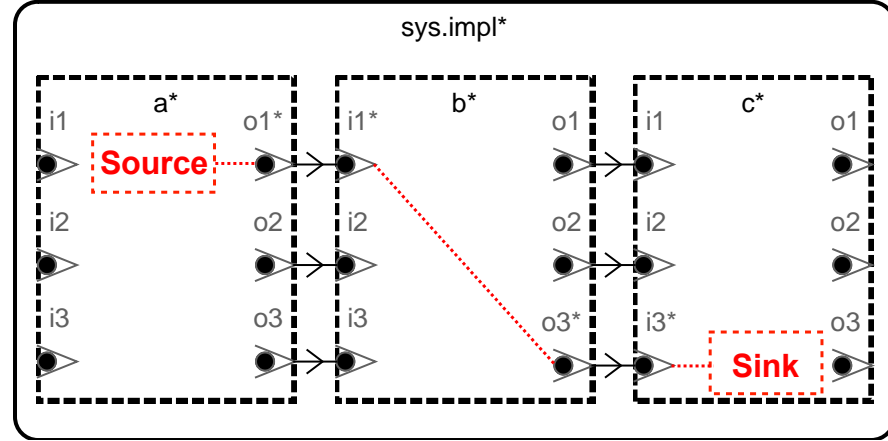
```

1: function CONSTRUCTNOMINAL(m)
2:    $ED := \emptyset$   $\triangleright$  Components with explicit decompositions
3:    $\mathcal{G}_N = (V_N, \rightarrow_{e_N}) := (\emptyset, \emptyset)$   $\triangleright$  Initialize  $\mathcal{G}_N$ 
4:   for feat  $\in$  m do  $\triangleright$  feat is a feature
5:      $V_N := V_N \cup \text{VERT}(\text{feat})$ 
6:   for conn  $\in$  m do  $\triangleright$  conn is a connection
7:      $V_N := V_N \cup \text{VERT}(\text{conn}_{Src}) \cup \text{VERT}(\text{conn}_{Dst})$ 
8:      $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{conn}_{Src}, \text{conn}_{Dst})$ 
9:      $ED := ED \cup \text{CONTAINER}(\text{conn})$ 
10:  for flow  $\in$  m do  $\triangleright$  flow is a end to end flow
11:     $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{flow}_{Src}, \text{flow}_{Dst})$ 
12:     $ED := ED \cup \text{CONTAINER}(\text{flow})$ 
13:  for comp  $\in$  (m  $\setminus$   $ED$ ) do  $\triangleright$  comp is a component
14:    for featin  $\in$  comp do
15:      for featout  $\in$  comp do
16:         $\rightarrow_{e_N} := \rightarrow_{e_N} \cup \text{EDGE}(\text{feat}_{in}, \text{feat}_{out})$ 

```

Graph Generation

Off-Nominal



(b) Off-Nominal Reachability

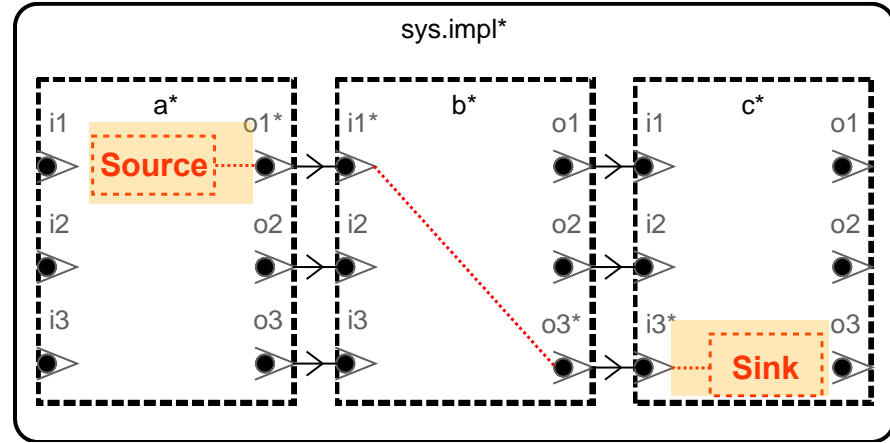
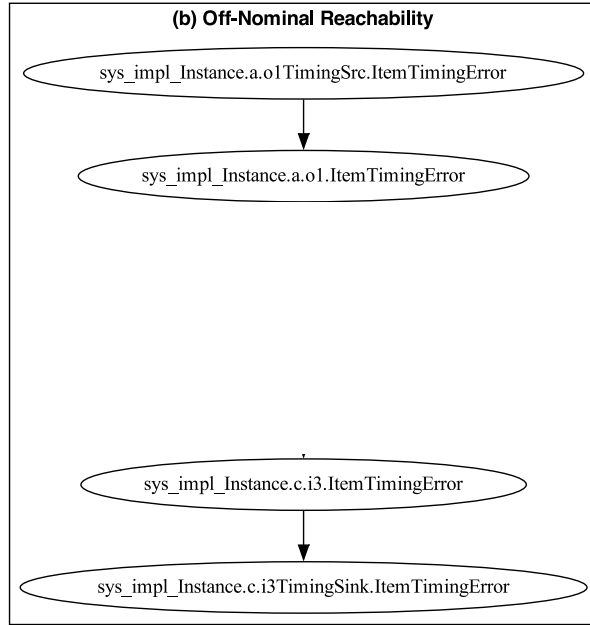
```

1: function CONSTRUCTOFFNOMINAL(m)
2:   PP := ∅ ▷ Set of possible propagations
3:   GO = (VO, →eO) := (∅, ∅) ▷ Initialize GO
4:   for (src, Psrc, Tsrc) ∈ mRsrc do ▷ Rsrc is a error source
5:     VO := VO ∪ VERT(src, Tsrc) ∪ VERT(Psrc, Tsrc)
6:     →eO := →eO ∪ EDGE(Vsrc, VPsrc)
7:   for (snk, Psnk, Tsnk) ∈ mRsnk do ▷ Rsnk is a error sink
8:     VO := VO ∪ VERT(snk, Tsnk) ∪ VERT(Psnk, Tsnk)
9:     →eO := →eO ∪ EDGE(VPsnk, Vsnk)
10:  for ErrPath ∈ m do ▷ ErrPath is a error path
11:    VO := VO ∪ VERT(ErrPathdst, Tdst)
12:    for (src, Tsrc) ∈ ErrPath do
13:      VO := VO ∪ VERT(ErrPathsrc, Tsrc)
14:      →eO := →eO ∪ EDGE(VErrPathsrc, VErrPathsnk)
15:  for PPath ∈ m do ▷ PPath is a propagation path
16:    PP := PP ∪ PPROP(PPathsrc, PPathdst)

```

Graph Generation

Off-Nominal

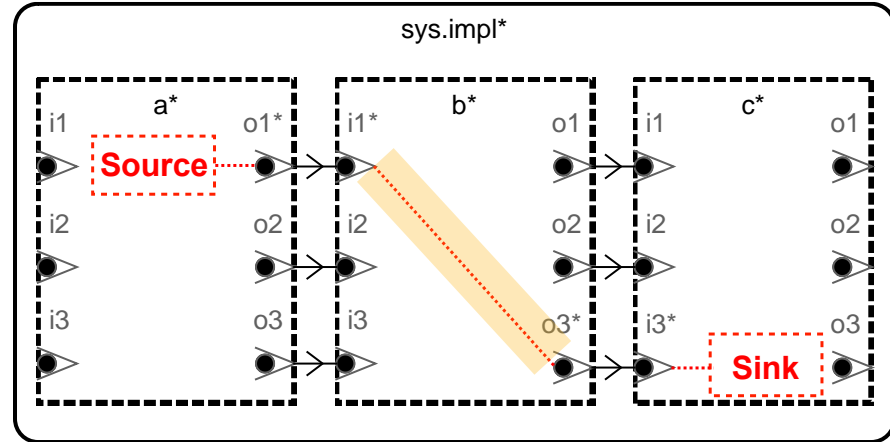
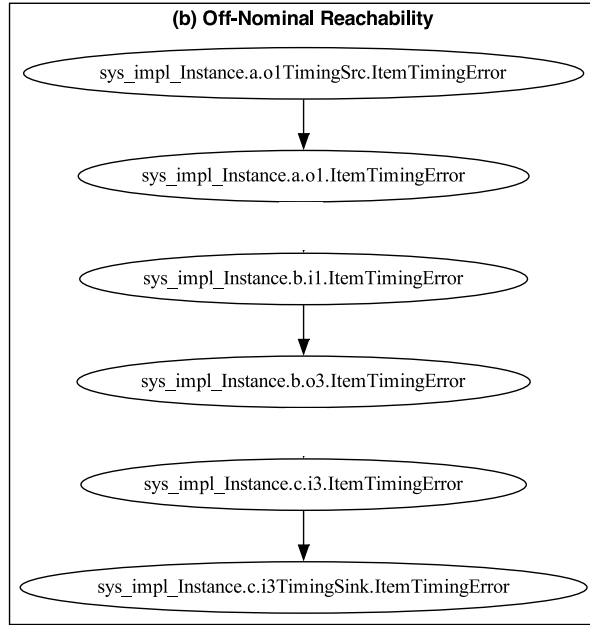


```

1: function CONSTRUCTOFFNOMINAL(m)
2:   PP := ∅           ▷ Set of possible propagations
3:   GO = (VO, →eO) := (∅, ∅)           ▷ Initialize GO
4:   for (src, Psrc, Tsrc) ∈ mRsrc do ▷ Rsrc is a error source
5:     VO := VO ∪ VERT(src, Tsrc) ∪ VERT(Psrc, Tsrc)
6:     →eO := →eO ∪ EDGE(Vsrc, VPsrc)
7:   for (snk, Psnk, Tsnk) ∈ mRsnk do ▷ Rsnk is a error sink
8:     VO := VO ∪ VERT(snk, Tsnk) ∪ VERT(Psnk, Tsnk)
9:     →eO := →eO ∪ EDGE(VPsnk, Vsnk)
10:  for ErrPath ∈ m do           ▷ ErrPath is a error path
11:    VO := VO ∪ VERT(ErrPathdst, Tdst)
12:    for (src, Tsrc) ∈ ErrPath do
13:      VO := VO ∪ VERT(ErrPathsrc, Tsrc)
14:      →eO := →eO ∪ EDGE(VErrPathsrc, VErrPathsnk)
15:  for PPath ∈ m do           ▷ PPath is a propagation path
16:    PP := PP ∪ PPROP(PPathsrc, PPathdst)
  
```

Graph Generation

Off-Nominal

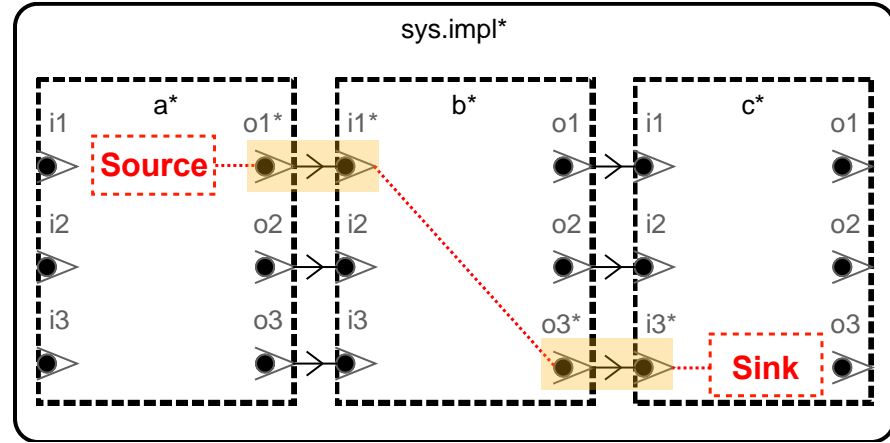
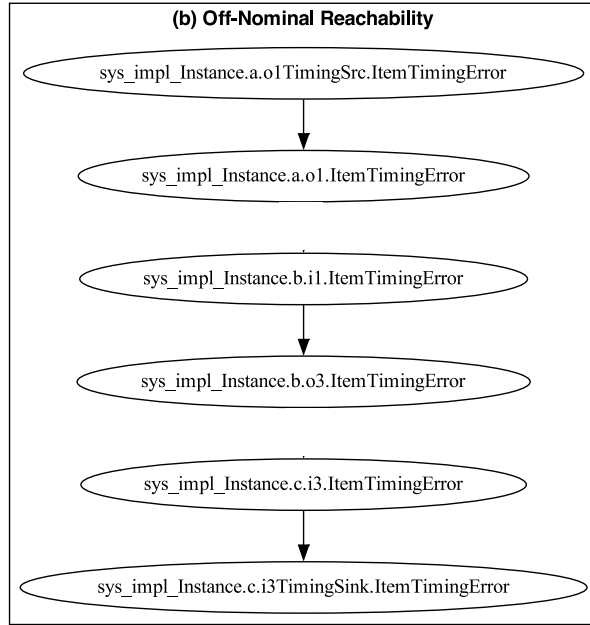


```

1: function CONSTRUCTOFFNOMINAL(m)
2:   PP := ∅           ▷ Set of possible propagations
3:    $\mathcal{G}_O = (V_O, \rightarrow_{e_O}) := (\emptyset, \emptyset)$    ▷ Initialize  $\mathcal{G}_O$ 
4:   for (src, Psrc, Tsrc) ∈ mRsrc do ▷ Rsrc is a error source
5:     VO := VO ∪ VERT(src, Tsrc) ∪ VERT(Psrc, Tsrc)
6:      $\rightarrow_{e_O}$  :=  $\rightarrow_{e_O}$  ∪ EDGE(Vsrc, VPsrc)
7:   for (snk, Psnk, Tsnk) ∈ mRsnk do ▷ Rsnk is a error sink
8:     VO := VO ∪ VERT(snk, Tsnk) ∪ VERT(Psnk, Tsnk)
9:      $\rightarrow_{e_O}$  :=  $\rightarrow_{e_O}$  ∪ EDGE(VPsnk, Vsnk)
10:  for ErrPath ∈ m do ▷ ErrPath is a error path
11:    VO := VO ∪ VERT(ErrPathdst, Tdst)
12:    for (src, Tsrc) ∈ ErrPath do
13:      VO := VO ∪ VERT(ErrPathsrc, Tsrc)
14:       $\rightarrow_{e_O}$  :=  $\rightarrow_{e_O}$  ∪ EDGE(VErrPathsrc, VErrPathsnk)
15:  for PPath ∈ m do ▷ PPath is a propagation path
16:    PP := PP ∪ PPROP(PPathsrc, PPathdst)
  
```

Graph Generation

Off-Nominal

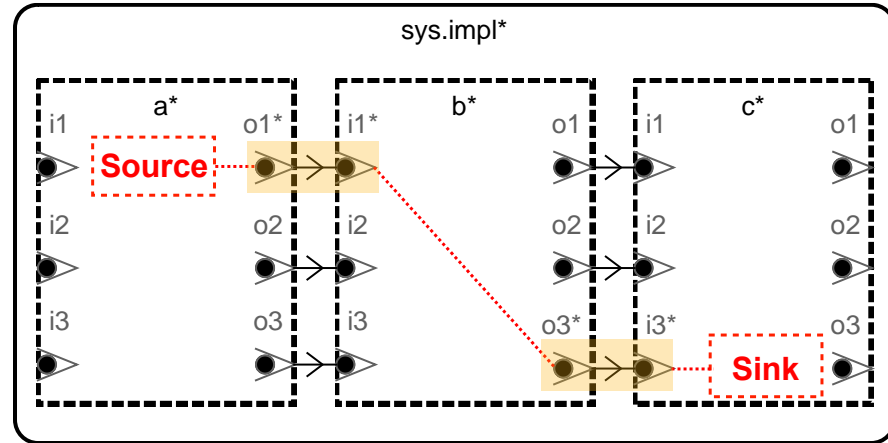
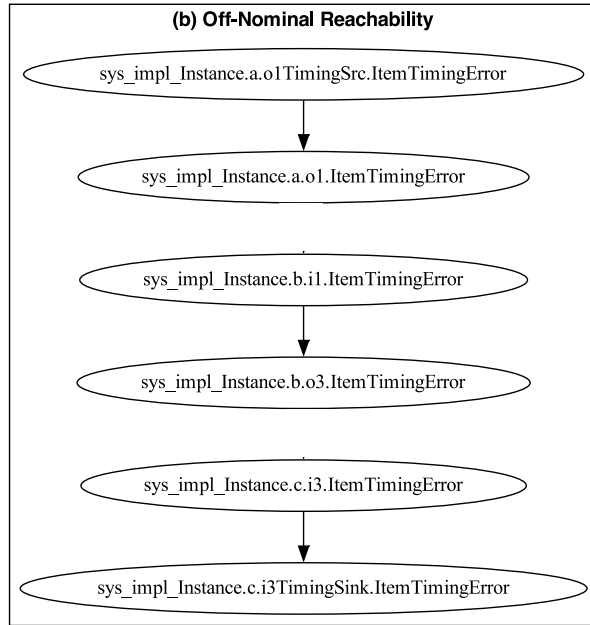


```

1: function CONSTRUCTOFFNOMINAL(m)
2:   PP := ∅           ▷ Set of possible propagations
3:    $\mathcal{G}_O = (V_O, \rightarrow_{e_O}) := (\emptyset, \emptyset)$    ▷ Initialize  $\mathcal{G}_O$ 
4:   for (src, Psrc, Tsrc) ∈ mRsrc do ▷ Rsrc is a error source
5:      $V_O := V_O \cup \text{VERT}(\text{src}, T_{\text{src}}) \cup \text{VERT}(P_{\text{src}}, T_{\text{src}})$ 
6:      $\rightarrow_{e_O} := \rightarrow_{e_O} \cup \text{EDGE}(V_{\text{src}}, V_{P_{\text{src}}})$ 
7:   for (snk, Psnk, Tsnk) ∈ mRsnk do ▷ Rsnk is a error sink
8:      $V_O := V_O \cup \text{VERT}(\text{snk}, T_{\text{snk}}) \cup \text{VERT}(P_{\text{snk}}, T_{\text{snk}})$ 
9:      $\rightarrow_{e_O} := \rightarrow_{e_O} \cup \text{EDGE}(V_{P_{\text{snk}}}, V_{\text{snk}})$ 
10:  for ErrPath ∈ m do ▷ ErrPath is a error path
11:     $V_O := V_O \cup \text{VERT}(\text{ErrPath}_{\text{dst}}, T_{\text{dst}})$ 
12:    for (src, Tsrc) ∈ ErrPath do
13:       $V_O := V_O \cup \text{VERT}(\text{ErrPath}_{\text{src}}, T_{\text{src}})$ 
14:       $\rightarrow_{e_O} := \rightarrow_{e_O} \cup \text{EDGE}(V_{\text{ErrPath}_{\text{src}}}, V_{\text{ErrPath}_{\text{snk}}})$ 
15:  for PPath ∈ m do ▷ PPath is a propagation path
16:     $PP := PP \cup \text{PPROP}(PP_{\text{src}}, PP_{\text{dst}})$ 
  
```

Graph Generation

Off-Nominal – Fixpoint Calculation

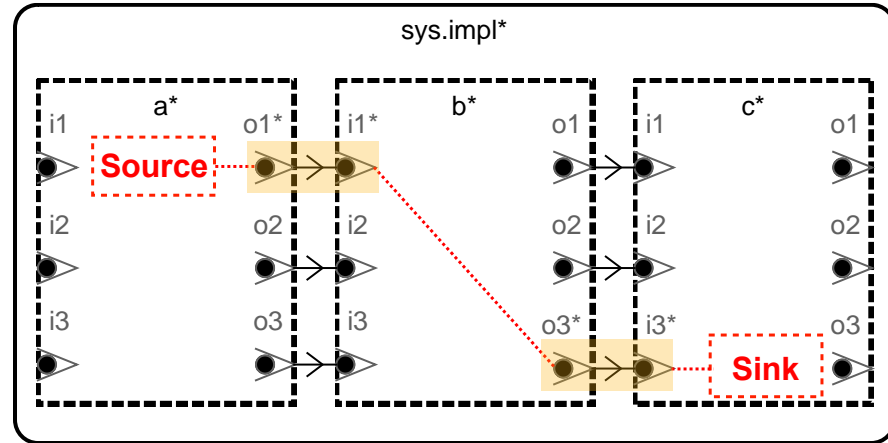
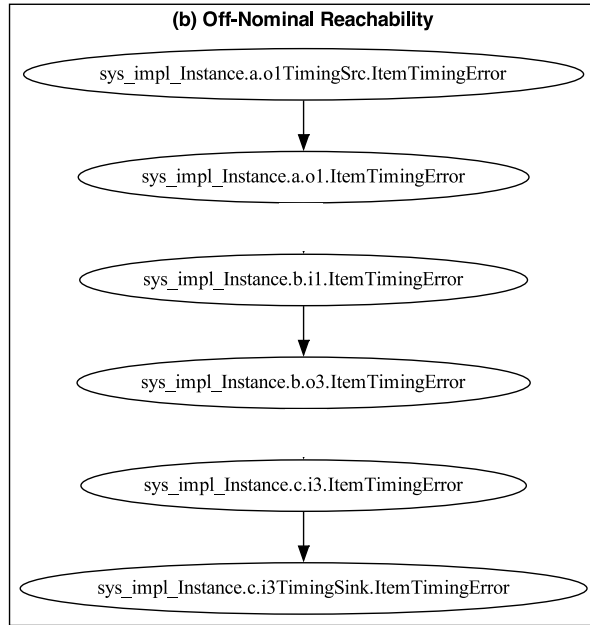


```

1: function CALCULATEFIXPOINT( $R_{Src}$ ,  $V_O$ ,  $\rightarrow_{e_O}$ ,  $PP$ )
2:   repeat
3:      $\rightarrow'_{e_O} := \rightarrow_{e_O}$ 
4:     for  $src \in R_{Src}$  do ▷ src is a error source
5:        $edges := V_{srcOut}$  ▷ Outgoing edges of  $V_{src}$ 
6:       while  $|edges| > 0$  do
7:          $CurrEdge := POP(edges)$ 
8:          $src := V_{CurrEdge_{Dst}}$  ▷ CurrEdge's dest.
9:         for  $OutEdge \in srcOut$  do
10:           $edges := edges \cup OutEdge$ 
11:        for  $Prop \in \{PP | Src = src\}$  do
12:           $tgt := Prop_{Dst}$ 
13:           $NewEdge := EDGE(src, tgt)$ 
14:          if  $NewEdge \notin \rightarrow_{e_O}$  then
15:             $edges := edges \cup NewEdge$ 
16:             $\rightarrow_{e_O} := \rightarrow_{e_O} \cup NewEdge$ 
17:   until  $\rightarrow'_{e_O} = \rightarrow_{e_O}$  ▷ Halt when edge set is unmodified
  
```

Graph Generation

Off-Nominal – Fixpoint Calculation

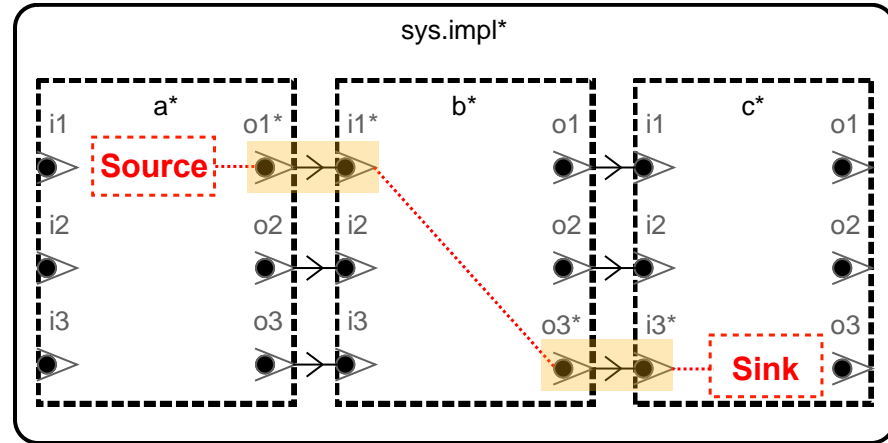
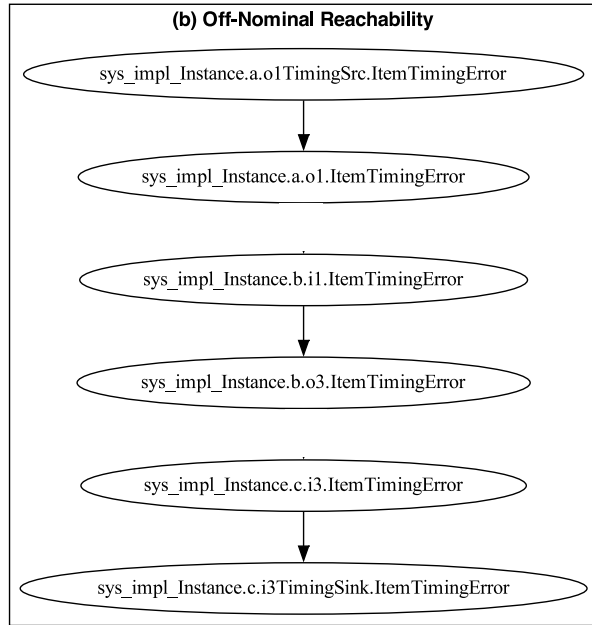


```

1: function CALCULATEFIXPOINT( $R_{Src}$ ,  $V_{O}$ ,  $\rightarrow_{e_O}$ ,  $PP$ )
2:   repeat
3:      $\rightarrow'_{e_O} := \rightarrow_{e_O}$ 
4:     for  $src \in R_{Src}$  do  $\triangleright src$  is a error source
5:       edges :=  $V_{srcOut}$   $\triangleright$  Outgoing edges of  $V_{src}$ 
6:       while |edges| > 0 do
7:         CurrEdge := POP(edges)
8:         src :=  $V_{CurrEdgeDst}$   $\triangleright$  CurrEdge's dest.
9:         for OutEdge  $\in srcOut$  do
10:          edges := edges  $\cup$  OutEdge
11:         for Prop  $\in \{PP | Src = src\}$  do
12:          tgt := PropDst
13:          NewEdge := EDGE(src, tgt)
14:          if NewEdge  $\notin \rightarrow_{e_O}$  then
15:            edges := edges  $\cup$  NewEdge
16:             $\rightarrow_{e_O} := \rightarrow_{e_O} \cup NewEdge$ 
17:   until  $\rightarrow'_{e_O} = \rightarrow_{e_O}$   $\triangleright$  Halt when edge set is unmodified
  
```

Graph Generation

Off-Nominal – Fixpoint Calculation

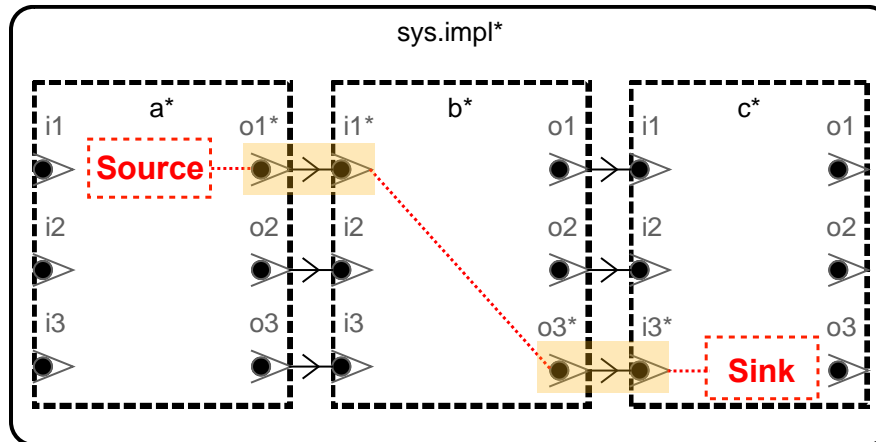
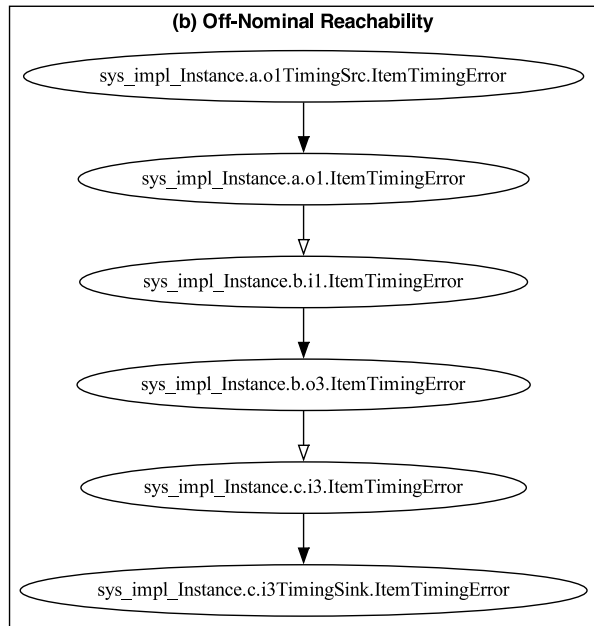


```

1: function CALCULATEFIXPOINT( $R_{Src}$ ,  $V_{O}$ ,  $\rightarrow_{e_O}$ ,  $PP$ )
2:   repeat
3:      $\rightarrow'_{e_O} := \rightarrow_{e_O}$ 
4:     for  $src \in R_{Src}$  do  $\triangleright src$  is a error source
5:       edges :=  $V_{srcOut}$   $\triangleright$  Outgoing edges of  $V_{src}$ 
6:       while |edges| > 0 do
7:         CurrEdge := POP(edges)
8:          $src := V_{CurrEdge_{Dst}}$   $\triangleright$  CurrEdge's dest.
9:         for OutEdge  $\in src_{Out}$  do
10:          edges := edges  $\cup$  OutEdge
11:         for Prop  $\in \{PP | Src = src\}$  do
12:          tgt := Prop $_{Dst}$ 
13:          NewEdge := EDGE(src, tgt)
14:          if NewEdge  $\notin \rightarrow_{e_O}$  then
15:            edges := edges  $\cup$  NewEdge
16:             $\rightarrow_{e_O} := \rightarrow_{e_O} \cup NewEdge$ 
17:   until  $\rightarrow'_{e_O} = \rightarrow_{e_O}$   $\triangleright$  Halt when edge set is unmodified
  
```

Graph Generation

Off-Nominal – Fixpoint Calculation

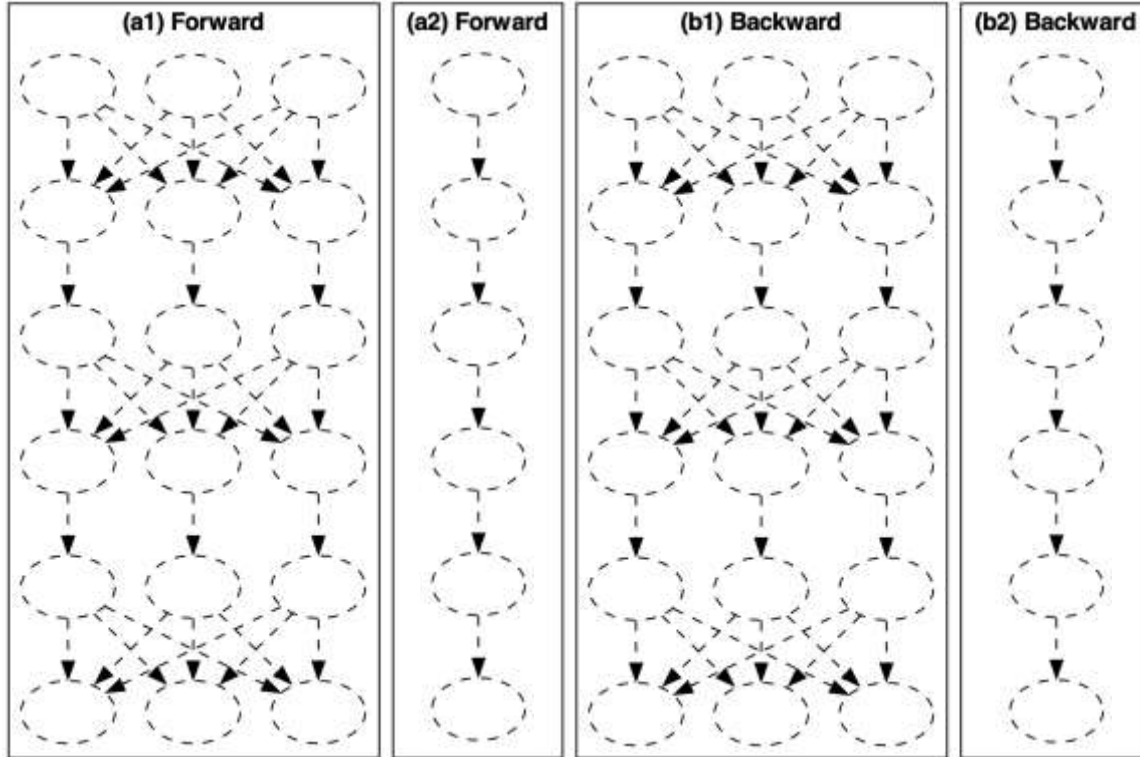


```

1: function CALCULATEFIXPOINT( $R_{Src}$ ,  $V_{O}$ ,  $\rightarrow_{e_O}$ ,  $PP$ )
2:   repeat
3:      $\rightarrow'_{e_O} := \rightarrow_{e_O}$ 
4:     for  $src \in R_{Src}$  do  $\triangleright src$  is a error source
5:       edges :=  $V_{srcOut}$   $\triangleright$  Outgoing edges of  $V_{src}$ 
6:       while |edges| > 0 do
7:         CurrEdge := POP(edges)
8:          $src := V_{CurrEdge_{Dst}}$   $\triangleright$  CurrEdge's dest.
9:         for OutEdge  $\in src_{Out}$  do
10:          edges := edges  $\cup$  OutEdge
11:         for Prop  $\in \{PP | Src = src\}$  do
12:          tgt := Prop $_{Dst}$ 
13:          NewEdge := EDGE(src, tgt)
14:          if NewEdge  $\notin \rightarrow_{e_O}$  then
15:            edges := edges  $\cup$  NewEdge
16:             $\rightarrow_{e_O} := \rightarrow_{e_O} \cup NewEdge$ 
17:   until  $\rightarrow'_{e_O} = \rightarrow_{e_O}$   $\triangleright$  Halt when edge set is unmodified
  
```

Graph Queries

Forward & Backward Reach



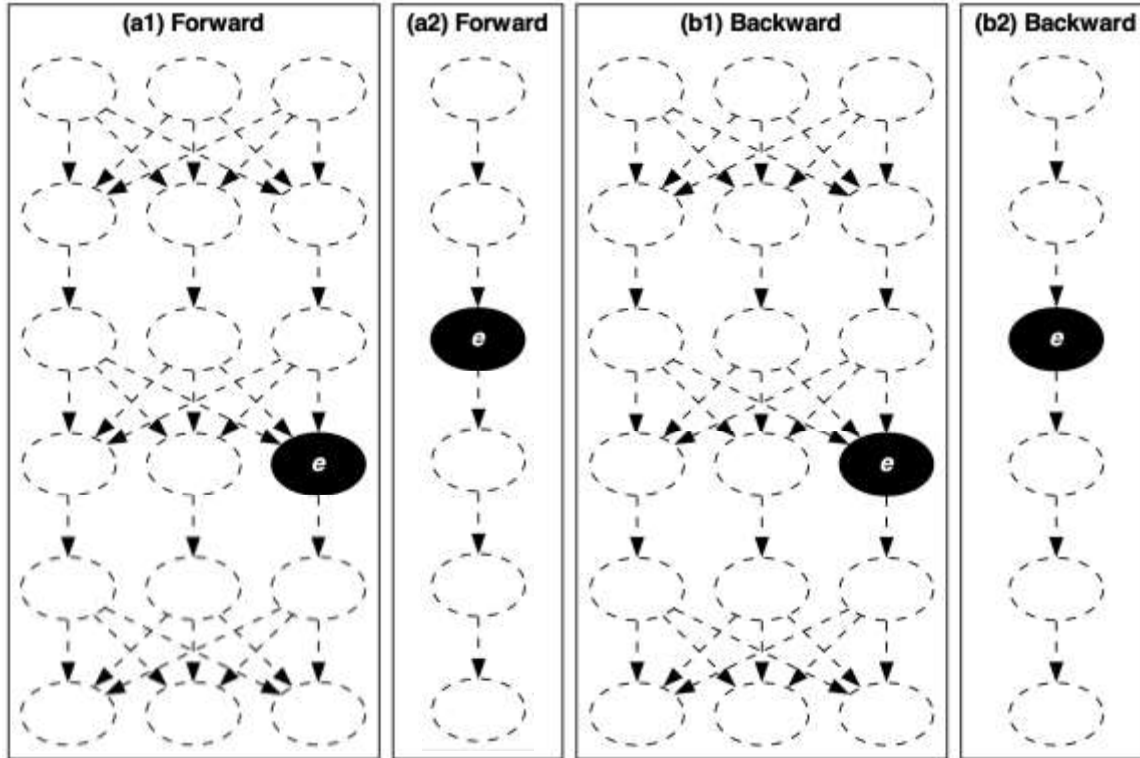
```

1: function REACH( $\mathcal{G}, v_{origin}$ )
2:    $\mathcal{G}_{sub} = (V_{sub}, \rightarrow_{e_{sub}}) := (\emptyset, \emptyset) = \text{SUBGRAPH}(\mathcal{G})$ 
3:    $V_{sub} := V_{sub} \cup v_{origin}$ 
4:    $v_{previous} := v_{origin}$ 
5:   for  $v_{current} \in \text{BFITER}(\mathcal{G}, v_{origin})$  do
6:      $V_{sub} := V_{sub} \cup v_{current}$ 
7:      $\rightarrow_{e_{sub}} := \rightarrow_{e_{sub}} \cup \text{EDGE}(v_{previous}, v_{current})$ 
8:      $v_{previous} := v_{current}$ 

```

Graph Queries

Forward & Backward Reach



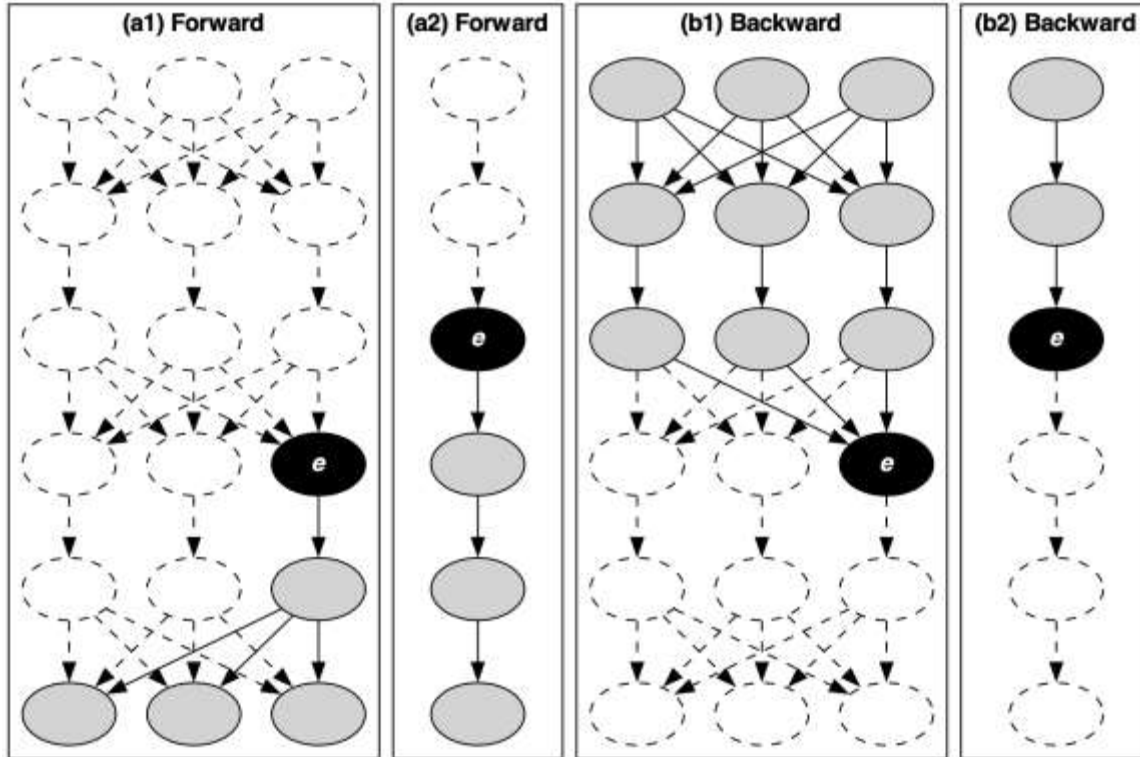
```

1: function REACH( $\mathcal{G}, v_{origin}$ )
2:    $\mathcal{G}_{sub} = (V_{sub}, \rightarrow_{e_{sub}}) := (\emptyset, \emptyset) = \text{SUBGRAPH}(\mathcal{G})$ 
3:    $V_{sub} := V_{sub} \cup v_{origin}$ 
4:    $v_{previous} := v_{origin}$ 
5:   for  $v_{current} \in \text{BFITER}(\mathcal{G}, v_{origin})$  do
6:      $V_{sub} := V_{sub} \cup v_{current}$ 
7:      $\rightarrow_{e_{sub}} := \rightarrow_{e_{sub}} \cup \text{EDGE}(v_{previous}, v_{current})$ 
8:      $v_{previous} := v_{current}$ 

```

Graph Queries

Forward & Backward Reach



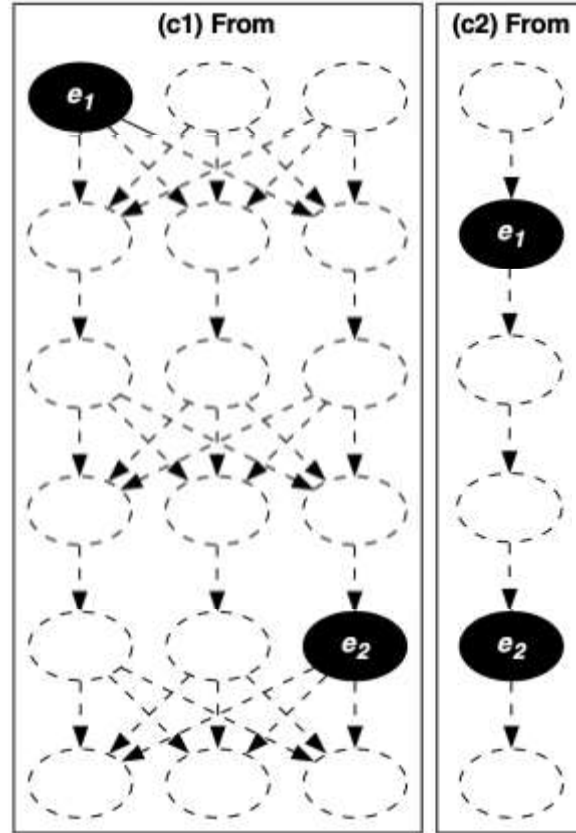
```

1: function REACH( $\mathcal{G}, v_{origin}$ )
2:    $\mathcal{G}_{sub} = (V_{sub}, \rightarrow_{e_{sub}}) := (\emptyset, \emptyset) = \text{SUBGRAPH}(\mathcal{G})$ 
3:    $V_{sub} := V_{sub} \cup v_{origin}$ 
4:    $v_{previous} := v_{origin}$ 
5:   for  $v_{current} \in \text{BFITER}(\mathcal{G}, v_{origin})$  do
6:      $V_{sub} := V_{sub} \cup v_{current}$ 
7:      $\rightarrow_{e_{sub}} := \rightarrow_{e_{sub}} \cup \text{EDGE}(v_{previous}, v_{current})$ 
8:      $v_{previous} := v_{current}$ 

```

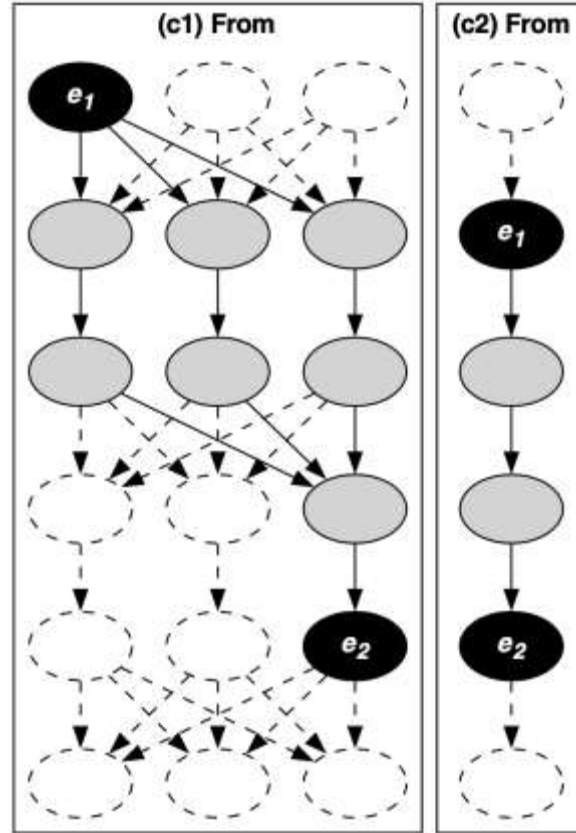
Graph Queries

Reach From



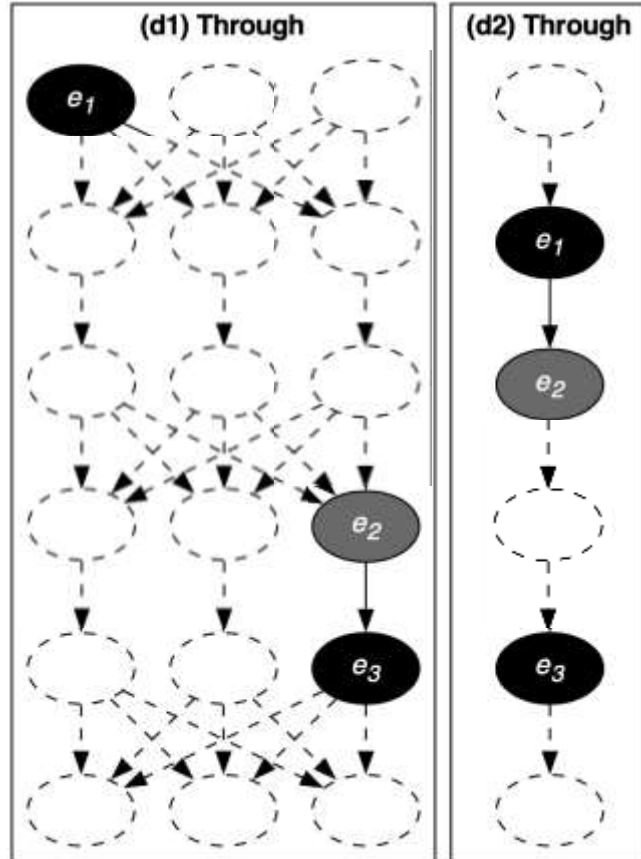
Graph Queries

Reach From



Graph Queries

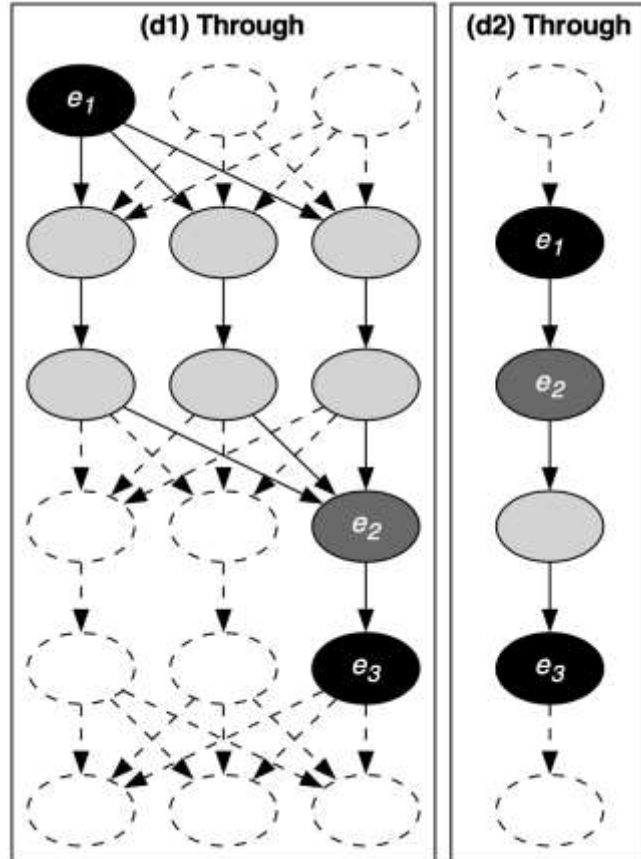
Reach Through



- 1: **function** REACHTHROUGH($\mathcal{G}, v_{origin}, v_{mid}, v_{target}$)
- 2: $\mathcal{G}_{fwd} := \text{REACHFORWARD}(\mathcal{G}, v_{origin})$
- 3: $\mathcal{G}_{back} := \text{REACHBACKWARD}(\mathcal{G}_{fwd}, v_{target})$
- 4: **if** $v_{mid} \notin \text{CUTPOINTS}(\mathcal{G}_{back})$ **then**
 ▷ Remove midpoint and all edges connected to it
- 5: $\mathcal{G}_{mask} := (V \setminus v_{mid}, \rightarrow_e \setminus (v_{mid}, _) \cup (_, v_{mid}))$
- 6: $\mathcal{G}_{path} := \text{PATH}(\mathcal{G}_{mask}, v_{origin}, v_{target})$
- 7: **else**
- 8: $\mathcal{G}_{path} := (\emptyset, \emptyset)$

Graph Queries

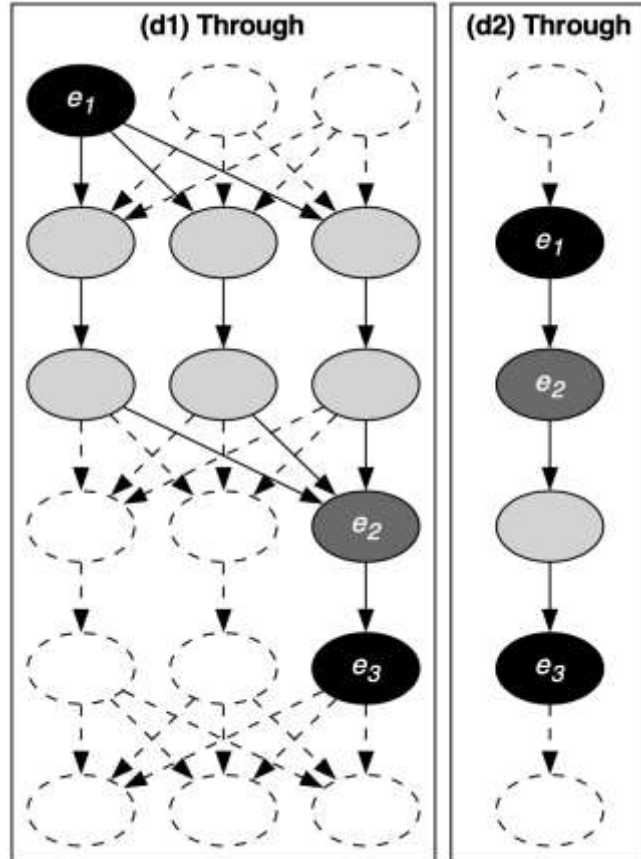
Reach Through



- 1: **function** REACHTHROUGH($\mathcal{G}, v_{origin}, v_{mid}, v_{target}$)
- 2: $\mathcal{G}_{fwd} := \text{REACHFORWARD}(\mathcal{G}, v_{origin})$
- 3: $\mathcal{G}_{back} := \text{REACHBACKWARD}(\mathcal{G}_{fwd}, v_{target})$
- 4: **if** $v_{mid} \notin \text{CUTPOINTS}(\mathcal{G}_{back})$ **then**
 ▷ Remove midpoint and all edges connected to it
- 5: $\mathcal{G}_{mask} := (V \setminus v_{mid}, \rightarrow_e \setminus (v_{mid}, _) \cup (_, v_{mid}))$
- 6: $\mathcal{G}_{path} := \text{PATH}(\mathcal{G}_{mask}, v_{origin}, v_{target})$
- 7: **else**
- 8: $\mathcal{G}_{path} := (\emptyset, \emptyset)$

Graph Queries

Reach Through



- 1: **function** REACHTHROUGH($\mathcal{G}, v_{origin}, v_{mid}, v_{target}$)
- 2: $\mathcal{G}_{fwd} := \text{REACHFORWARD}(\mathcal{G}, v_{origin})$
- 3: $\mathcal{G}_{back} := \text{REACHBACKWARD}(\mathcal{G}_{fwd}, v_{target})$
- 4: **if** $v_{mid} \notin \text{CUTPOINTS}(\mathcal{G}_{back})$ **then**
- 5: ▷ Remove midpoint and all edges connected to it
- 6: $\mathcal{G}_{mask} := (V \setminus v_{mid}, \rightarrow_e \setminus (v_{mid}, _) \cup (_, v_{mid}))$
- 7: $\mathcal{G}_{path} := \text{PATH}(\mathcal{G}_{mask}, v_{origin}, v_{target})$
- 8: **else**
- 9: $\mathcal{G}_{path} := (\emptyset, \emptyset)$

Graph Queries

Others – Validation, Neighbors

Assumption Validation

- Can every error source reach a sink?
- Can every sink be reached from an error source?

Neighbors

- What components, *at a given hierarchical depth*, communicate with a given component?

Agenda

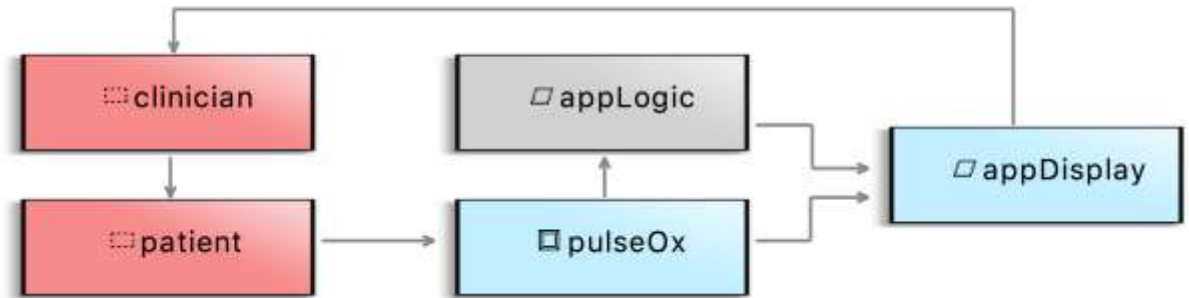
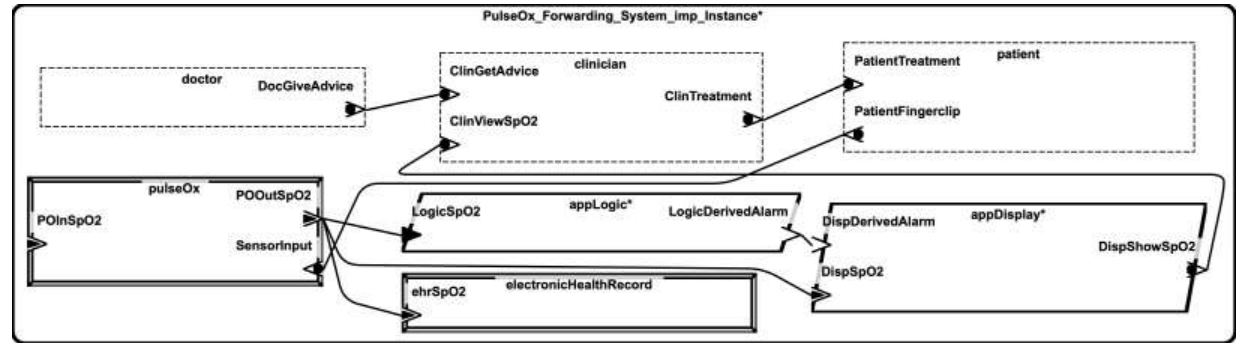
- Introduction & Background
- The OSATE Slicer
- **Evaluation**
 - **Analyses**
 - **Performance**

Suitability for Analyses

Safety – Architecture Supported Audit Processor

Questions from Analysis

- Q1: Who can send messages to a component?
- Q2: Who gets messages a component sends?
- A1: “Neighbors” Query

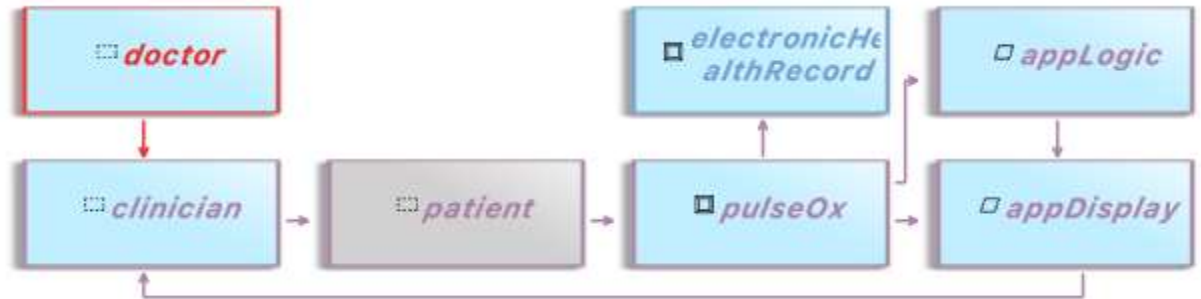
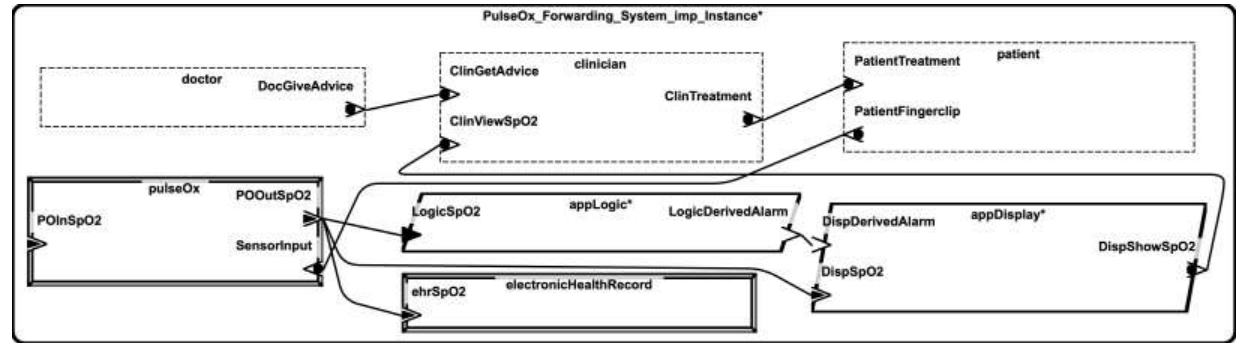


Suitability for Analyses

Safety – Architecture Supported Audit Processor

Questions from Analysis

- Q1: Who can send messages to a component?
- Q2: Who gets messages a component sends?
- A1: “Neighbors” Query
- Q3: Who can be affected by affected by a component?
- Q4: Who affects a component?
- Q5: Are there feedback loops present?
- A2: Forward + Backward Slice + Overlap



Suitability for Analyses

Safety, Security, Latency

Safety

Fault Impact

Question: If this error occurs, where does it go? What happens?

Answer: Forward slice

Security

Bell-LaPadula

Classic security policy, 3 of 4 properties can be (potentially) verified using the Slicer.

Attack Trees

Existing implementation is brittle and presents maintenance challenges.

Performance

System Latency

Popular analysis, many special cases. Requires support for additional AADL features, e.g., modes.

Performance Relative to Awacs



The OSATE Slicer: Graph-Based Reachability for Architectural Models

JULY 20, 2023

Sam Procter

