Kronecker Algebra for Static Analysis of Ada Programs with Protected Objects

Bernd Burgstaller[1] and Johann Blieberger[2]

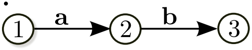
[1]Yonsei University

[2]Vienna University of Technology

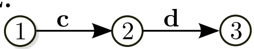
Motivation: Task Interleavings

- Arbitrary interleaving of tasks t1 and t2: order on computation steps where each step is taken from t1 or t2 in program order
- Totality of all possible arbitrary interleavings well-suited for concurrent program analysis & comprehension

Task t1:



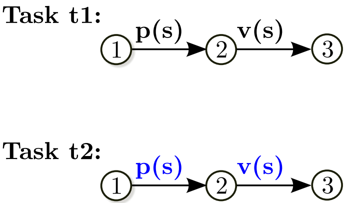
Task t2:



Interleavings
$a \cdot b \cdot c \cdot d$
$a \cdot c \cdot b \cdot d$
$a \cdot c \cdot d \cdot b$
$c \cdot a \cdot b \cdot d$
$c \cdot a \cdot d \cdot b$
$c \cdot d \cdot a \cdot b$

Non-Arbitrary Interleavings

- Semantics of synchronization primitives constrain possible interleavings
  - Example: binary semaphore s



Interleavings

$p(s) \cdot v(s) \cdot p(s) \cdot v(s)$ ✓

$p(s) \cdot p(s) \cdot v(s) \cdot v(s)$ ✗

$p(s) \cdot p(s) \cdot v(s) \cdot v(s)$ ✗

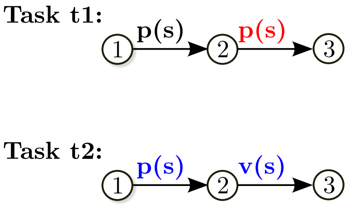
$p(s) \cdot p(s) \cdot v(s) \cdot v(s)$ ✗

$p(s) \cdot p(s) \cdot v(s) \cdot v(s)$ ✗

$p(s) \cdot v(s) \cdot p(s) \cdot v(s)$ ✓

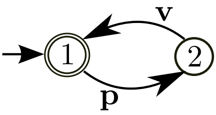
Non-Arbitrary Interleavings (cont.)

- Semantics of synchronization primitives allow additional interleavings that constitute deadlocks
  - Example: binary semaphore, self-deadlock of Task t1:



Interleavings
p(s) · v(s) · p(s)
p(s)

- Internal behavior of semaphore can be modelled by a deterministic finite automaton (DFA):
  - Example: binary semaphore



Internal Behavior of Protected Objects

```
protected body BlackBox is
   entry Some when ⟨condition⟩ is
   begin
      -- User-supplied code . . .
   end Some;
end BlackBox;
```

- Behavior of POs is characterized by:
  1. The "boilerplate" semantics of entries, procedures & functions prescribed by the Ada RM
  2. The user-supplied code
- For static analysis of POs, "understanding" of the user-supplied code is necessary

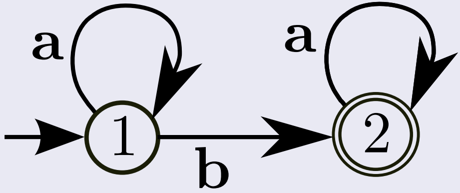
Contributions

- Algebra-based approach to model protected objects
  - Incorporates user-supplied code into analysis
  - Capable to generate all interleavings of PO-related task communication
- Graph templates for protected entries, procedures and functions
  - Adaptable to chosen implementation
  - Concrete instances for the "eggshell model"
- Symbolic analysis approach to eliminate infeasible (dead) program paths

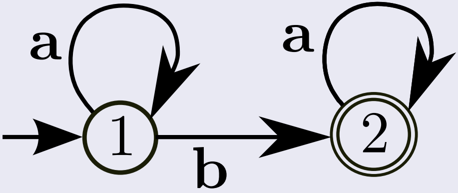
Outline

1. Motivation
2. Matrix Representation of DFAs
3. Kronecker Algebra of Matrices
4. Semaphores
5. Protected Object Graph Templates
6. Running Example
7. Static Analysis

Matrices representing DFAs

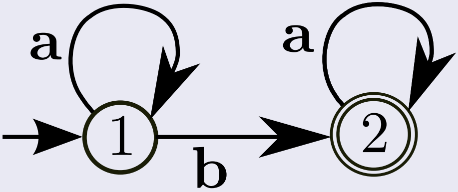


Matrices representing DFAs



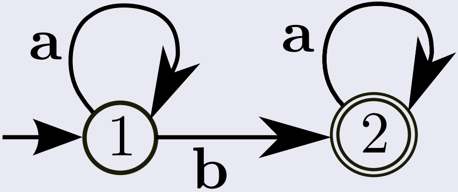
$$M = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$$

2 Hops in automaton $= M^2$



$$\begin{pmatrix} a & b \\ 0 & a \end{pmatrix} \cdot \begin{pmatrix} a & b \\ 0 & a \end{pmatrix} = \begin{pmatrix} a^2 & ab + ba \\ 0 & a^2 \end{pmatrix}$$

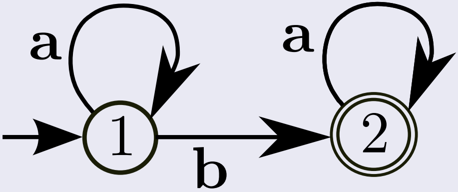
3 Hops in automaton $= M^3$



$$\begin{pmatrix} a^2 & ab+ba \\ 0 & a^2 \end{pmatrix} \cdot \begin{pmatrix} a & b \\ 0 & a \end{pmatrix} =$$

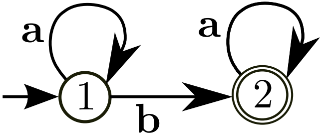
$$= \begin{pmatrix} a^3 & a^2b+aba+ba^2 \\ 0 & a^3 \end{pmatrix}$$

k Hops in automaton $= M^k$



$$\begin{pmatrix} a & b \\ 0 & a \end{pmatrix}^k = \begin{pmatrix} a^k & \sum_{i=0}^{k-1} a^i b a^{k-i-1} \\ 0 & a^k \end{pmatrix}$$

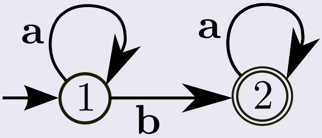
M^*



$$M^* = \sum_{k\geq 0} M^k = \begin{pmatrix} \sum_{k\geq 0} a^k & \sum_{k\geq 1}\sum_{i=0}^{k-1} a^i b a^{k-i-1} \\ 0 & \sum_{k\geq 0} a^k \end{pmatrix}$$

$$= \begin{pmatrix} a^* & a^* b a^* \\ 0 & a^* \end{pmatrix}$$

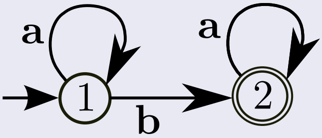
Start and Final Nodes



$$\text{Column Vector } F = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

1s where there is a **final** node; 0s otherwise

Start and Final Nodes



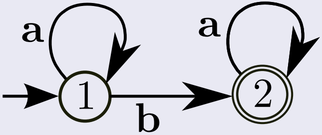
$$\text{Column Vector } F = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

1s where there is a **final** node; 0s otherwise

$$\text{Row Vector } S = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

1 if **start** node; 0s otherwise

Behaviour of Automaton



$$S \cdot M^* \cdot F =$$

$$= \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} a^* & a^*ba^* \\ 0 & a^* \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} =$$

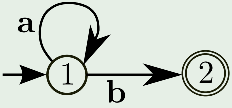
$$= a^*ba^*$$

Kronecker Product

Definition

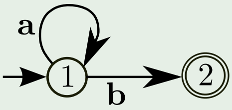
Given a m-by-n matrix A and a p-by-q matrix B, their Kronecker product denoted by $A \otimes B$ is a mp-by-nq block matrix defined by $A \otimes B = \begin{pmatrix} a_{1,1}B & \dots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \dots & a_{m,n}B \end{pmatrix}$.

Example

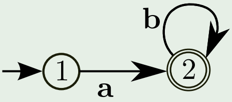


$$A = \begin{pmatrix} a & b \\ 0 & 0 \end{pmatrix}$$

Example

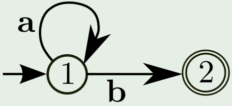


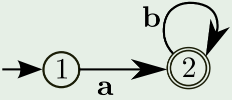
$$A = \begin{pmatrix} a & b \\ 0 & 0 \end{pmatrix}$$



$$B = \begin{pmatrix} 0 & a \\ 0 & b \end{pmatrix}$$

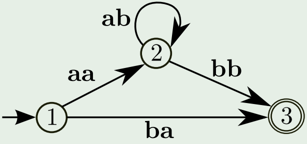
Example



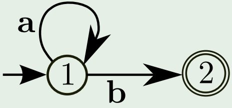


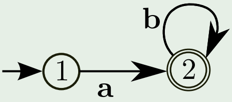
$$A = \begin{pmatrix} a & b \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & a \\ 0 & b \end{pmatrix}$$

$$A \otimes B = \begin{pmatrix} 0 & aa & 0 & ba \\ 0 & ab & 0 & bb \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



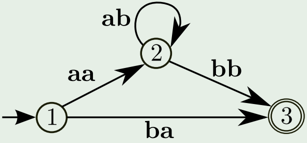
Example





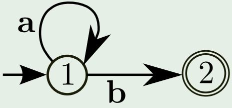
$$A = \begin{pmatrix} a & b \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & a \\ 0 & b \end{pmatrix}$$

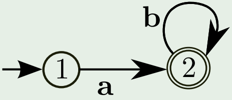
$$A \otimes B = \begin{pmatrix} 0 & aa & 0 & ba \\ 0 & ab & 0 & bb \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



- Simultaneous execution of A and B

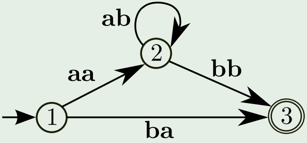
Example





$$A = \begin{pmatrix} a & b \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & a \\ 0 & b \end{pmatrix}$$

$$A \otimes B = \begin{pmatrix} 0 & aa & 0 & ba \\ 0 & ab & 0 & bb \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



- Simultaneous execution of A and B
- $\otimes$ can be used to “synchronize” automata

Kronecker Sum

Definition

Given a matrix A of order m and a matrix B of order n, their Kronecker sum denoted by $A \oplus B$ is a matrix of order mn defined by

$$A \oplus B = A \otimes I_n + I_m \otimes B,$$

where I_m and I_n denote identity matrices of order m and n, respectively.

Example

$$① \xrightarrow{\mathbf{a}} ② \xrightarrow{\mathbf{b}} ③$$

$$A = \begin{pmatrix} 0 & a & 0 \\ 0 & 0 & b \\ 0 & 0 & 0 \end{pmatrix}$$

Example

$$1 \xrightarrow{\mathbf{a}} 2 \xrightarrow{\mathbf{b}} 3$$

$$A = \begin{pmatrix} 0 & a & 0 \\ 0 & 0 & b \\ 0 & 0 & 0 \end{pmatrix}$$

$$1 \xrightarrow{\mathbf{c}} 2 \xrightarrow{\mathbf{d}} 3$$

$$B = \begin{pmatrix} 0 & c & 0 \\ 0 & 0 & d \\ 0 & 0 & 0 \end{pmatrix}$$

Example

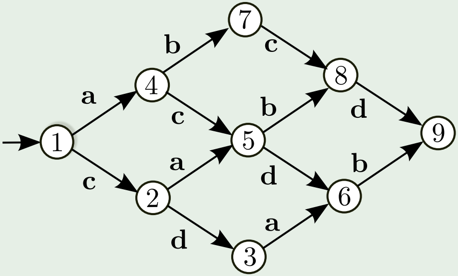
$1 \xrightarrow{\mathbf{a}} 2 \xrightarrow{\mathbf{b}} 3$

$$A = \begin{pmatrix} 0 & a & 0 \\ 0 & 0 & b \\ 0 & 0 & 0 \end{pmatrix}$$

$1 \xrightarrow{\mathbf{c}} 2 \xrightarrow{\mathbf{d}} 3$

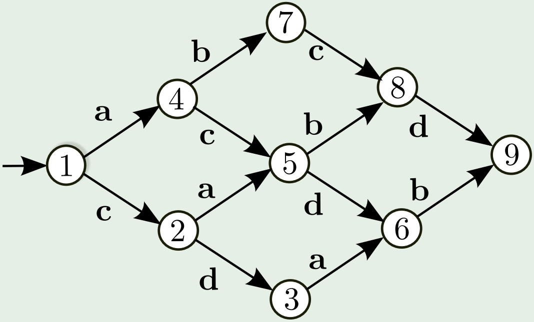
$$B = \begin{pmatrix} 0 & c & 0 \\ 0 & 0 & d \\ 0 & 0 & 0 \end{pmatrix}$$

$A \oplus B$



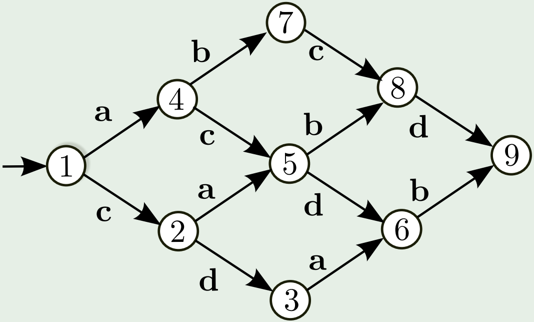
Example

Interleavings
$a \cdot b \cdot c \cdot d$
$a \cdot c \cdot b \cdot d$
$a \cdot c \cdot d \cdot b$
$c \cdot a \cdot b \cdot d$
$c \cdot a \cdot d \cdot b$
$c \cdot d \cdot a \cdot b$



Example

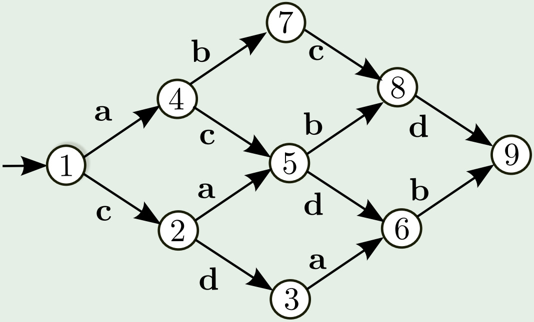
Interleavings
$a \cdot b \cdot c \cdot d$
$a \cdot c \cdot b \cdot d$
$a \cdot c \cdot d \cdot b$
$c \cdot a \cdot b \cdot d$
$c \cdot a \cdot d \cdot b$
$c \cdot d \cdot a \cdot b$



- interleaving

Example

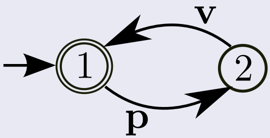
Interleavings
$a \cdot b \cdot c \cdot d$
$a \cdot c \cdot b \cdot d$
$a \cdot c \cdot d \cdot b$
$c \cdot a \cdot b \cdot d$
$c \cdot a \cdot d \cdot b$
$c \cdot d \cdot a \cdot b$



- interleaving
- $\oplus$ can be used to model concurrency

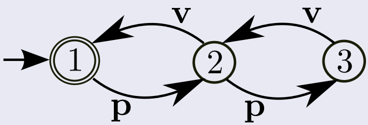
Semaphores

Binary Semaphore:



$$S = \begin{pmatrix} 0 & p \\ v & 0 \end{pmatrix}$$

Counting Semaphore:



$$S = \begin{pmatrix} 0 & p & 0 \\ v & 0 & p \\ 0 & v & 0 \end{pmatrix}$$

Concurrent Threads

Let $T^{(i)}$ be the matrix of the control flow graph of thread i.

Concurrent Threads

Let $T^{(i)}$ be the matrix of the control flow graph of thread i.

Then $T = \bigoplus_{i=1}^{k} T^{(i)}$ models all interleavings of the threads.

Concurrent Threads

Let $T^{(i)}$ be the matrix of the control flow graph of thread i.

Then $T = \bigoplus_{i=1}^{k} T^{(i)}$ models all interleavings of the threads.

Edge labels are

- IDs of basic blocks and
- p_j and v_j for semaphore calls to semaphore j.

Concurrent Threads

Let $T^{(i)}$ be the matrix of the control flow graph of thread i.

Then $T = \bigoplus_{i=1}^{k} T^{(i)}$ models all interleavings of the threads.

Edge labels are

- IDs of basic blocks and
- p_j and v_j for semaphore calls to semaphore j.

Edge labeling requires splitting of basic blocks (edges).

“Concurrent” Semaphores

Let $S^{(j)}$ be the matrix of semaphore j.

“Concurrent” Semaphores

Let $S^{(j)}$ be the matrix of semaphore j.

Then $S = \bigoplus_{j=1}^{r} S^{(j)}$ models all interleavings of the semaphores.

"Concurrent" Semaphores

Let $S^{(j)}$ be the matrix of semaphore j.

Then $S = \bigoplus_{j=1}^{r} S^{(j)}$ models all interleavings of the semaphores.

Edge labels are

- p_j and v_j.

“Synchronizing” Threads and Semaphores

- Split matrix T into summands T_S and T_V such that $T = T_S + T_V$, T_S contains only semaphore calls, and T_V contains the other edge labels.

"Synchronizing" Threads and Semaphores

- Split matrix T into summands T_S and T_V such that $T = T_S + T_V$, T_S contains only semaphore calls, and T_V contains the other edge labels.
- Behaviour of the overall system can be modelled by

$$P = T_S \otimes S + T_V \oplus S$$

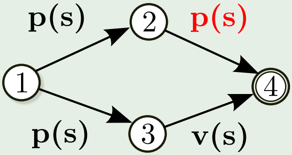
“Synchronizing” Threads and Semaphores

- Split matrix T into summands T_S and T_V such that $T = T_S + T_V$, T_S contains only semaphore calls, and T_V contains the other edge labels.
- Behaviour of the overall system can be modelled by

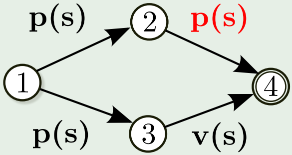
$$P = T_S \otimes S + T_V \oplus S$$

- For simplicity we write $p_j \cdot p_j = p_j$, $v_j \cdot v_j = v_j$, and $\boldsymbol{p_i \cdot x_j = v_i \cdot x_j = 0}$.

Example

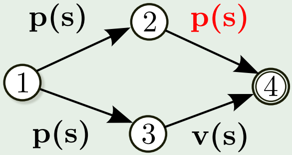


Example



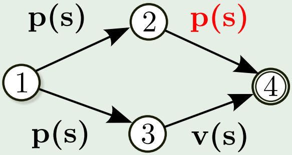
$$\begin{pmatrix} 0 & \mathbf{p} & \mathbf{p} & 0 \\ 0 & 0 & 0 & \color{red}{\mathbf{p}} \\ 0 & 0 & 0 & \mathbf{v} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Example



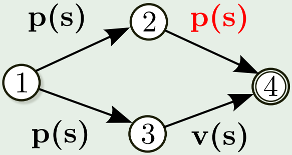
$$\begin{pmatrix} 0 & \mathbf{p} & \mathbf{p} & 0 \\ 0 & 0 & 0 & \color{red}{\mathbf{p}} \\ 0 & 0 & 0 & \mathbf{v} \\ 0 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & \mathbf{p} \\ \mathbf{v} & 0 \end{pmatrix} =$$

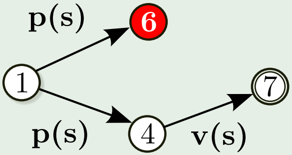
Example



$$\begin{pmatrix} 0 & \mathbf{p} & \mathbf{p} & 0 \\ 0 & 0 & 0 & \mathbf{p} \\ 0 & 0 & 0 & \mathbf{v} \\ 0 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & \mathbf{p} \\ \mathbf{v} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & \mathbf{p} & 0 & \mathbf{p} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{v} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

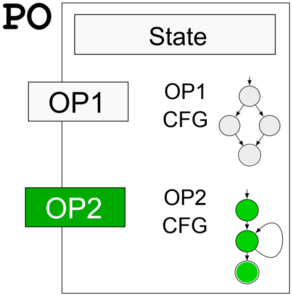
Example





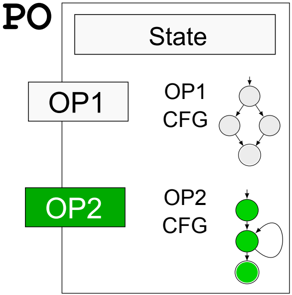
$$\begin{pmatrix} 0 & \mathbf{p} & \mathbf{p} & 0 \\ 0 & 0 & 0 & \mathbf{p} \\ 0 & 0 & 0 & \mathbf{v} \\ 0 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & \mathbf{p} \\ \mathbf{v} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & \mathbf{p} & 0 & \mathbf{p} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{v} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

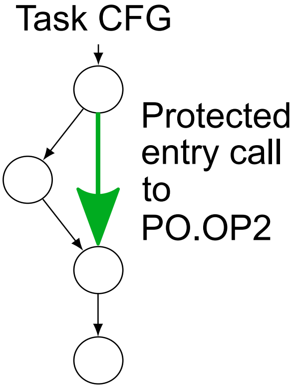
Modelling Protected Objects



- User provides protected object implementation PO

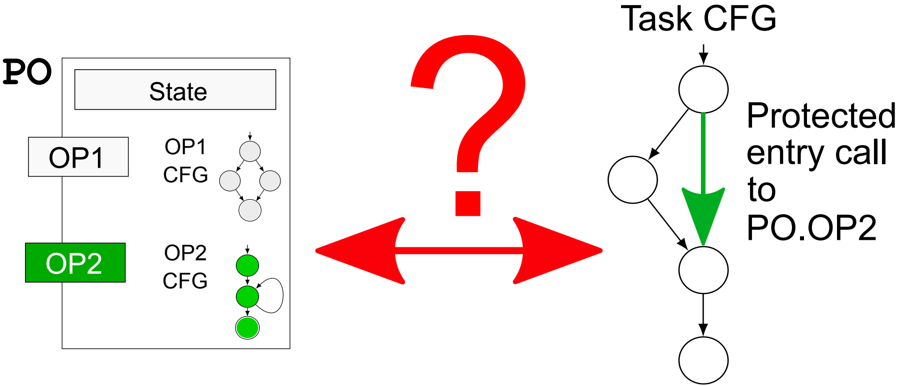
Modelling Protected Objects





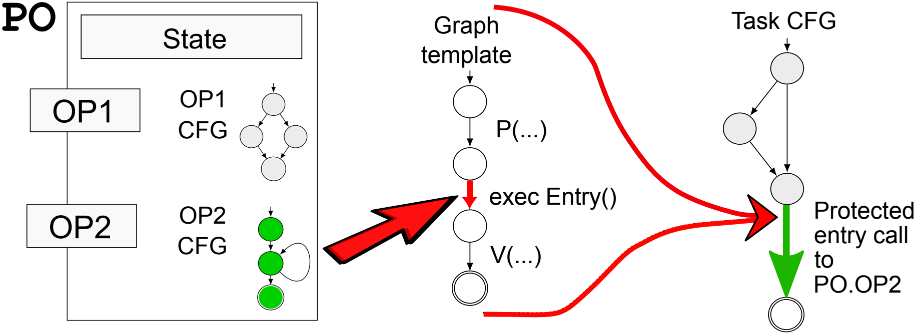
- User provides protected object implementation PO
- User provides task calling entry OP2

Modelling Protected Objects



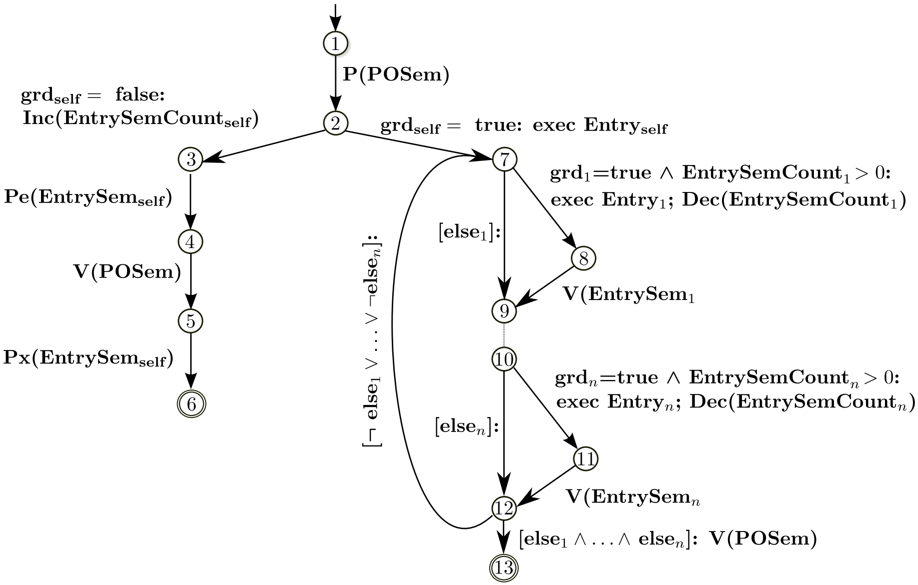
- User provides protected object implementation PO
- User provides task calling entry OP2

Graph Template Instantiation

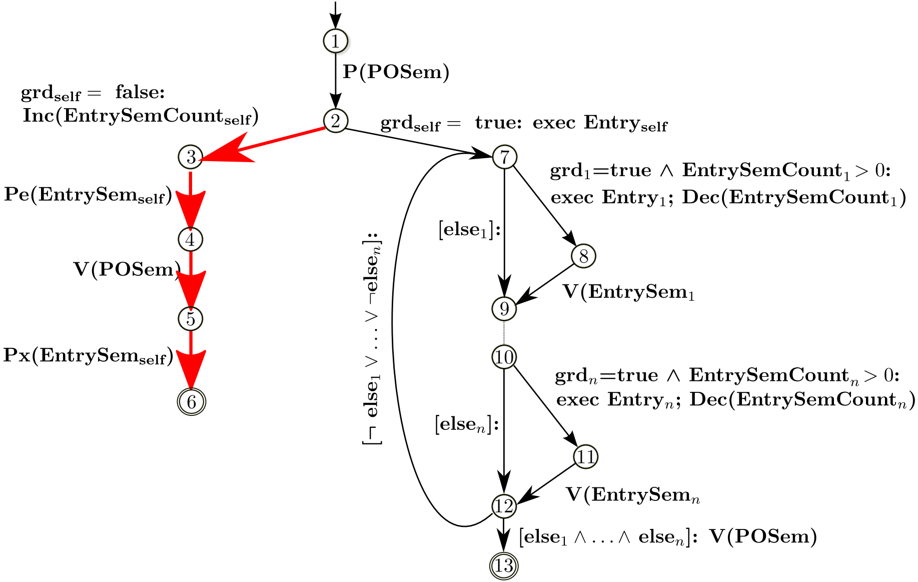


- Graph template connects protected entry's user-code with task call-site
- Provides the concurrency-semantics prescribed by the RM, clear separation with user-code
- Instantiation: insert entry code into template, then insert instantiated template at task's call-site

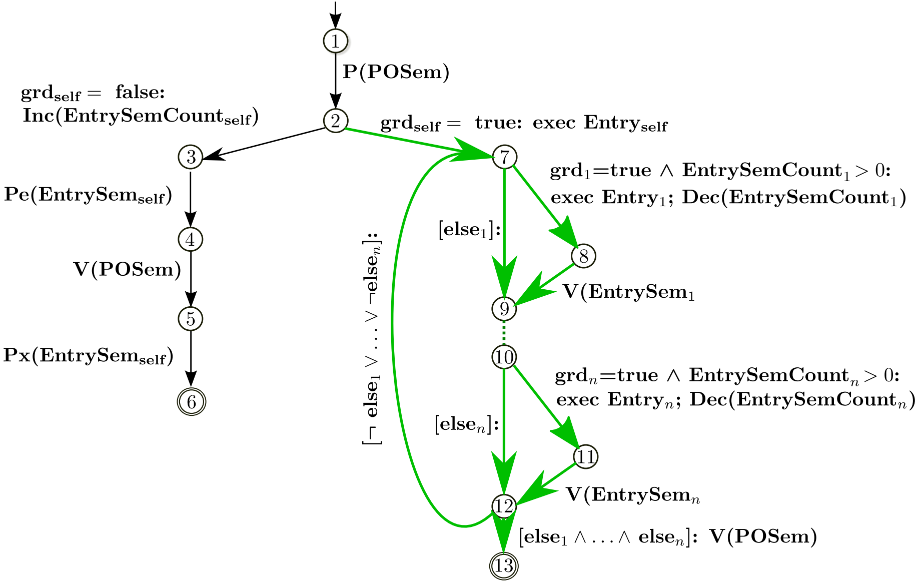
Protected Entry Graph Template



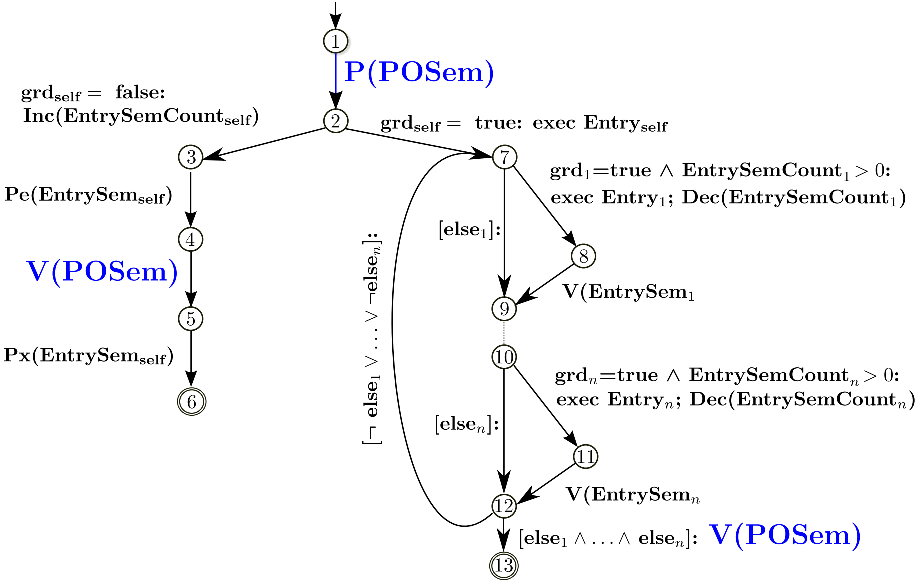
Blocking Task on Closed Entry



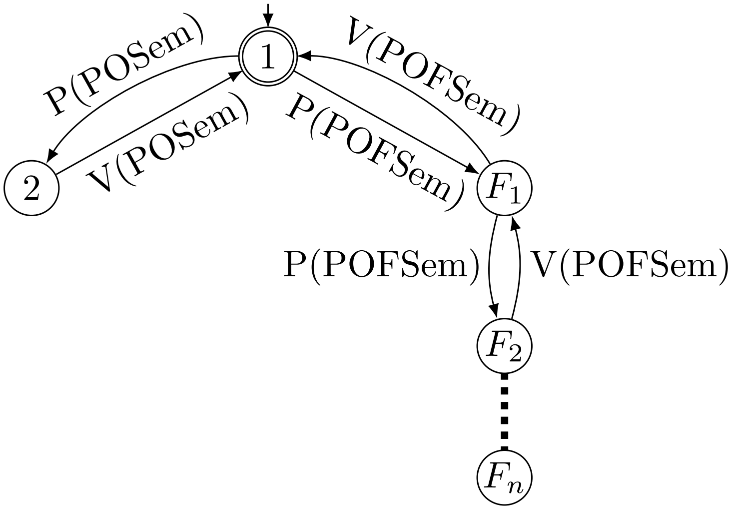
Entry Execution and Proxy Execution



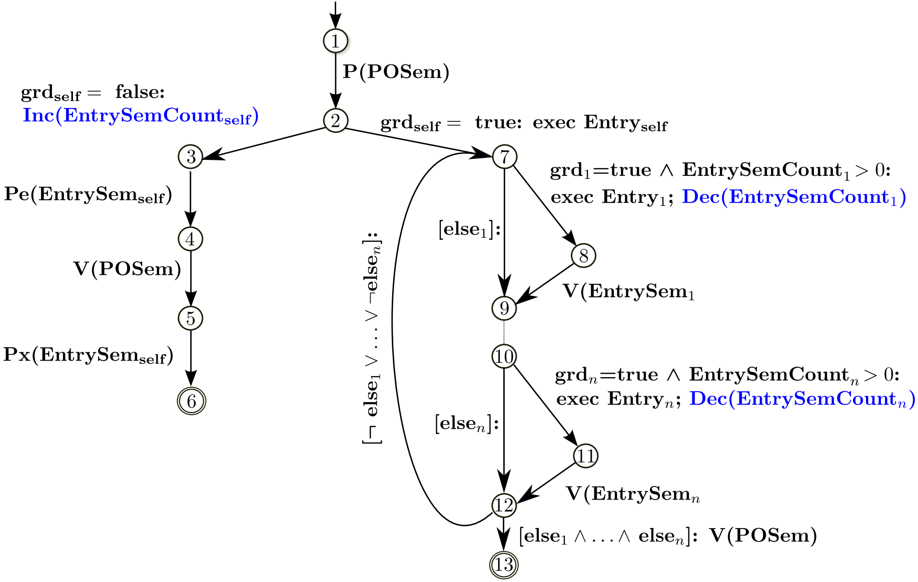
Protected Object Master Lock: POSem



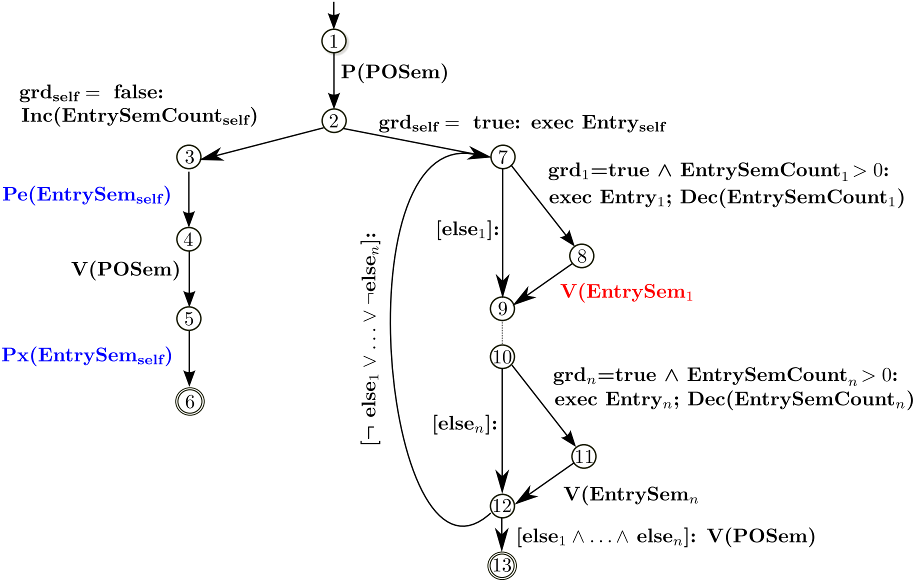
POSem (cont.)



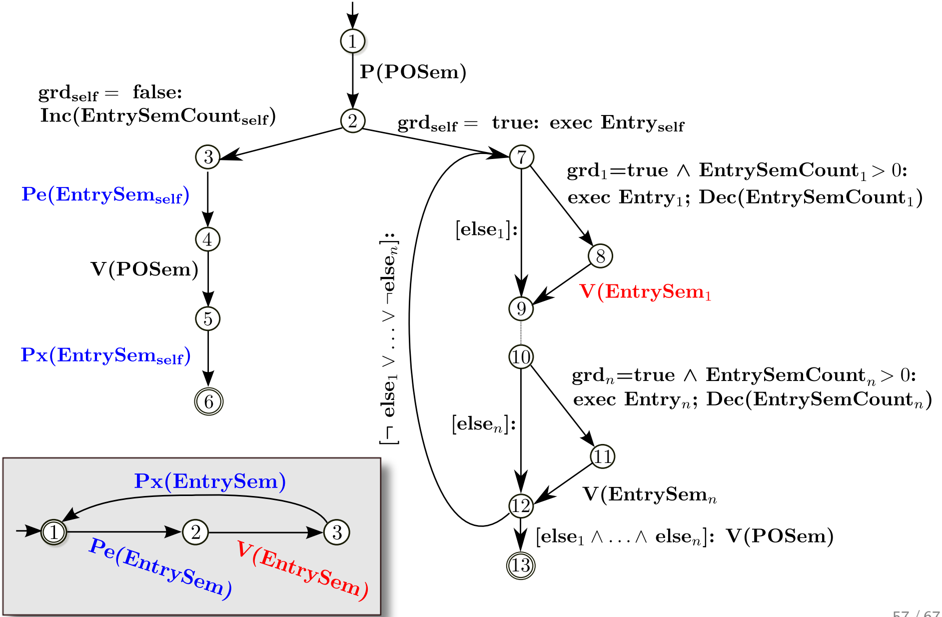
of tasks queued on closed entry



Blocking tasks on closed entries



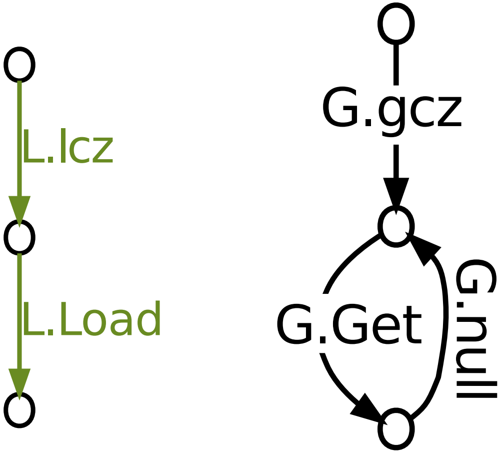
Blocking tasks on closed entries



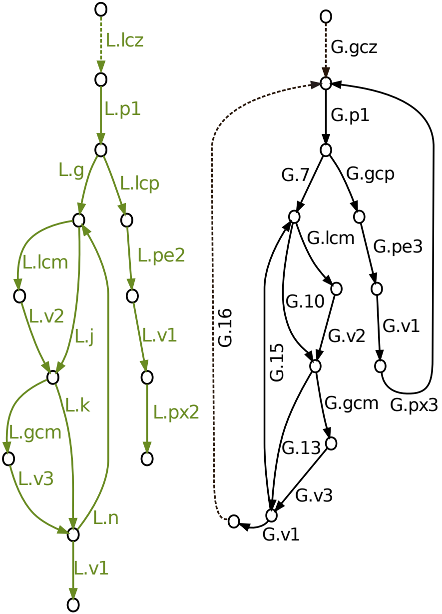
Running Example

```
protected type Buffer (M: Integer)
is
 entry Load (S: in String);
 entry Get (C: out Character);
end Buffer;

protected body Buffer is
 entry Load(S: in String)
   when BufferEmpty is
  begin
   -- load buffer with S
  end Load;

  entry Get(C: out Character)
   when not BufferEmpty is
  begin
   -- return next character
  end Get; end Buffer;
B: Buffer(16);

task Getter;
task body Getter is
 C: Character;
begin
 loop
  B.Get(C);
 end loop;
end Getter;

begin
B.Load("Hello!");
end Example;
```

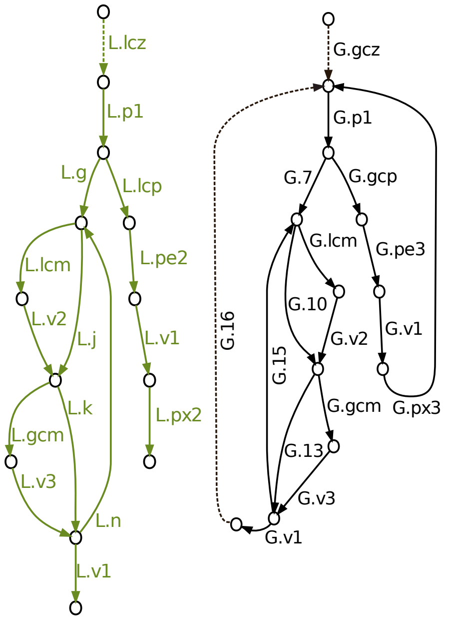
Running Example CFGs



Running Example CFGs



Running Example CFGs



- Task matrix T consists of Kronecker sum of Load and Get CFGs
- Synchronization Matrix S consist of Kronecker sum of
  - 1 POSem
  - 2 EntrySems (Load & Get entries)
- Result matrix (CPG):
  - size 7560x7560
  - 87 nodes, 128 edges
  - 13 deadlock nodes
  - 116ms construction time

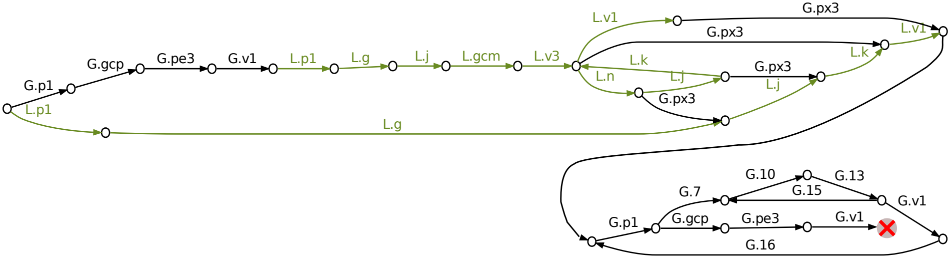
False Positives

- Matrix algebra not concerned with edge conditions on labels
- Results in infeasible paths from CPG start-node to deadlock node
  - False positive deadlock
- Example: Getter task's guard will only become open after loader has filled the buffer
- We employ static analysis to detect infeasible program paths

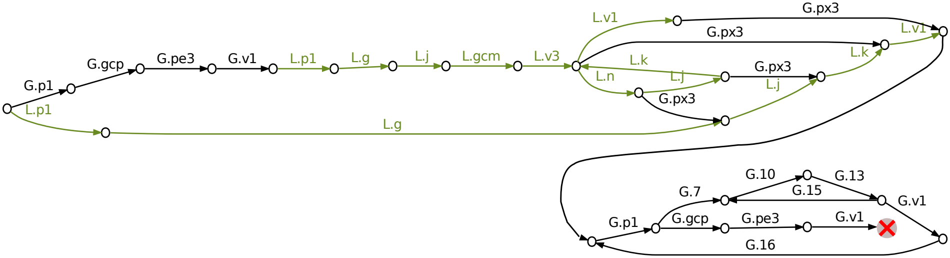
Infeasible Program Path Detection

- Symbolic analysis uses symbolic expressions to describe computations as algebraic formulae
  - Derives all valid variable bindings at given program point
- We want to detect dead paths to reduce the number of false positives
- Analysis problem: prove edge condition to be false on all paths through the CPG
- For edge $e = (s \rightarrow t)$, if edge condition proven false, t is no longer reachable
  - Nodes only reachable via node t become unreachable, too
  - Cut nodes and adjacent edges along t's dominance frontier

Running Example (cont.)



Running Example (cont.)



- Pruned result matrix (CPG):
  - 27 nodes, 33 edges
  - 1 deadlock node

Conclusions

- Introduced Algebra-based approach to model protected objects
  - Incorporates user-supplied code into analysis
  - Capable to generate all interleavings of PO-related task communication
- Graph templates for protected entries, procedures and functions
  - Adaptable to chosen implementation
  - Concrete instances for the "eggshell model"
- Symbolic analysis approach to eliminate infeasible deadlocks (dead program paths)

Thank You!