Deductive Temporal Verification of Parametrized Concurrent Systems

Alejandro Sánchez[1] César Sánchez[1,2]

[1]IMDEA Software Institute, Spain

[2]Spanish Council for Scientific Research (CSIC), Spain

SVARM'11, Saarbrücken, 2 April 2011

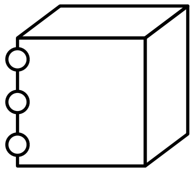
Verification of Concurrent Data-structures

Main Idea

Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Verification of Concurrent Data-structures

Main Idea

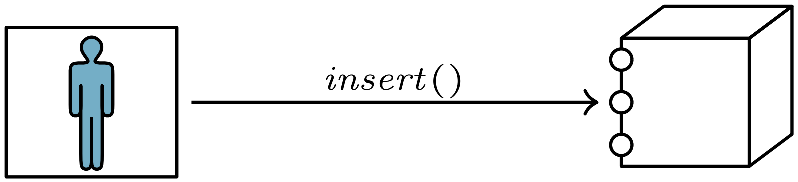
Concurrent DataStructure

Most General Client

Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

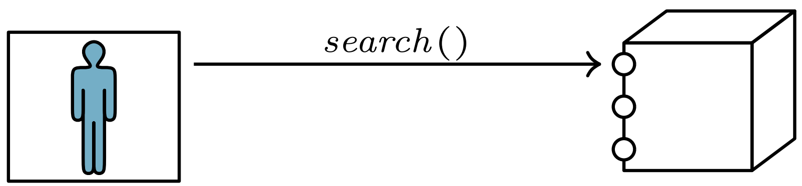


Most General Client

Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

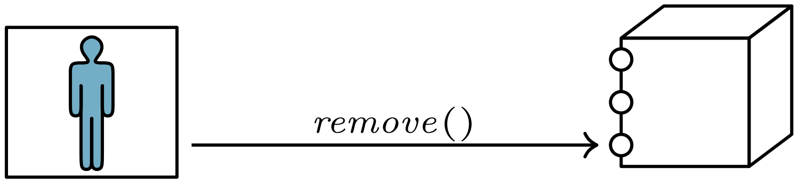


Most General Client

Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

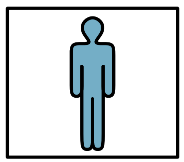


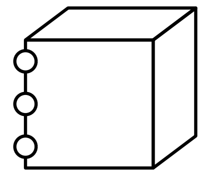
Most General Client

Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



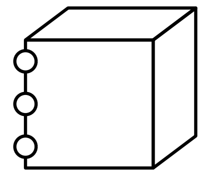


Most General Client

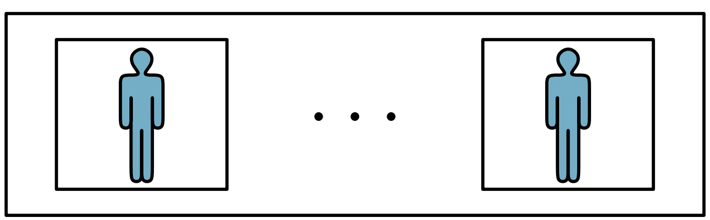
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



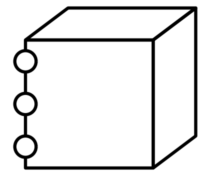
Most General Client



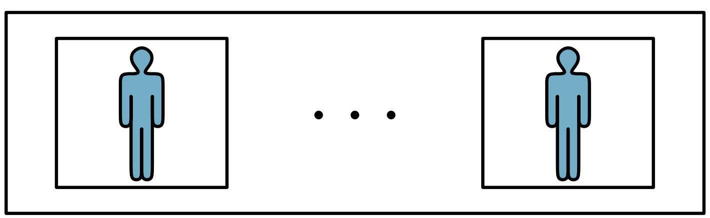
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client

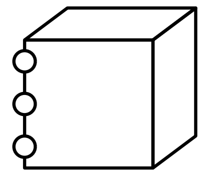


$$P[N] : P(1) || \cdots || P(N)$$

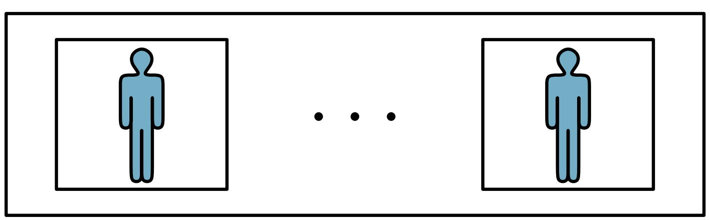
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$$P[N] : P(1)||\cdots||P(N)$$

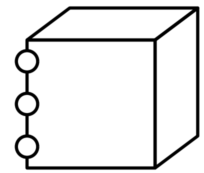
Property

$$\varphi^{(k)}$$

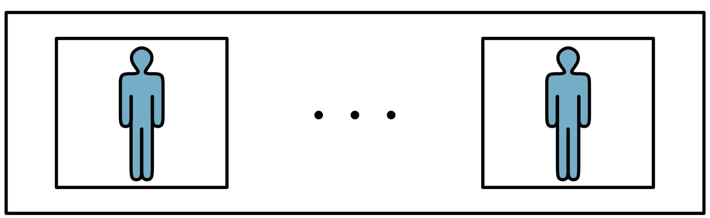
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$P[N] : P(1) || \cdots || P(N)$

Property

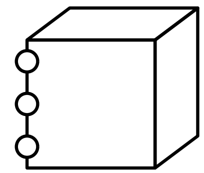
$\varphi^{(k)}$

LTL ($\Box$,$\Diamond$,$\mathcal{U}$,...)

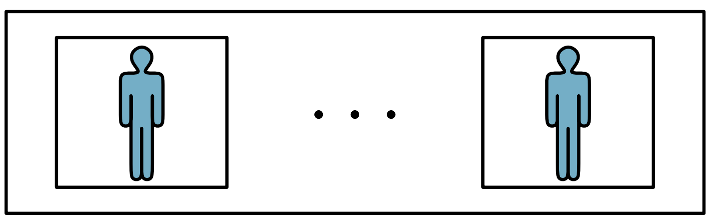
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$P[N] : P(1) || \cdots || P(N)$

Diagram

$\mathcal{D}$

Property

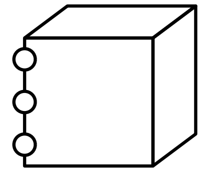
$\varphi^{(k)}$

LTL ($\Box$,$\Diamond$,$\mathcal{U}$,. . .)

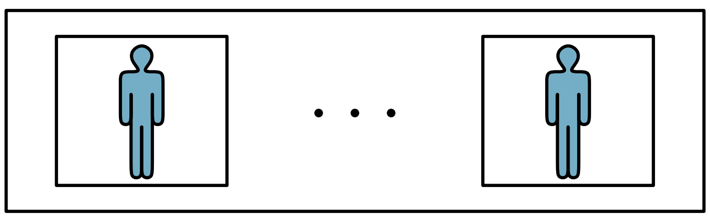
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$P[N] : P(1)||\cdots||P(N)$

Diagram

Property

$\models \quad \mathcal{D} \quad \models \quad \varphi^{(k)}$

Verification Conditions:

- Initiation
- Consecution
- Acceptance
- Fairness

Satisfaction (Model Checking)

Motivating Example: Mutual Exclusion Algorithm

Motivating Example: Mutual Exclusion Algorithm

global
  $\mathit{Int}\ \mathit{tick} := 0$
  $\mathit{Set}\langle \mathit{Int}\rangle\ \mathit{announced} := \emptyset$

procedure $\textsc{MutExc}$
  $\mathit{Int}\ \mathit{ticket}$
begin
loop
  nondet
    $\left\langle \begin{array}{l} \mathit{ticket} := \mathit{tick}++ \\ \mathit{announced}.\mathit{add}(\mathit{ticket}) \end{array} \right\rangle$
  await $(\mathit{announced}.\mathit{min} == \mathit{ticket})$
  critical
  $\mathit{announced}.\mathit{remove}(\mathit{ticket})$
end loop
end procedure

Motivating Example: Mutual Exclusion Algorithm

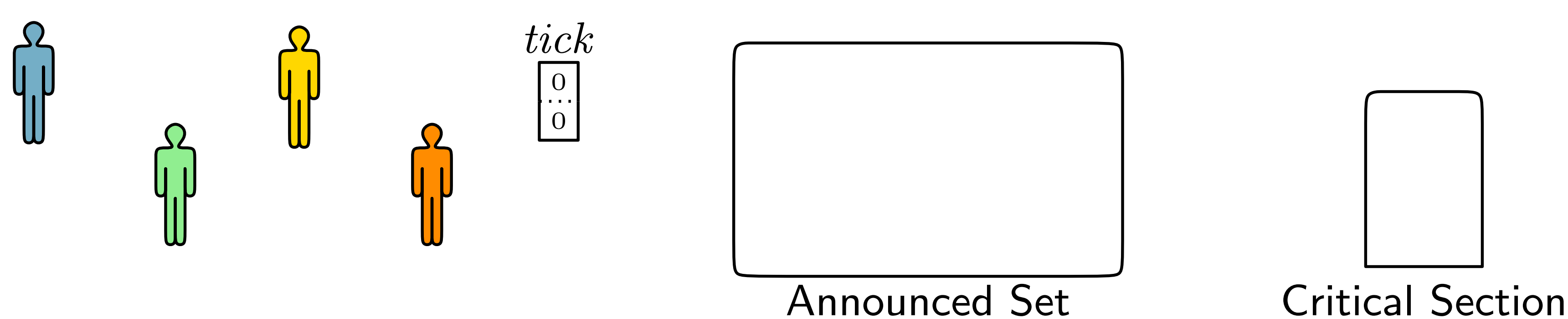
global
$\quad Int\ tick := 0$
$\quad Set\langle Int\rangle\ announced := \emptyset$

procedure $\textsc{MutExc}$
$\quad Int\ ticket$
begin
loop
$\quad$ **nondet**
$\quad \left\langle \begin{array}{l} ticket := tick{+}{+} \\ announced.add(ticket) \end{array} \right\rangle$
$\quad$ **await** $(announced.min == ticket)$
$\quad$ **critical**
$\quad announced.remove(ticket)$
end loop
end procedure

Motivating Example: Mutual Exclusion Algorithm

global
  $Int\ tick := 0$
  $Set\langle Int\rangle\ announced := \emptyset$

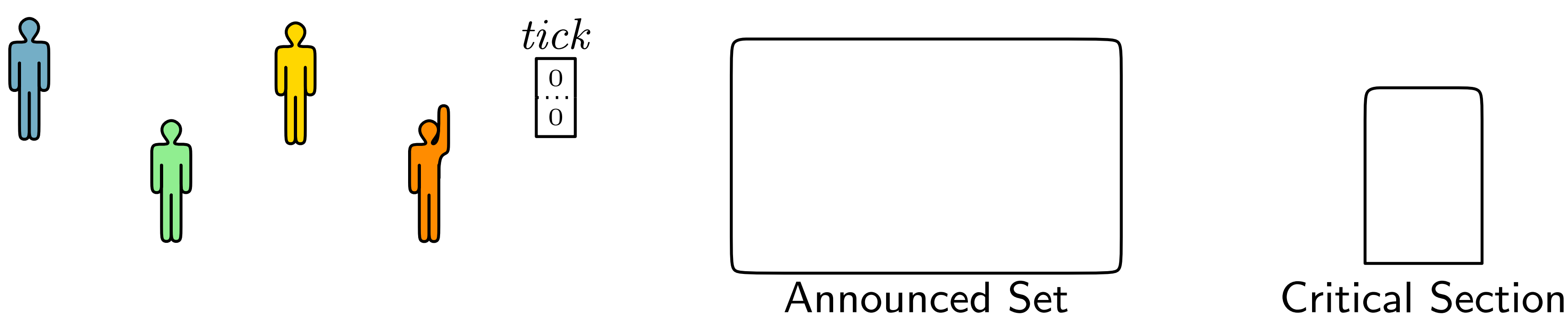
procedure MutExc
  $Int\ ticket$
begin
loop
  nondet
    $\left\langle \begin{array}{l} ticket := tick++ \\ announced.add(ticket) \end{array} \right\rangle$
  await $(announced.min == ticket)$
  critical
  $announced.remove(ticket)$
end loop
end procedure



Motivating Example: Mutual Exclusion Algorithm

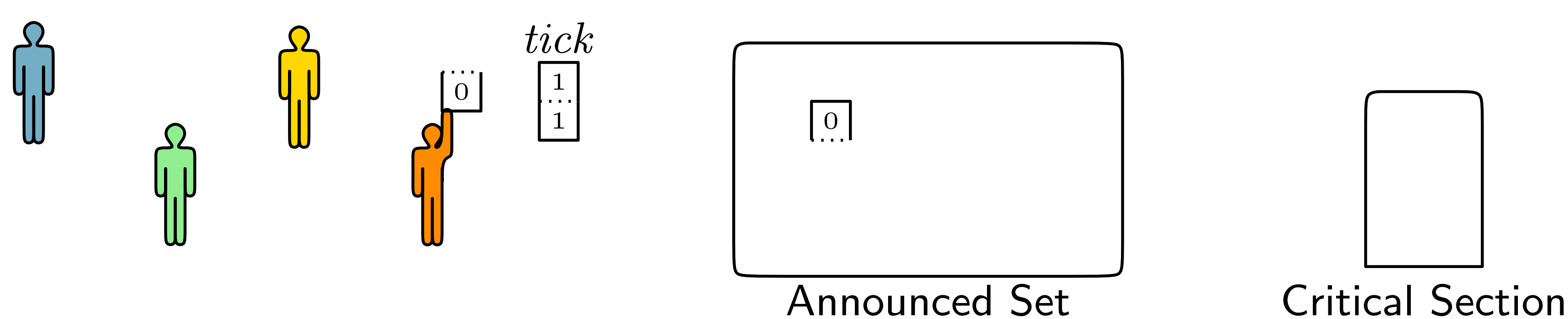
global
  $Int\ tick := 0$
  $Set\langle Int\rangle\ announced := \emptyset$

procedure MUTEXC
  $Int\ ticket$
begin
loop
  nondet
  $\left\langle \begin{array}{l} ticket := tick++ \\ announced.add(ticket) \end{array} \right\rangle$
  await $(announced.min == ticket)$
  critical
  $announced.remove(ticket)$
end loop
end procedure



Motivating Example: Mutual Exclusion Algorithm

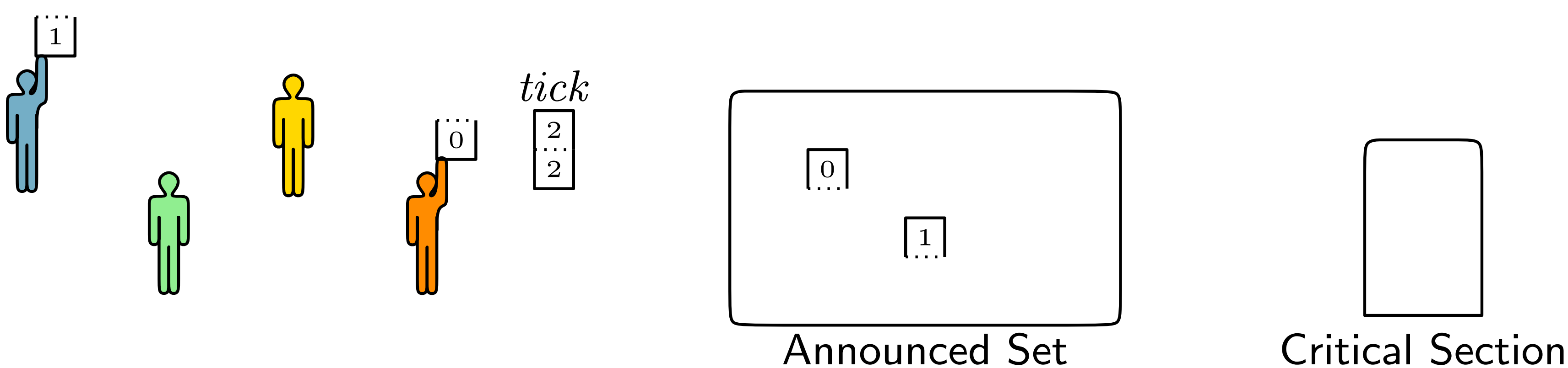
```
global
  Int tick := 0
  Set⟨Int⟩ announced := ∅

procedure MutExc
  Int ticket
begin
loop
    nondet
    ⟨ ticket := tick++
      announced.add(ticket) ⟩
    await (announced.min == ticket)
    critical
    announced.remove(ticket)
end loop
end procedure
```



Motivating Example: Mutual Exclusion Algorithm

global
$\quad Int\ tick := 0$
$\quad Set\langle Int\rangle\ announced := \emptyset$

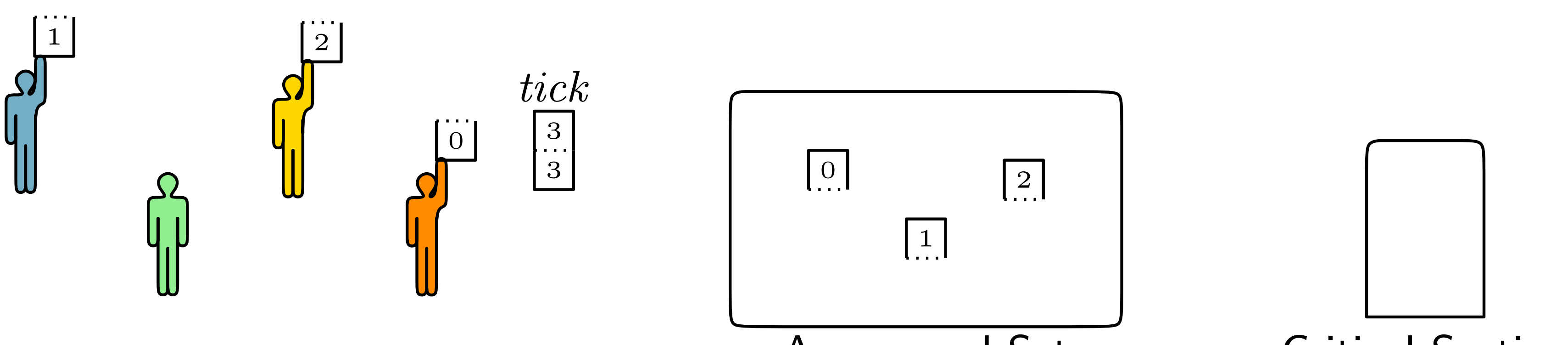
procedure $\textsc{MutExc}$
$\quad Int\ ticket$
begin
loop
$\quad$ **nondet**
$\quad \left\langle \begin{array}{l} ticket := tick++ \\ announced.add(ticket) \end{array} \right\rangle$
$\quad$ **await** $(announced.min == ticket)$
$\quad$ **critical**
$\quad announced.remove(ticket)$
end loop
end procedure



Motivating Example: Mutual Exclusion Algorithm

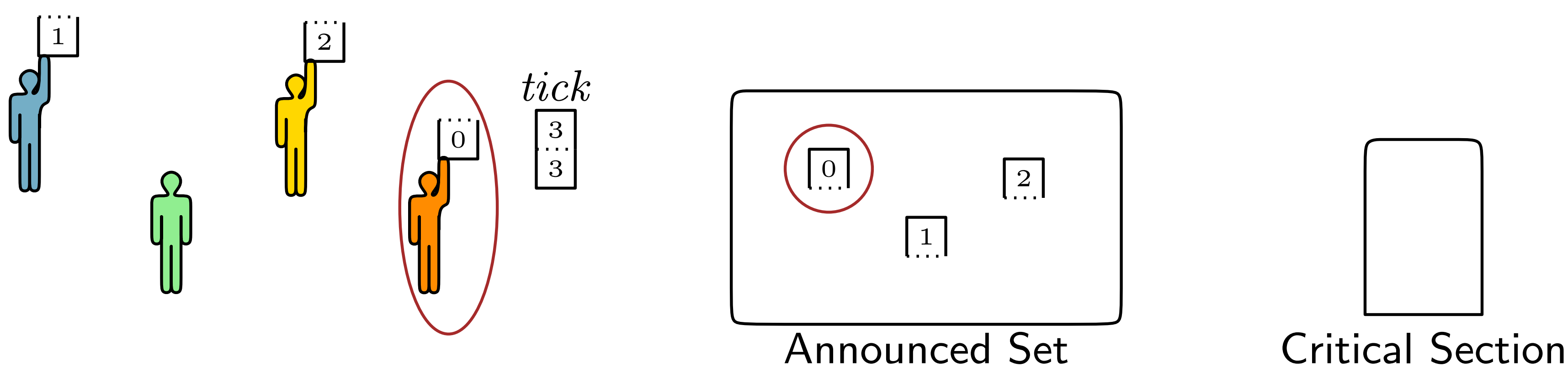
global
  $Int\ tick := 0$
  $Set\langle Int\rangle\ announced := \emptyset$

procedure MUTEXC
  $Int\ ticket$
begin
loop
  nondet
    $\left\langle \begin{array}{l} ticket := tick{+}{+} \\ announced.add(ticket) \end{array} \right\rangle$
  await $(announced.min == ticket)$
  critical
  $announced.remove(ticket)$
end loop
end procedure



Motivating Example: Mutual Exclusion Algorithm

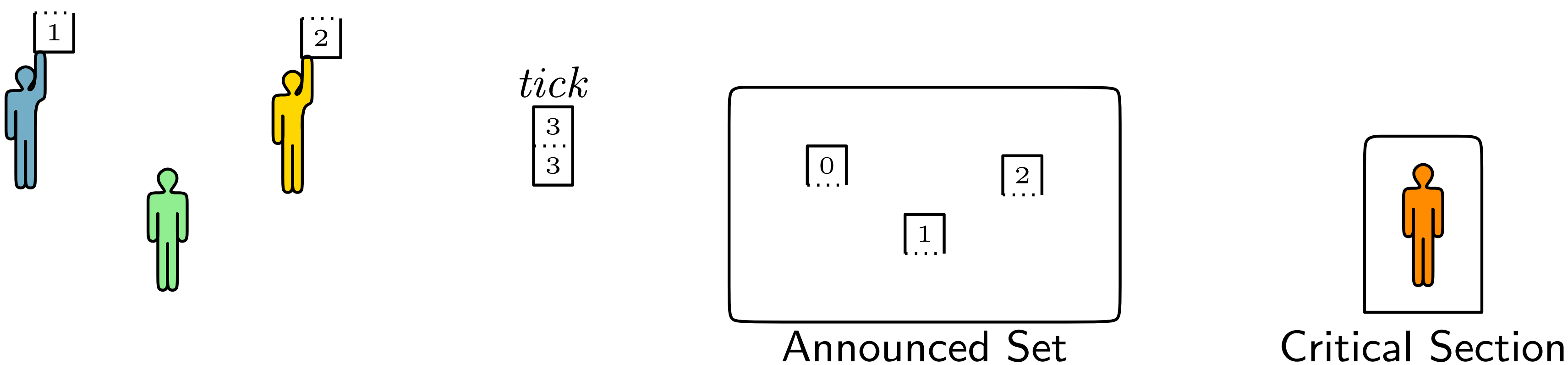
```
global
   Int tick := 0
   Set⟨Int⟩ announced := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := tick++
        announced.add(ticket) ⟩
      await (announced.min == ticket)
      critical
      announced.remove(ticket)
end loop
end procedure
```



Motivating Example: Mutual Exclusion Algorithm

global
  $Int\ tick := 0$
  $Set\langle Int\rangle\ announced := \emptyset$

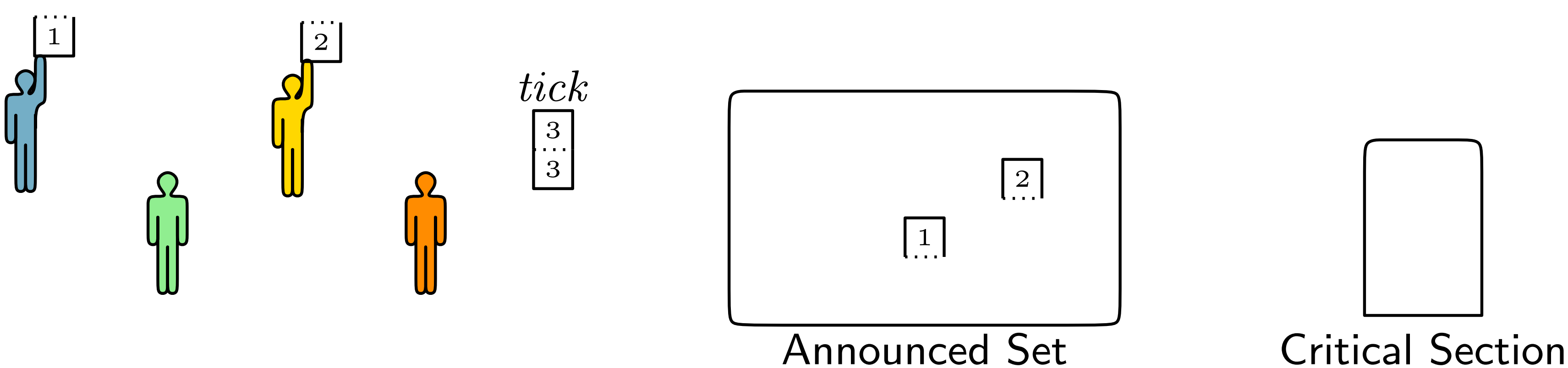
procedure MutExc
  $Int\ ticket$
begin
loop
  nondet
  $\left\langle \begin{array}{l} ticket := tick{+}{+} \\ announced.add(ticket) \end{array} \right\rangle$
  await $(announced.min == ticket)$
  critical
  $announced.remove(ticket)$
end loop
end procedure



Motivating Example: Mutual Exclusion Algorithm

global
  $\mathit{Int}\ \mathit{tick} := 0$
  $\mathit{Set}\langle \mathit{Int}\rangle\ \mathit{announced} := \emptyset$

procedure $\textsc{MutExc}$
  $\mathit{Int}\ \mathit{ticket}$
begin
loop
  nondet
    $\left\langle \begin{array}{l} \mathit{ticket} := \mathit{tick}{+}{+} \\ \mathit{announced}.\mathit{add}(\mathit{ticket}) \end{array} \right\rangle$
  await $(\mathit{announced}.\mathit{min} == \mathit{ticket})$
  critical
  $\mathit{announced}.\mathit{remove}(\mathit{ticket})$
end loop
end procedure



Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \to \Diamond access\_critical(k))$

Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \square(\mathit{announced}(k) \rightarrow \diamond\,\mathit{access\_critical}(k))$
- Let's assume a system with 2 threads: T_1 and T_2

Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$

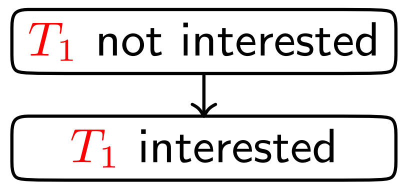
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$

T_1 not interested

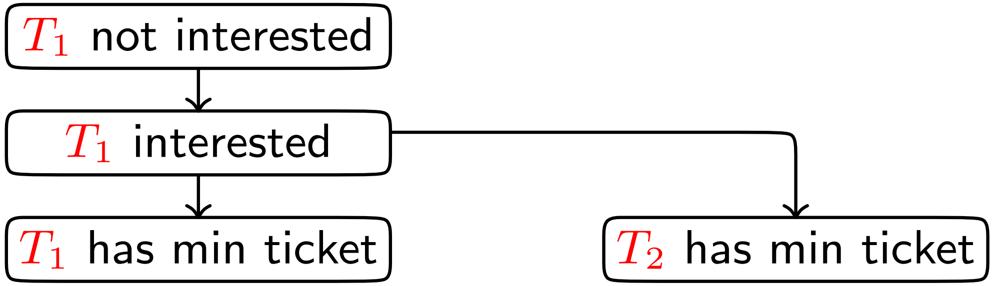
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



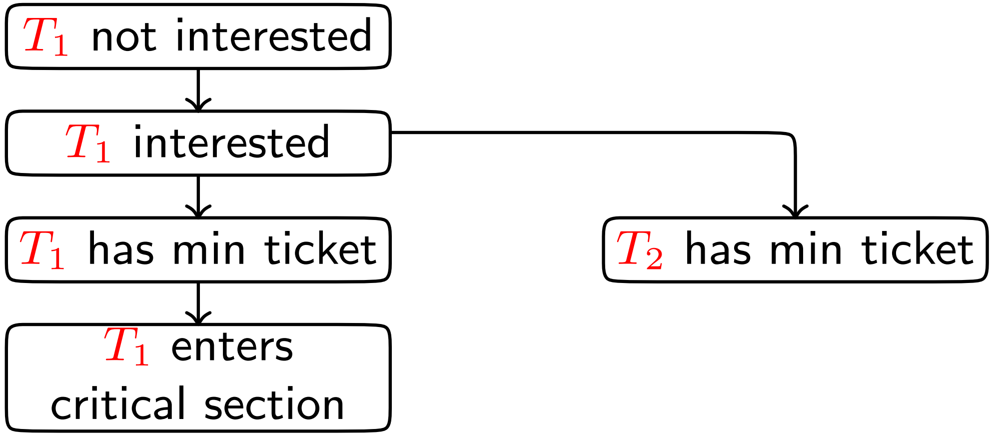
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \square(announced(k) \rightarrow \diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



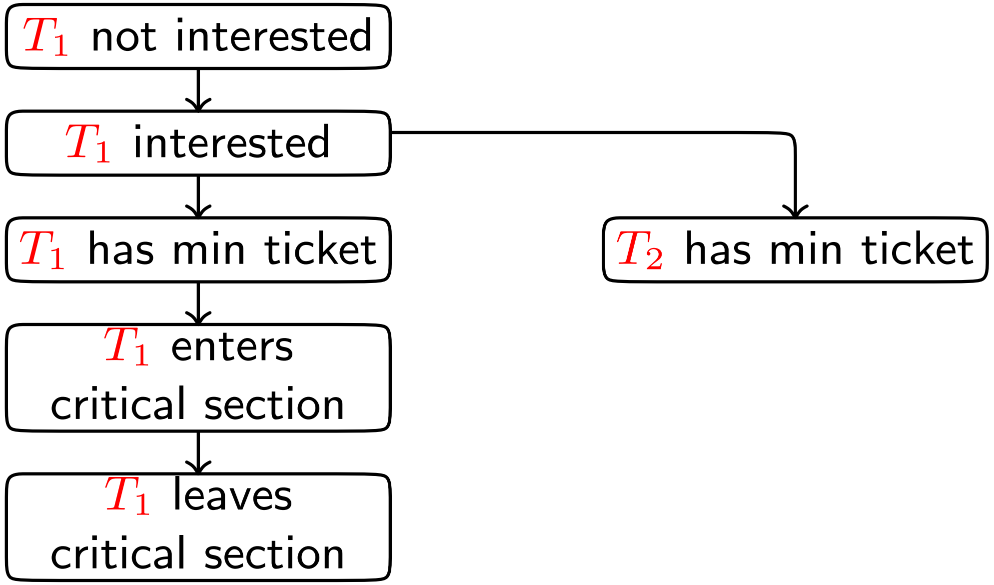
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \square(announced(k) \rightarrow \diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



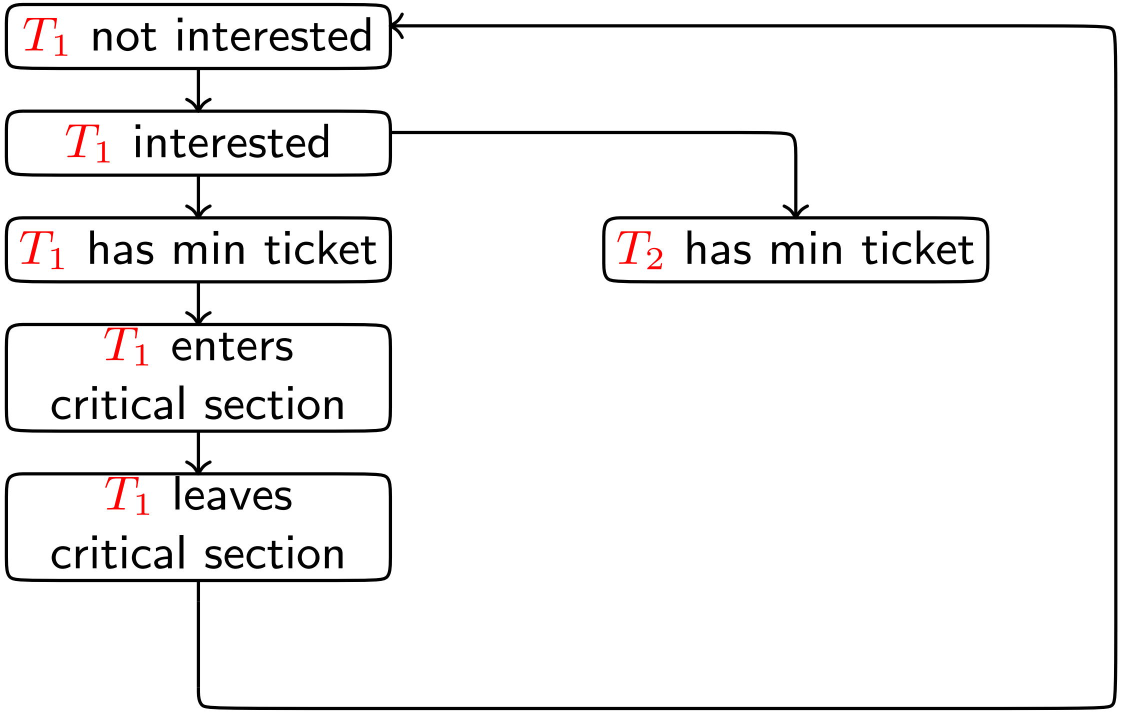
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



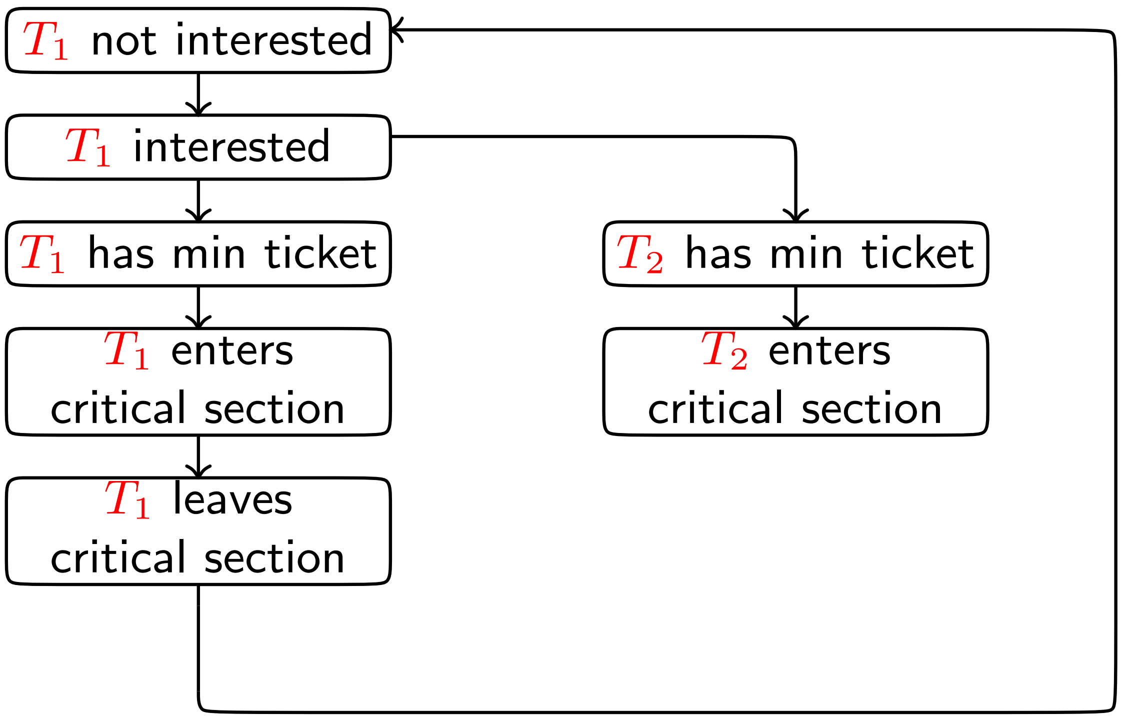
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \square(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



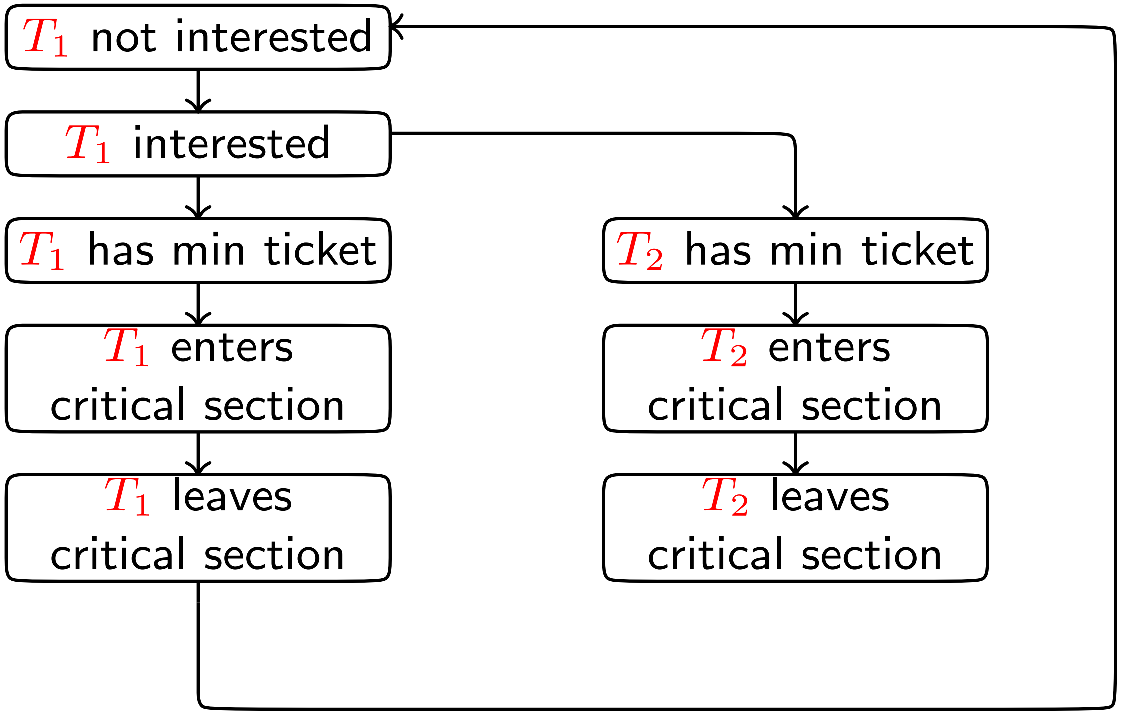
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



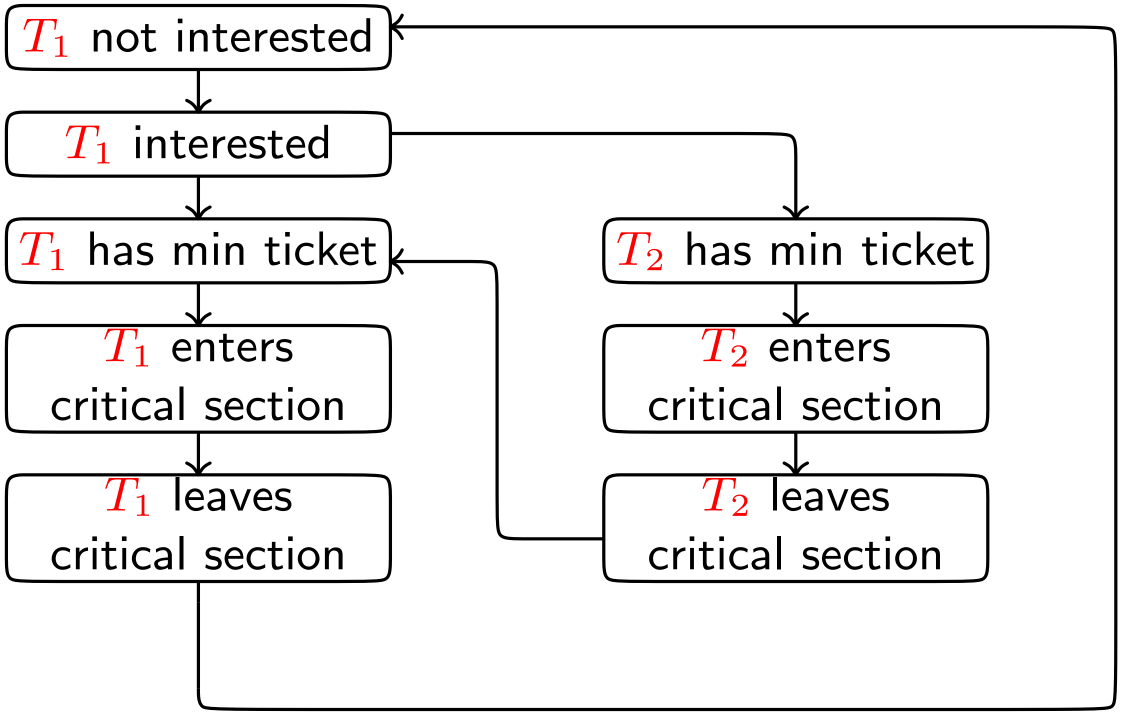
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \to \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



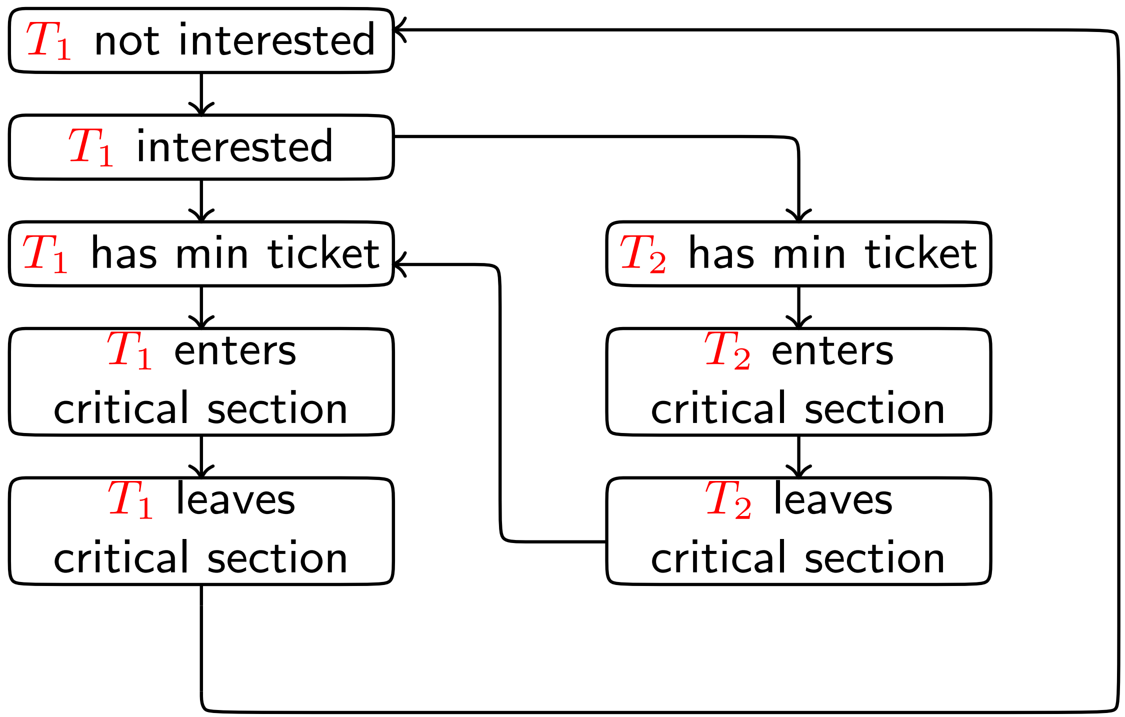
Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$



Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \to \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$

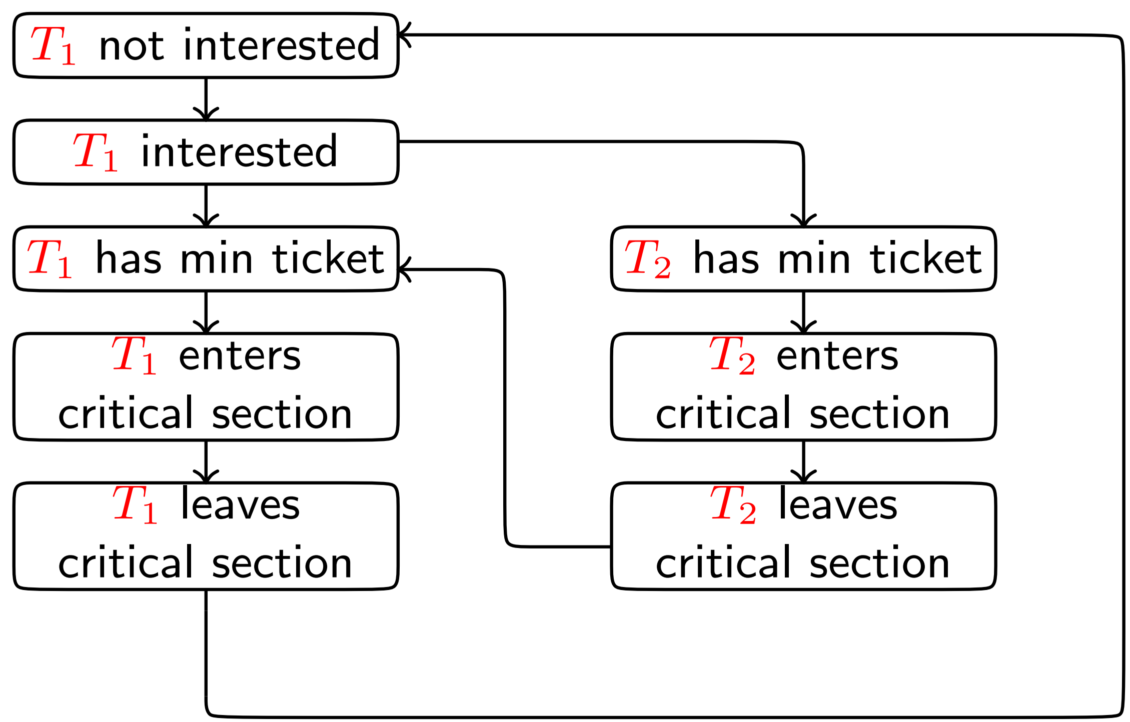


Some verification conditions

$$\left\{ \begin{array}{l} n \wedge \tau[1] \ \to \ succ(n) \\ n \wedge \tau[2] \ \to \ succ(n) \end{array} \right.$$

Verification Diagram for Mutual Exclusion Algorithm

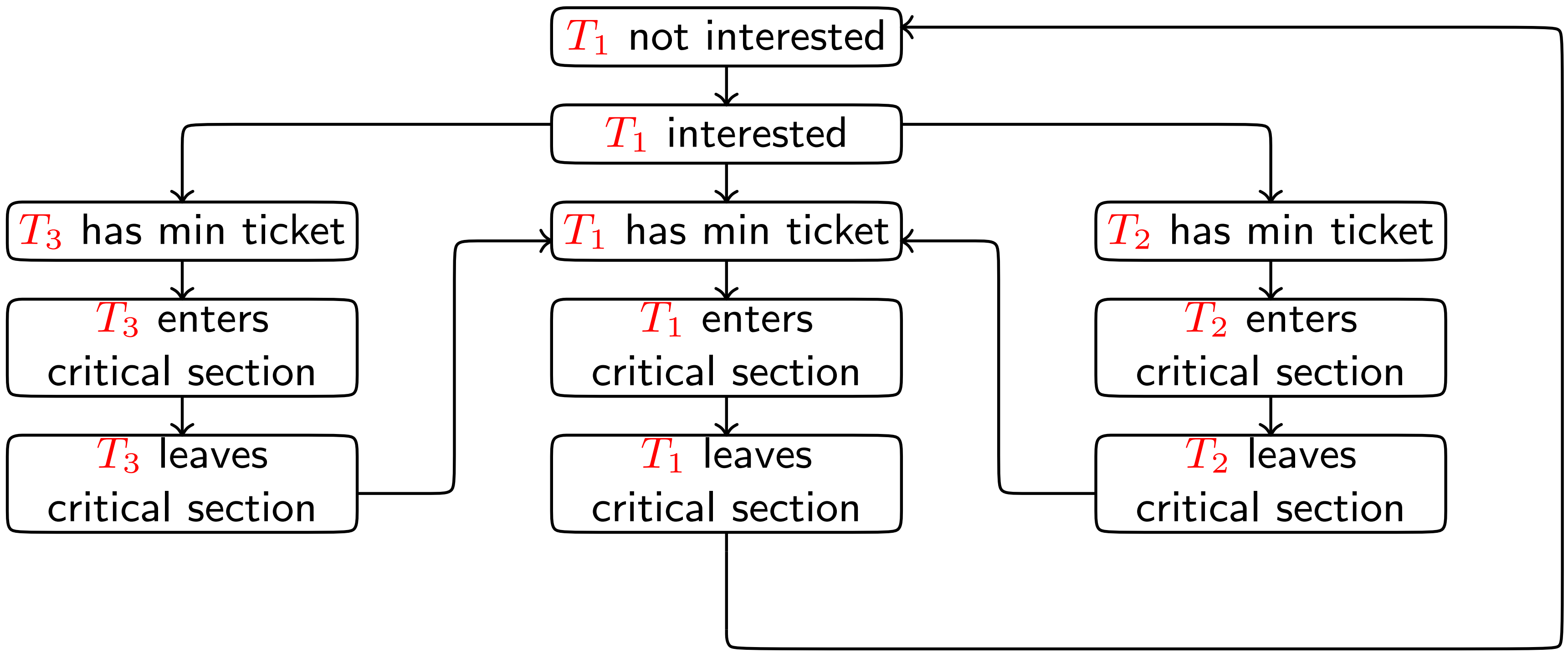
- For all k, $\varphi(k) \; = \; \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$
- Imagine now a system with 3 threads: T_1, T_2 and T_3



Some verification conditions
$$\left\{ \begin{array}{lcl} n \wedge \tau[1] & \rightarrow & succ(n) \\ n \wedge \tau[2] & \rightarrow & succ(n) \end{array} \right.$$

Verification Diagram for Mutual Exclusion Algorithm

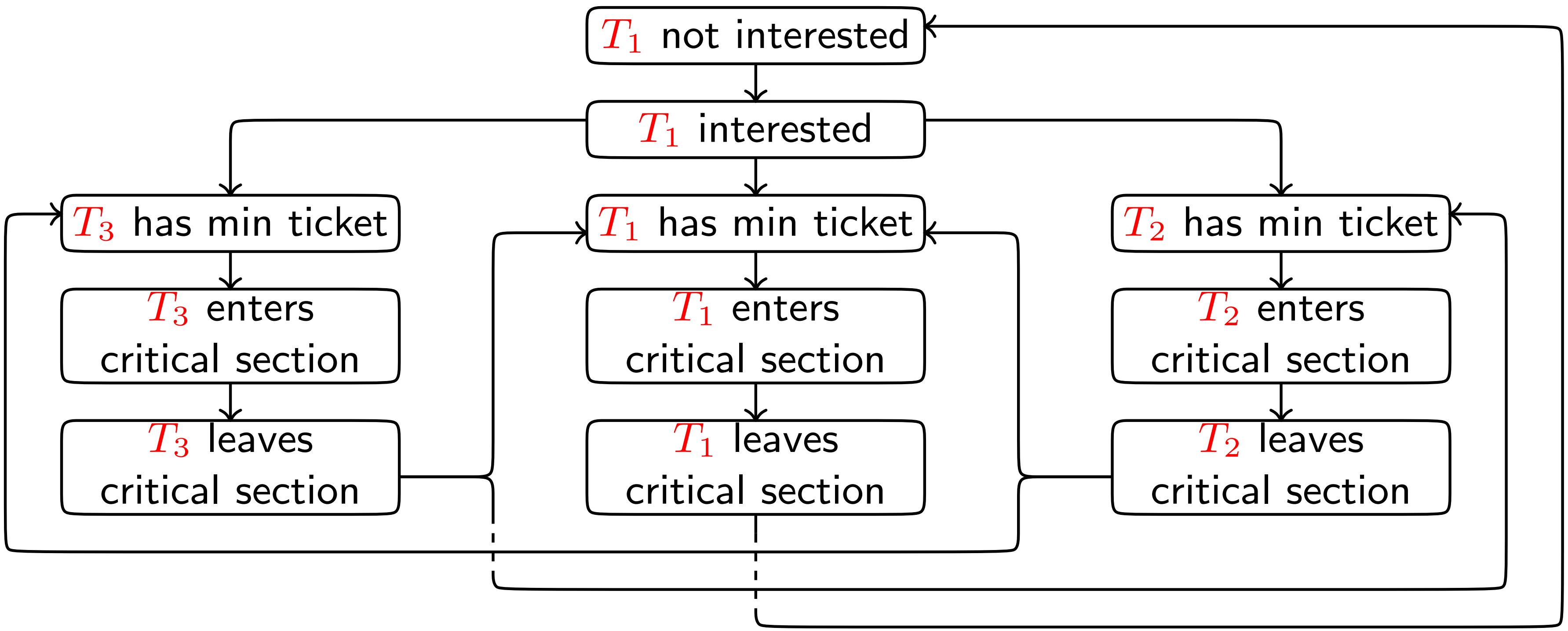
- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$
- Imagine now a system with 3 threads: T_1, T_2 and T_3



Some verification conditions
$$\begin{cases} n \wedge \tau[1] \rightarrow succ(n) \\ n \wedge \tau[2] \rightarrow succ(n) \end{cases}$$

Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$
- Imagine now a system with 3 threads: T_1, T_2 and T_3

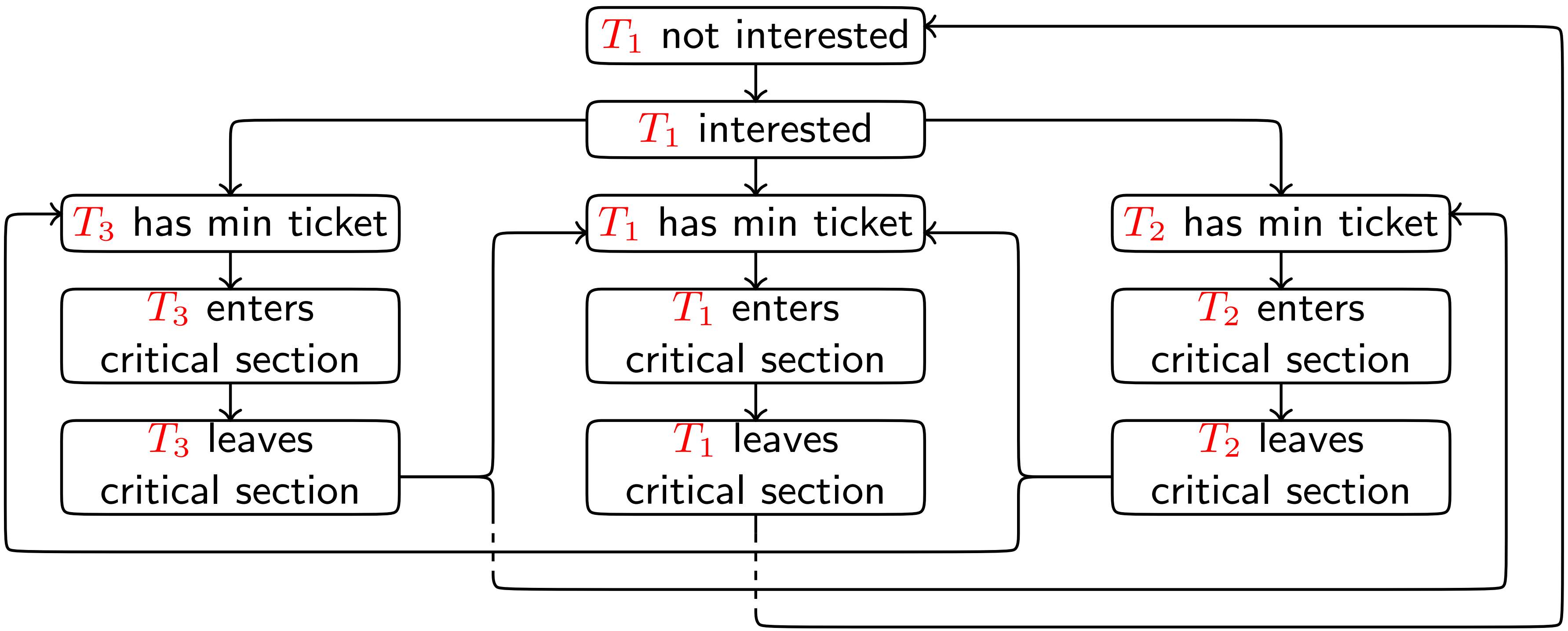


Some verification conditions

$$\left\{ \begin{array}{rcl} n \wedge \tau[1] & \rightarrow & succ(n) \\ n \wedge \tau[2] & \rightarrow & succ(n) \end{array} \right.$$

Verification Diagram for Mutual Exclusion Algorithm

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- Let's assume a system with 2 threads: T_1 and T_2
- Let's verify $\varphi(T_1)$
- Imagine now a system with 3 threads: T_1, T_2 and T_3



Some verification conditions
$$\left\{ \begin{array}{lcl} n \wedge \tau[1] & \rightarrow & succ(n) \\ n \wedge \tau[2] & \rightarrow & succ(n) \\ n \wedge \tau[3] & \rightarrow & succ(n) \end{array} \right.$$

Motivation for Parametrized Verification Diagrams

Problem

- **Not a single diagram** for arbitrary number of threads
- **Unbounded** number of **verification conditions**

Motivation for Parametrized Verification Diagrams

Problem

- **Not a single diagram** for arbitrary number of threads
- **Unbounded** number of **verification conditions**

Our solution

- **Unique diagram** for arbitrary number of threads
- **Finite and bounded** number of verification conditions

Parametrized Verification Diagrams exploits the similarities within **symetric systems**

Parametrized Fair Transition Systems

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P

$$\boxed{\mathcal{S}^{[M]} = \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle}$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P

$$\boxed{\mathcal{S}^{[M]} = \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle}$$

$$V = V_{\mathit{global}}$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

$$\boxed{\mathcal{S}^{[M]} = \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle}$$

$$V = V_{\mathit{global}} \cup (V_{\mathit{local}} \times [M])$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

$$\boxed{\mathcal{S}^{[M]} \;=\; \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle}$$

$$V \;=\; V_{\mathit{global}} \;\cup\; (V_{\mathit{local}} \times [M]) \;\cup\; pc[M]$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

$$\boxed{\mathcal{S}^{[M]} = \langle V, \Theta, \mathcal{T}, \mathcal{J} \rangle}$$

$$V = V_{global} \cup (V_{local} \times [M]) \cup pc[M]$$

$$\mathcal{T} = \bigcup_{l \in 1..L} \bigcup_{i \in [M]} \tau_l[i]$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

```
global
   Int tick := 0
   Set⟨Int⟩ announced := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := tick++
        announced.add(ticket) ⟩
      await (announced.min == ticket)
      critical
      announced.remove(ticket)
end loop
end procedure
```

For $\mathcal{S}^{[2]}$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

```
global
  Int tick := 0
  Set⟨Int⟩ announced := ∅

procedure MutExc
  Int ticket
begin
loop
    nondet
    ⟨ ticket := tick++
      announced.add(ticket) ⟩
    await (announced.min == ticket)
    critical
    announced.remove(ticket)
end loop
end procedure
```

For $\mathcal{S}^{[2]}$

$$V \quad = \quad \{tick, announced\}$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

global
$\quad Int\ tick := 0$
$\quad Set\langle Int\rangle\ announced := \emptyset$

procedure $\textsc{MutExc}$
$\quad \boxed{Int\ ticket}$
begin
loop
$\quad$ **nondet**
$\quad \left\langle \begin{array}{l} ticket := tick{+}{+} \\ announced.add(ticket) \end{array} \right\rangle$
$\quad$ **await** $(announced.min == ticket)$
$\quad$ **critical**
$\quad announced.remove(ticket)$
end loop
end procedure

For $\mathcal{S}^{[2]}$

$$V = \{tick, announced\} \cup \{ticket[1], ticket[2]\}$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

```
global
   Int tick := 0
   Set⟨Int⟩ announced := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := tick++
        announced.add(ticket) ⟩
      await (announced.min == ticket)
      critical
      announced.remove(ticket)
end loop
end procedure
```

For $\mathcal{S}^{[2]}$

$$\begin{aligned} V \quad = \quad & \{tick, announced\} \quad \cup \\ & \{ticket[1], ticket[2]\} \quad \cup \\ & \{pc[1], pc[2]\} \end{aligned}$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

global
  $Int\ tick := 0$
  $Set\langle Int\rangle\ announced := \emptyset$

procedure MUTEXC
  $Int\ ticket$
begin
1: **loop**
2:   **nondet**
3:   $\left\langle \begin{array}{l} ticket := tick{+}{+} \\ announced.add(ticket) \end{array} \right\rangle$
4:   **await** $(announced.min == ticket)$
5:   **critical**
6:   $announced.remove(ticket)$
7: **end loop**
end procedure

For $\mathcal{S}^{[2]}$

$$\begin{array}{lcll} V & = & \{tick, announced\} & \cup \\ & & \{ticket[1], ticket[2]\} & \cup \\ & & \{pc[1], pc[2]\} & \end{array}$$

$$\mathcal{T} = \bigcup_{l \in 1..7} \{\tau_l[1], \tau_l[2]\}$$

Parametrized Fair Transition Systems

- Let P be a program consisting of L **lines of code**
- Assuming M **threads** running program P
- Let V_{global} be the set of **global variables** of program P
- Let V_{local} be the set of **local variables** of program P

```
global
    Int tick := 0
    Set⟨Int⟩ announced := ∅

procedure MutExc
    Int ticket
begin
1:  loop
2:      nondet
3:      ⟨ ticket := tick++
          announced.add(ticket) ⟩
4:      await (announced.min == ticket)
5:      critical
6:      announced.remove(ticket)
7:  end loop
end procedure
```

For $\mathcal{S}^{[2]}$

$$
\begin{aligned}
V &= \{tick, announced\} \cup \\
&\quad \{ticket[1], ticket[2]\} \cup \\
&\quad \{pc[1], pc[2]\} \\
\mathcal{T} &= \bigcup_{l \in 1..7} \{\tau_l[1], \tau_l[2]\} \\
\mathcal{J} &= \mathcal{T}
\end{aligned}
$$

Symmetric Systems

Symmetric Systems

- Assume a parametrized transition system $\mathcal{S}^{[M]}$
- All threads execute the **same program**
- Only **equality and inequality** between **thread identifiers**

Symmetric Systems

- Assume a parametrized transition system $\mathcal{S}^{[M]}$
- All threads execute the **same program**
- Only **equality and inequality** between **thread identifiers**

$$\sigma \quad = \quad s_0 \qquad s_1 \qquad s_2 \qquad \ldots$$

Symmetric Systems

- Assume a parametrized transition system $\mathcal{S}^{[M]}$
- All threads execute the **same program**
- Only **equality and inequality** between **thread identifiers**

$$\sigma \quad = \quad s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} \dots$$

Symmetric Systems

- Assume a parametrized transition system $\mathcal{S}^{[M]}$
- All threads execute the **same program**
- Only **equality and inequality** between **thread identifiers**

$$\sigma \quad = \quad s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} \dots$$

$i, j \in [M]$

Symmetric Systems

- Assume a parametrized transition system $\mathcal{S}^{[M]}$
- All threads execute the **same program**
- Only **equality and inequality** between **thread identifiers**

$$
\begin{tikzcd}
\sigma = & s_0 \arrow[r, "\tau_0"] \arrow[d, dashed] & s_1 \arrow[r, "\tau_1"] \arrow[d, dashed] & s_2 \arrow[r, "\tau_2"] \arrow[d, dashed] & \dots \\
& s_0^{i \leftrightarrow j} & s_1^{i \leftrightarrow j} & s_2^{i \leftrightarrow j} & \dots
\end{tikzcd}
$$

$i, j \in [M]$ | switch $v[i]$ with $v[j]$

Symmetric Systems

- Assume a parametrized transition system $\mathcal{S}^{[M]}$
- All threads execute the **same program**
- Only **equality and inequality** between **thread identifiers**

$$\begin{tikzcd}
\sigma \arrow[r, phantom, "="] & s_0 \arrow[r, "\tau_0"] \arrow[d, dashed] & s_1 \arrow[r, "\tau_1"] \arrow[d, dashed] & s_2 \arrow[r, "\tau_2"] \arrow[d, dashed] & \ldots \\
 & s_0^{i\leftrightarrow j} \arrow[r, "\tau_0^{i\leftrightarrow j}"] & s_1^{i\leftrightarrow j} \arrow[r, "\tau_1^{i\leftrightarrow j}"] & s_2^{i\leftrightarrow j} \arrow[r, "\tau_2^{i\leftrightarrow j}"] & \ldots
\end{tikzcd}$$

$i, j \in [M]$

switch $v[i]$ with $v[j]$

switch $\tau[i]$ with $\tau[j]$

Symmetric Systems

- Assume a parametrized transition system $\mathcal{S}^{[M]}$
- All threads execute the **same program**
- Only **equality and inequality** between **thread identifiers**

$$\begin{array}{ccccccccc}
\sigma & = & s_0 & \xrightarrow{\tau_0} & s_1 & \xrightarrow{\tau_1} & s_2 & \xrightarrow{\tau_2} & \dots \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
\sigma^{i\leftrightarrow j} & = & s_0^{i\leftrightarrow j} & \xrightarrow{\tau_0^{i\leftrightarrow j}} & s_1^{i\leftrightarrow j} & \xrightarrow{\tau_1^{i\leftrightarrow j}} & s_2^{i\leftrightarrow j} & \xrightarrow{\tau_2^{i\leftrightarrow j}} & \dots
\end{array}$$

$i, j \in [M]$	switch $v[i]$ with $v[j]$ switch $\tau[i]$ with $\tau[j]$

Parametrized Verification Diagrams

- Prove that **all instances** of $\mathcal{S}^{[M]}$ satisfy a temporal specification

  with a unique diagram

Parametrized Verification Diagrams

- Prove that **all instances** of $\mathcal{S}^{[M]}$ satisfy a temporal specification

  with a unique diagram

- We add $\Sigma_{\textbf{tid}} = (\{\textbf{tid}\}, \emptyset, \emptyset)$ and $T_{\textbf{param}}$ of uninterpreted functions

Parametrized Verification Diagrams

- Prove that **all instances** of $\mathcal{S}^{[M]}$ satisfy a temporal specification

  with a unique diagram

- We add $\Sigma_{\textbf{tid}} = (\{\textbf{tid}\}, \emptyset, \emptyset)$ and $T_{\textbf{param}}$ of uninterpreted functions
- For each $v : \alpha$, we add $f_v : \mathsf{tid} \rightarrow \alpha$

  $f_{pc} : \mathsf{tid} \rightarrow Loc$

- T_{param} is:
  - stable infinite
  - polite

Parametrized Verification Diagrams

- Prove that **all instances** of $\mathcal{S}^{[M]}$ satisfy a temporal specification

  with a unique diagram

- We add $\Sigma_{\textbf{tid}} = (\{\textbf{tid}\}, \emptyset, \emptyset)$ and $T_{\textbf{param}}$ of uninterpreted functions

- For each $v : \alpha$, we add $f_v : \mathsf{tid} \rightarrow \alpha$

  $$f_{pc} : \mathsf{tid} \rightarrow Loc$$

- T_{param} is:
  - stable infinite $\xrightarrow{\textbf{combinable with}}$ stable infinite theories
  - polite $\longrightarrow$ non stable infinite theories

$$\boxed{T = T_{\mathsf{prog}} + T_{\mathsf{param}}}$$

Soundness of Parametrized Verification Diagrams

Theorem:

Let $\mathcal{S}^{[M]}$ be a symmetric parametrized FTS and $\varphi(k)$ a temporal formula.

If there exists a (M,k)-valid PVD $\mathcal{D}^{[M]}$, then:

$$\boxed{\mathcal{S}^{[M]} \qquad \models \qquad \varphi(k)}$$

Soundness of Parametrized Verification Diagrams

Theorem:

Let $\mathcal{S}^{[M]}$ be a symmetric parametrized FTS and $\varphi(k)$ a temporal formula.

If there exists a (M, k)-valid PVD $\mathcal{D}^{[M]}$, then:

$$\boxed{\mathcal{S}^{[M]} \ \models\ \mathcal{D}^{[\mathcal{M}]} \ \models\ \varphi(k)}$$

for all M

Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

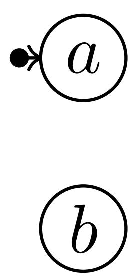
a

b

Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

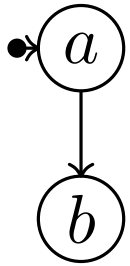
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

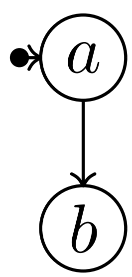
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

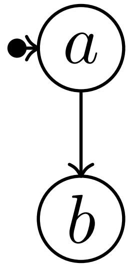
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

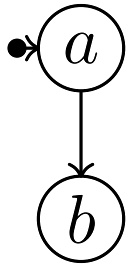
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

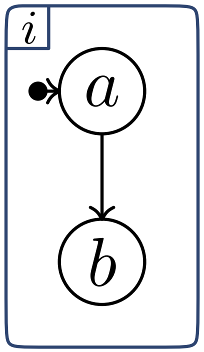
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

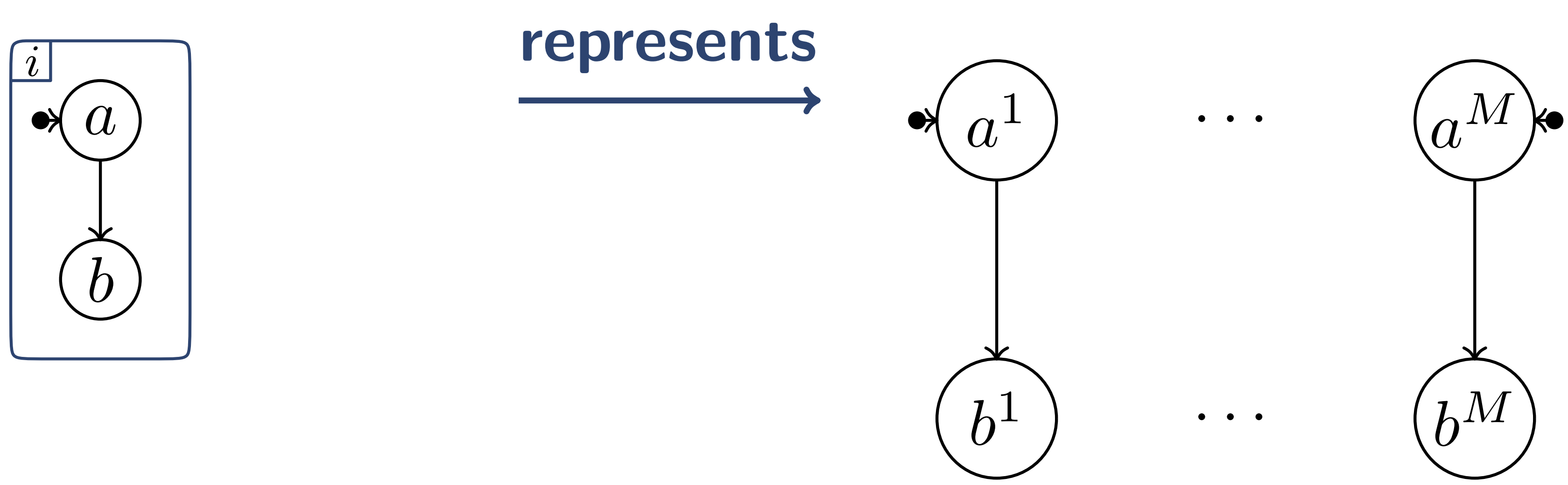


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

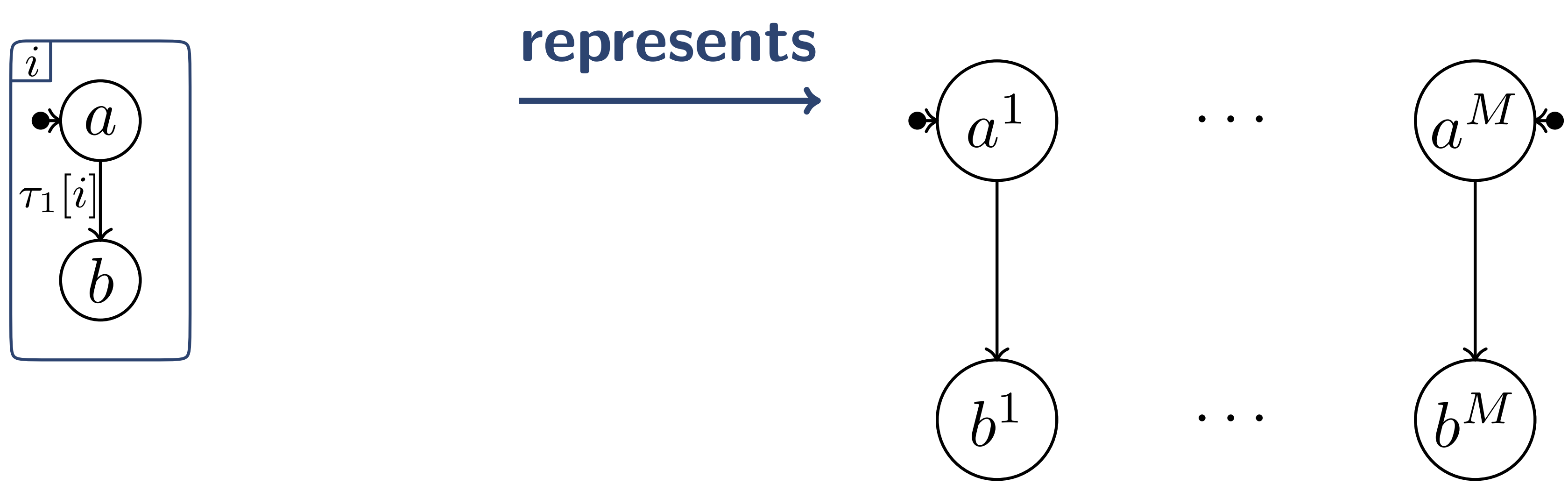


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

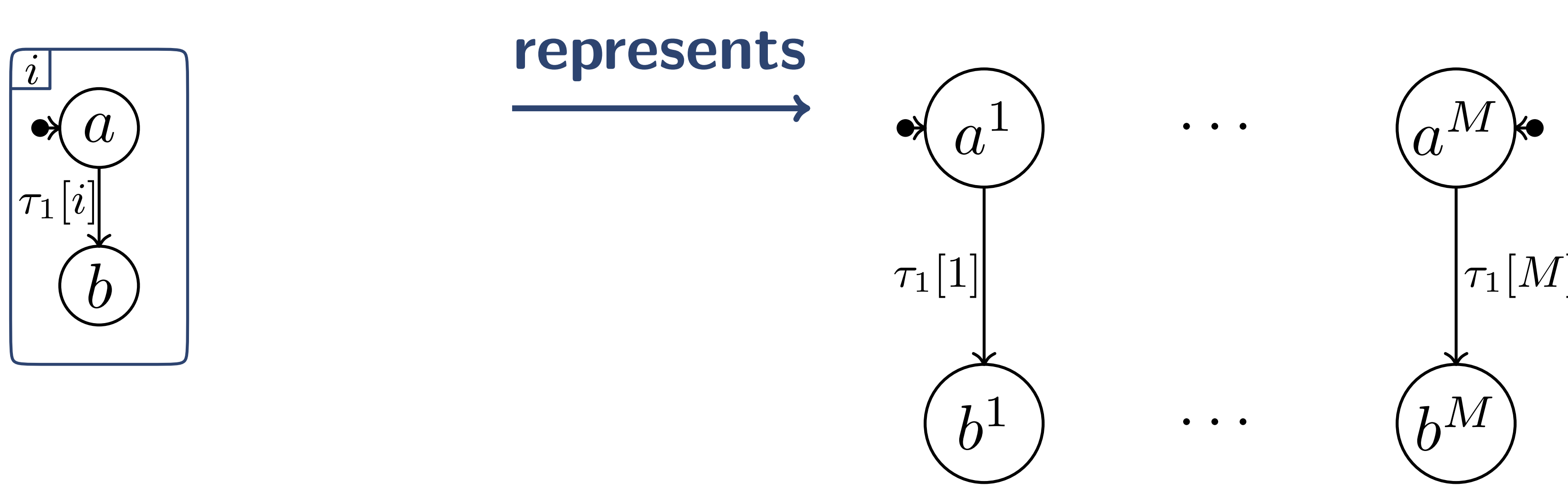


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

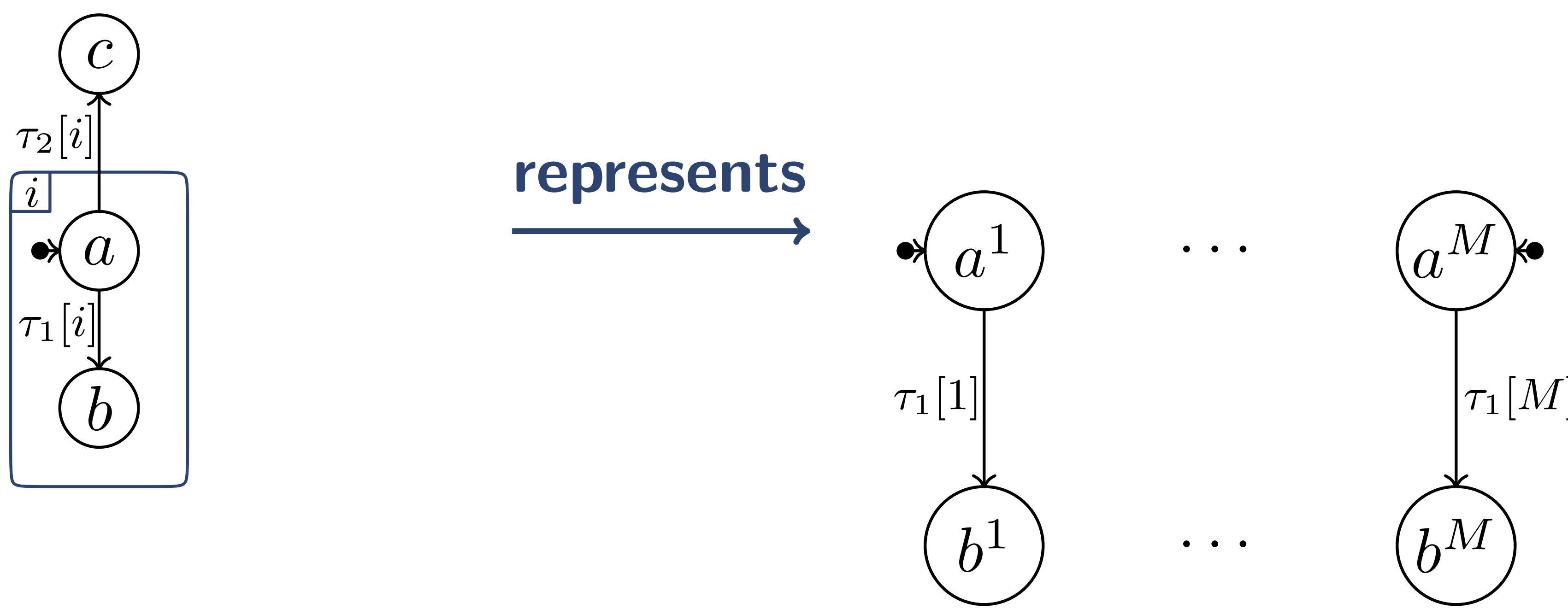


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

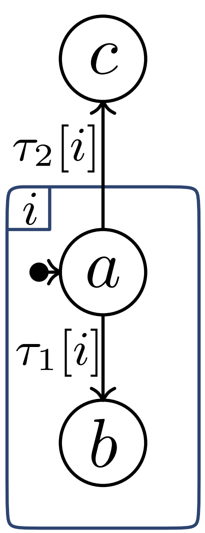


Parametrized Verification Diagrams

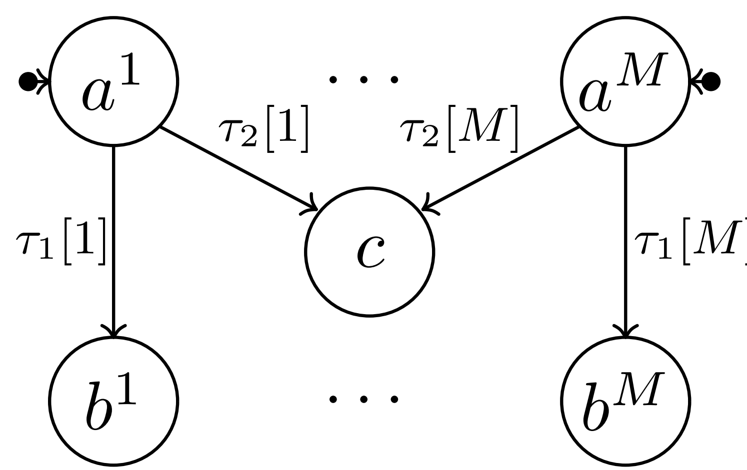
- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads



represents $\longrightarrow$

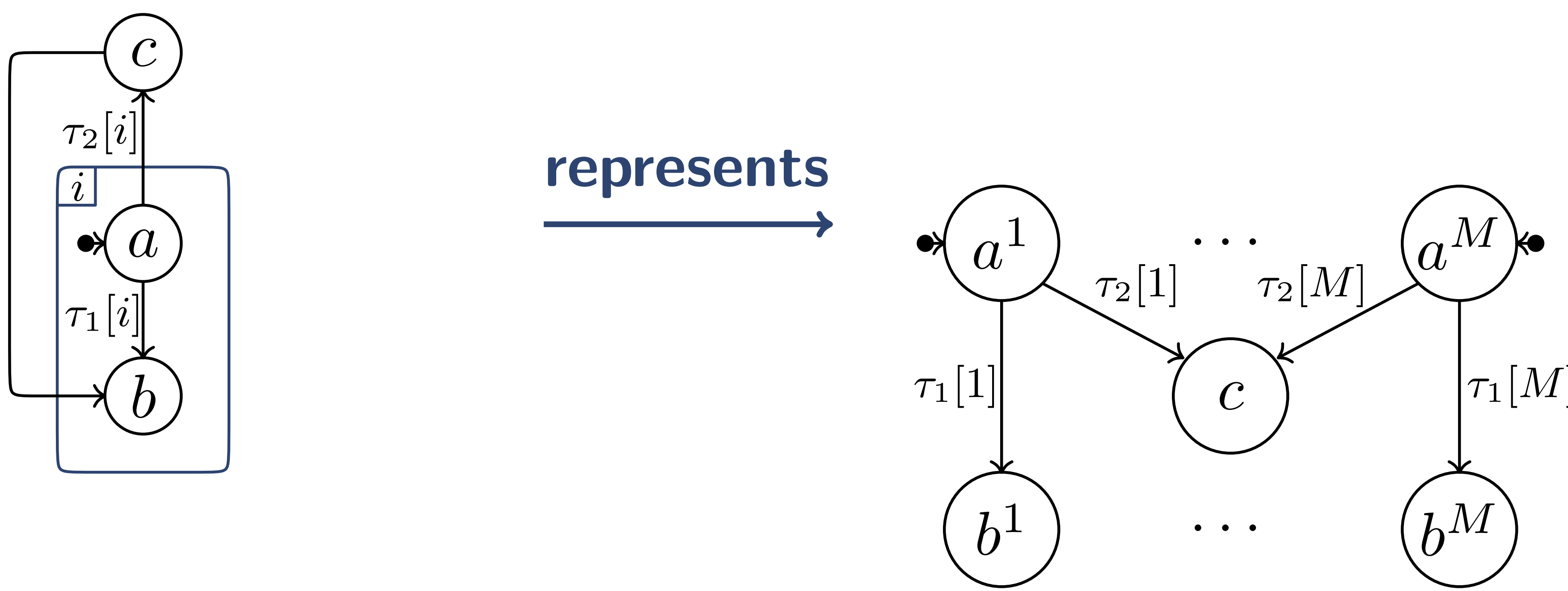


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

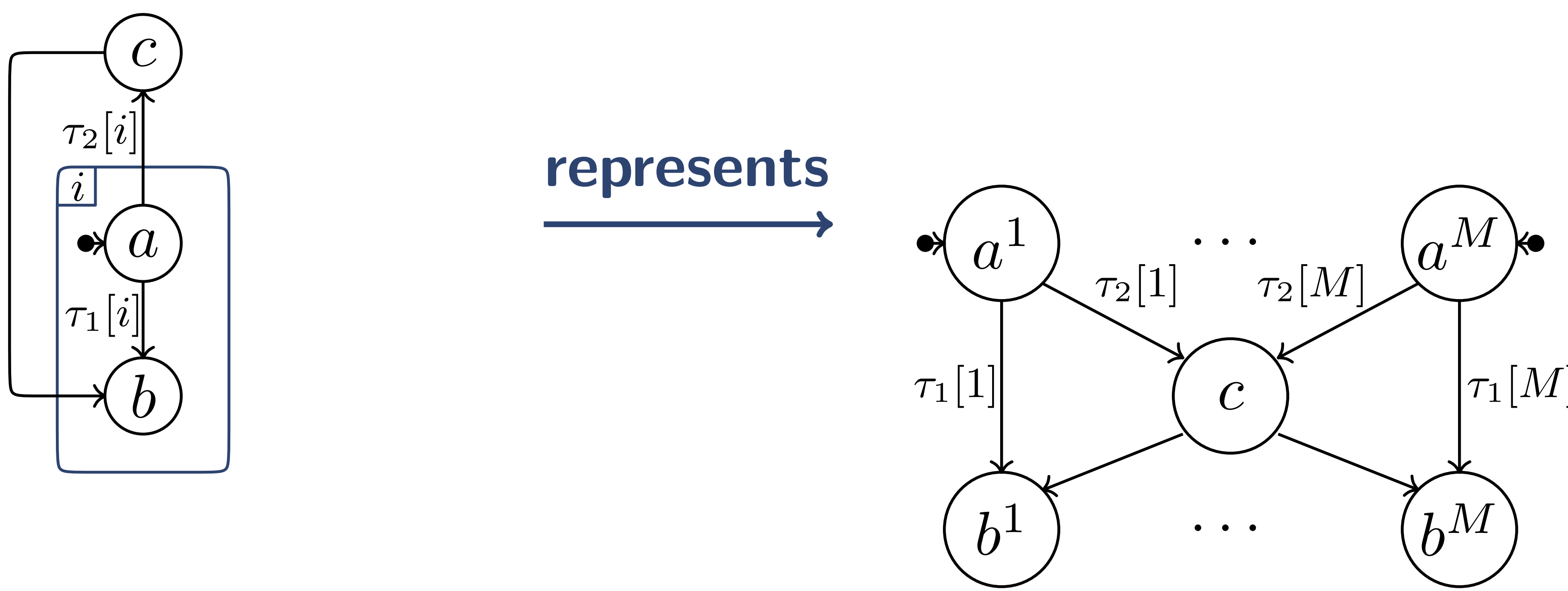


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

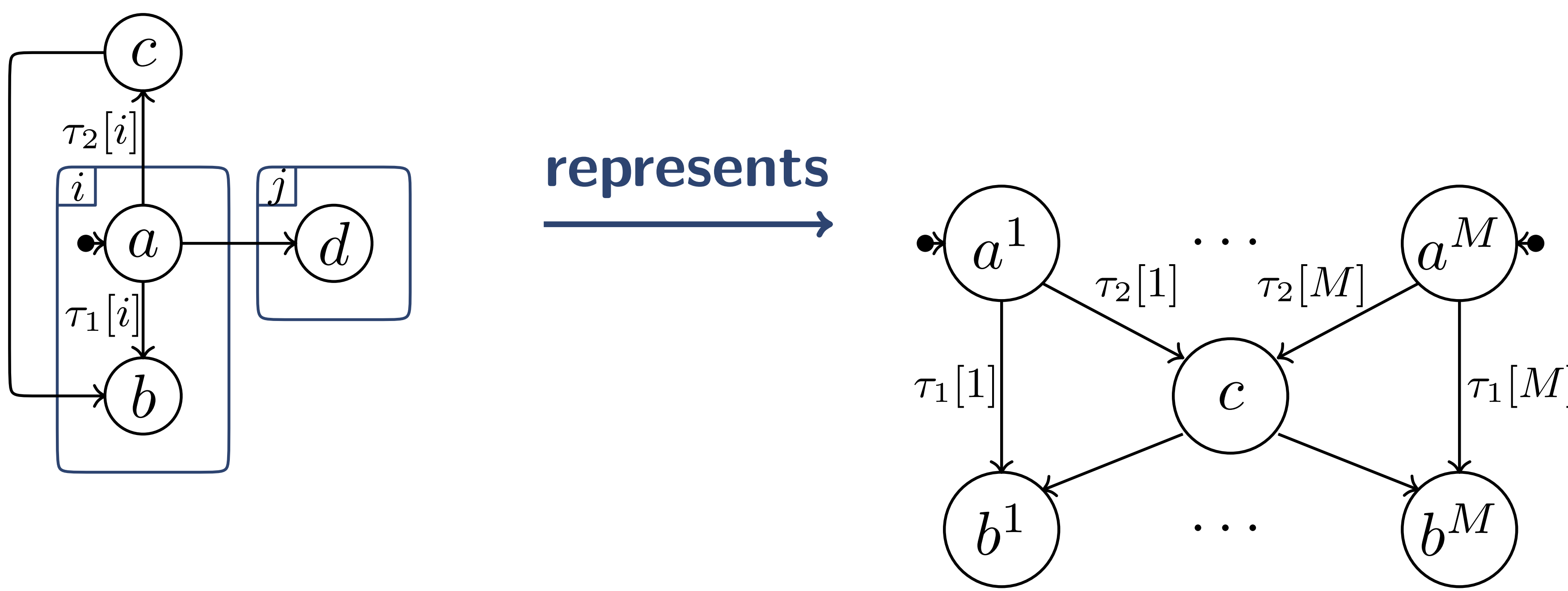


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

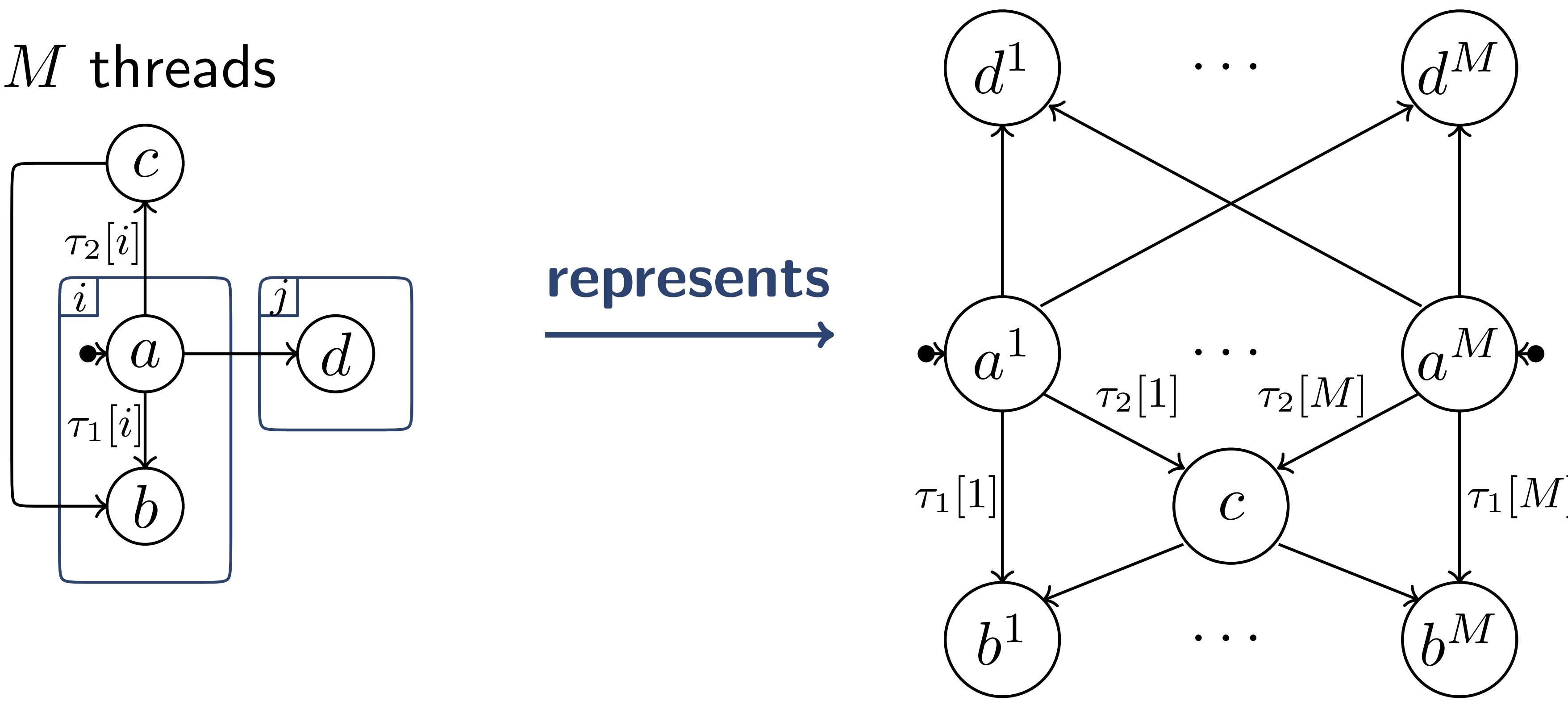


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads

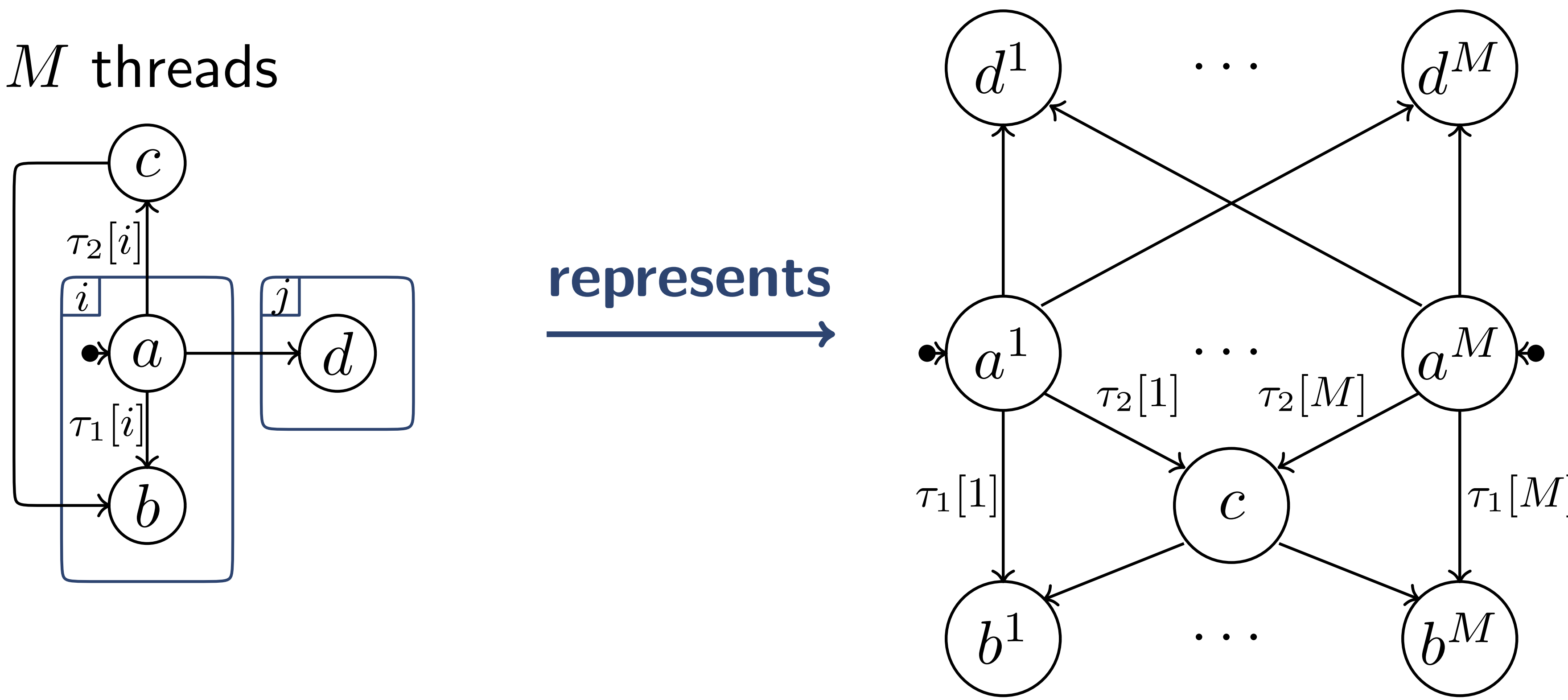


Parametrized Verification Diagrams

- PVDs are **an extension** of GVD
- We add the notion of **boxes**
- A PVD **abstracts all instantiations** of a parametric system

$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$

For M threads



Verification Conditions for Parametrized Diagrams

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\boxed{\quad n \wedge \tau \ \rightarrow \ succ(n) \quad}$$

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\boxed{n \wedge \tau \;\rightarrow\; succ(n)}$$

$$\bigwedge_l \bigwedge_i n \wedge \tau_l[i] \;\rightarrow\; succ(n)$$

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\frac{\boxed{n \wedge \tau \;\rightarrow\; succ(n)}}{\bigwedge_l \bigwedge_i \underbrace{n \wedge \tau_l[i] \;\rightarrow\; succ(n)}}$$

only tid appearing in n and $succ(n)$ are relevant

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\boxed{n \wedge \tau \;\rightarrow\; succ(n)}$$

$$\bigwedge_l \bigwedge_i \; n \wedge \tau_l[i] \;\rightarrow\; succ(n)$$

$$Voc\big(n, succ(n)\big) \;=\; \{i_1, \ldots, i_q\} \;=\; I$$

tid appearing on n and $succ(n)$

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\boxed{\quad n \wedge \tau \;\rightarrow\; succ(n) \quad}$$

$$\bigwedge_l \bigwedge_i n \wedge \tau_l[i] \;\rightarrow\; succ(n)$$

$$Voc\big(n, succ(n)\big) \;=\; \{i_1, \ldots, i_q\} \;=\; I$$

$$\begin{array}{lll} \bigwedge_l & n \wedge \tau_l[i_1] & \rightarrow succ(n) \\ & \quad\vdots & \quad\vdots \\ \bigwedge_l & n \wedge \tau_l[i_q] & \rightarrow succ(n) \end{array}$$

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \to \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\boxed{n \wedge \tau \;\to\; succ(n)}$$

$$\bigwedge_l \bigwedge_i \; n \wedge \tau_l[i] \;\to\; succ(n)$$

$$Voc\big(n, succ(n)\big) \;=\boxed{\{i_1, \ldots, i_q\}} = \; I$$

$$\boxed{\begin{array}{lll} \bigwedge_l & n \wedge \tau_l[i_1] & \to succ(n) \\ & \vdots & \vdots \\ \bigwedge_l & n \wedge \tau_l[i_q] & \to succ(n) \end{array}}$$

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \to \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\boxed{n \wedge \tau \;\to\; succ(n)}$$

$$\bigwedge_l \bigwedge_i \; n \wedge \tau_l[i] \;\to\; succ(n)$$

$$Voc(n, succ(n)) \;=\; \{i_1, \ldots, i_q\} \;=\; I$$

$$\begin{array}{llr}
\bigwedge_l & n \wedge \tau_l[i_1] & \to succ(n) \\
 & \vdots & \vdots \\
\bigwedge_l & n \wedge \tau_l[i_q] & \to succ(n) \\
\bigwedge_l & n \wedge \tau_l[i] \quad \wedge \quad \bigwedge_{j \in I} i \neq j & \to succ(n)
\end{array}$$

abstracts all other cases thanks to **symmetry**

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\boxed{n \wedge \tau \;\rightarrow\; succ(n)}$$

$$\bigwedge_l \bigwedge_i n \wedge \tau_l[i] \;\rightarrow\; succ(n)$$

$$Voc(n, succ(n)) \;=\; \{i_1, \ldots, i_q\} \;=\; I$$

$$\boxed{\begin{array}{llll}
\bigwedge_l & n \wedge \tau_l[i_1] & & \rightarrow succ(n) \\
& \vdots & & \vdots \\
\bigwedge_l & n \wedge \tau_l[i_q] & & \rightarrow succ(n) \\
\bigwedge_l & n \wedge \tau_l[i] & \wedge \quad \bigwedge_{j \in I} i \neq j & \rightarrow succ(n)
\end{array}}$$

unbounded

$\mathbf{L} \times \mathbf{M}$

Before

bounded

$\mathbf{L} \times (\mathbf{q} + \mathbf{1})$

Now

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \rightarrow \mu(N_0)$
- **Consecution:** For every $n \in N$, let $I = Voc(n, next(n))$,

$$\begin{array}{lllll} \text{(C1)} & \mu(n)(s) & \wedge\ \rho_{\tau[i]}(s,s') & & \rightarrow \mu(next(n))(s') \quad \text{, for each } i \in I \\ \text{(C2)} & \mu(n)(s) & \wedge\ \rho_{\tau[i]}(s,s') & \wedge \quad \bigwedge_{j \in I} i \neq j & \rightarrow \mu(next(n))(s') \end{array}$$

- **Acceptance:** If $(n_1, n_2) \in P \setminus R$, let $I = Voc(n_1, n_2)$,

$$\text{(a)} \quad \left[\begin{array}{l} \rho_{\tau[i]}(s,s') \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array} \right] \rightarrow \delta_{j,n_1}(s) \succeq \delta_{j,n_2}(s') \quad \text{, for each } i \in I$$

$$\text{(b)} \quad \left[\begin{array}{l} \rho_{\tau[i]}(s,s') \wedge \bigwedge_{j \in I} i \neq j \quad \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array} \right] \rightarrow \delta_{j,n_1}(s) \succeq \delta_{j,n_2}(s')$$

and if $(n_1, n_2) \notin P \cup R$,

$$\text{(a)} \quad \left[\begin{array}{l} \rho_{\tau[i]}(s,s') \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array} \right] \rightarrow \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s') \quad \text{, for each } i \in I$$

$$\text{(b)} \quad \left[\begin{array}{l} \rho_{\tau[i]}(s,s') \wedge \bigwedge_{j \in I} i \neq j \quad \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array} \right] \rightarrow \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s')$$

- **Fairness:** For each $e = (n_1, n_2) \in E$ and $i \in \beta_v(n_1)$:

$$\begin{array}{lll} \text{(F1)} & \mu(n_1)(s) \wedge \tau[i] \in \eta(e) & \rightarrow En(\tau[i]) \\ \text{(F2)} & \mu(n_1)(s) \wedge \tau[i] \in \eta(e) \wedge \rho_{\tau[i]}(s,s') & \rightarrow \mu(\tau[i](n_1))(s') \end{array}$$

Verification Conditions for Parametrized Diagrams

- **Initialization:** $\Theta \to \mu(N_0)$
- **Consecution:** For every $n \in N$, let $I = Voc(n, next(n))$,

  (C1) $\mu(n)(s) \quad \wedge \rho_{\tau[i]}(s, s') \quad \to \mu(next(n))(s')$, for each $i \in I$
  (C2) $\mu(n)(s) \quad \wedge \rho_{\tau[i]}(s, s') \quad \wedge \quad \bigwedge_{j \in I} i \neq j \quad \to \mu(next(n))(s')$

- **Acceptance:** If $(n_1, n_2) \in P \setminus R$, let $I = Voc(n_1, n_2)$,

  (a) $\left[\begin{array}{l} \rho_{\tau[i]}(s, s') \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array}\right] \to \delta_{j,n_1}(s) \succeq \delta_{j,n_2}(s')$, for each $i \in I$

  (b) $\left[\begin{array}{l} \rho_{\tau[i]}(s, s') \wedge \bigwedge_{j \in I} i \neq j \quad \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array}\right] \to \delta_{j,n_1}(s) \succeq \delta_{j,n_2}(s')$

  and if $(n_1, n_2) \notin P \cup R$,

  (a) $\left[\begin{array}{l} \rho_{\tau[i]}(s, s') \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array}\right] \to \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s')$, for each $i \in I$

  (b) $\left[\begin{array}{l} \rho_{\tau[i]}(s, s') \wedge \bigwedge_{j \in I} i \neq j \quad \wedge \\ \mu(n_1)(s) \wedge \mu(n_2)(s') \end{array}\right] \to \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s')$

- **Fairness:** For each $e = (n_1, n_2) \in E$ and $i \in \beta_v(n_1)$:

  (F1) $\mu(n_1)(s) \wedge \tau[i] \in \eta(e) \to En(\tau[i])$
  (F2) $\mu(n_1)(s) \wedge \tau[i] \in \eta(e) \wedge \rho_{\tau[i]}(s, s') \to \mu(\tau[i](n_1))(s')$

Mutual Exclusion Algorithm (revisited)

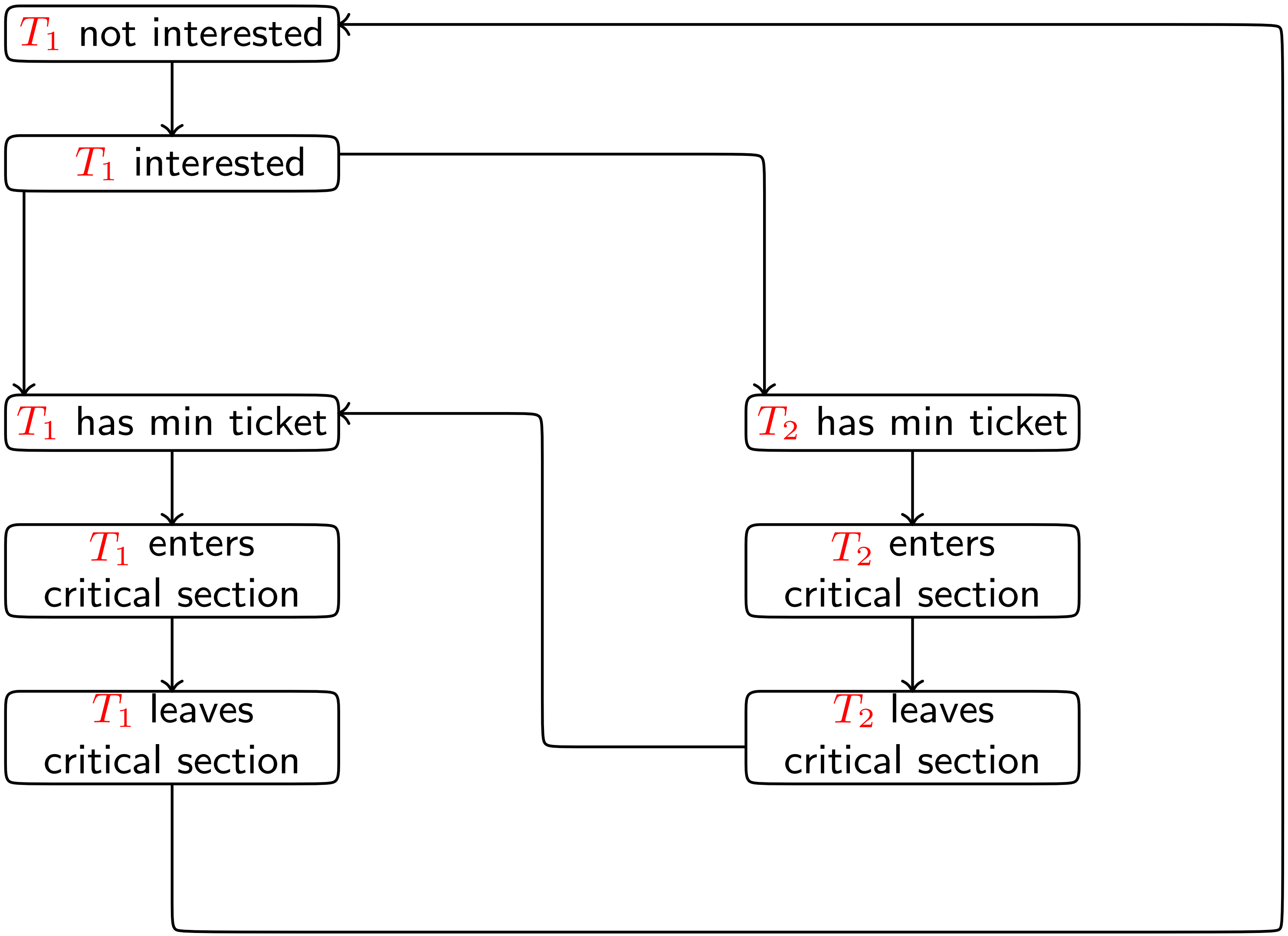
- For all k, $\varphi(k) \; = \; \Box(announced(k) \rightarrow \Diamond access\_critical(k))$

Mutual Exclusion Algorithm (revisited)

- For all k, $\varphi(k) \;=\; \square(\mathit{announced}(k) \rightarrow \diamond\, \mathit{access\_critical}(k))$
- By **symmetry**: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$

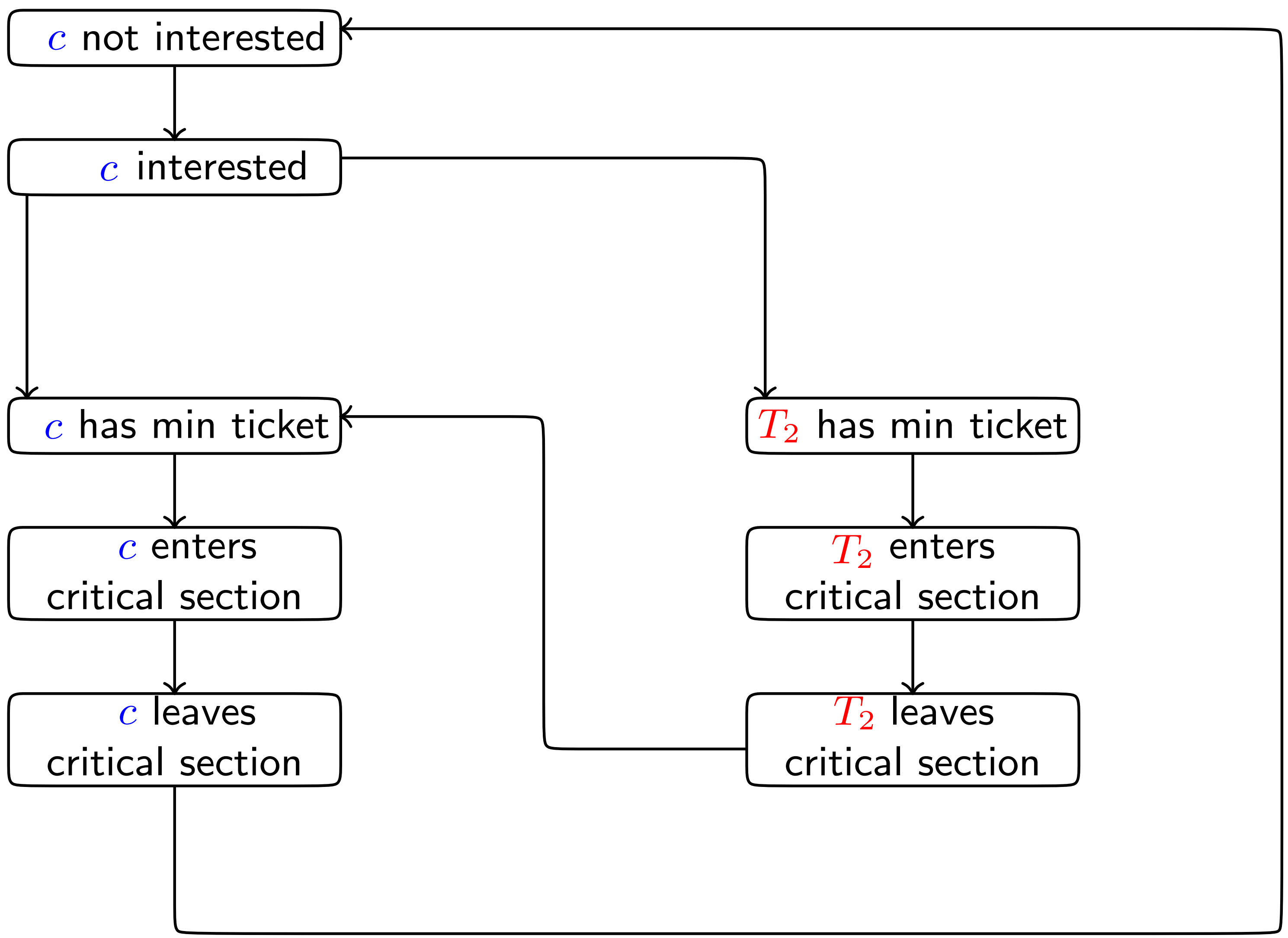
Mutual Exclusion Algorithm (revisited)

- For all k, $\varphi(k) \;=\; \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- By **symmetry**: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$



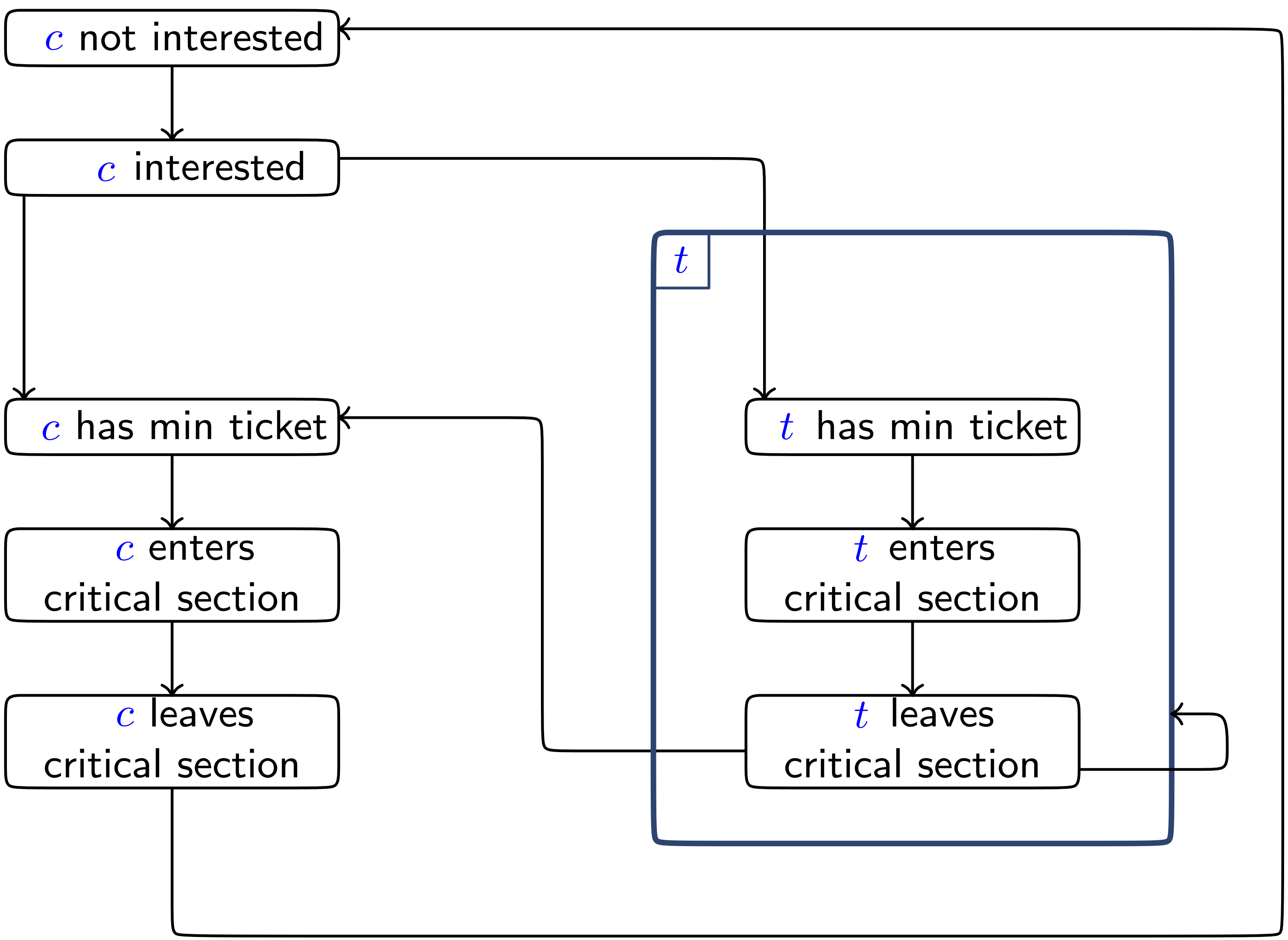
Mutual Exclusion Algorithm (revisited)

- For all k, $\varphi(k) \;=\; \Box(\mathit{announced}(k) \rightarrow \Diamond\, \mathit{access\_critical}(k))$
- By **symmetry**: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$



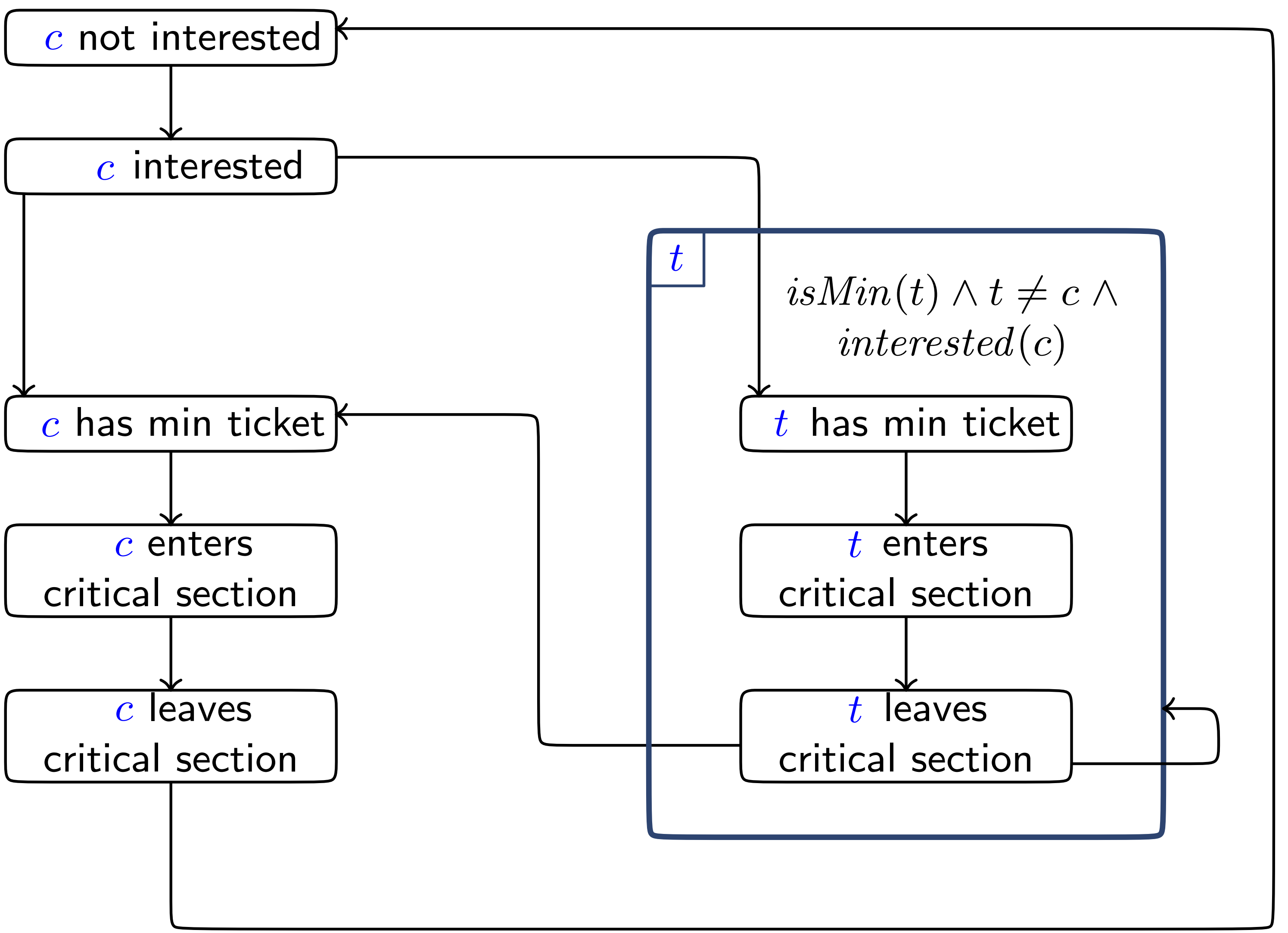
Mutual Exclusion Algorithm (revisited)

- For all k, $\varphi(k) = \square(announced(k) \rightarrow \diamond access\_critical(k))$
- By **symmetry**: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$



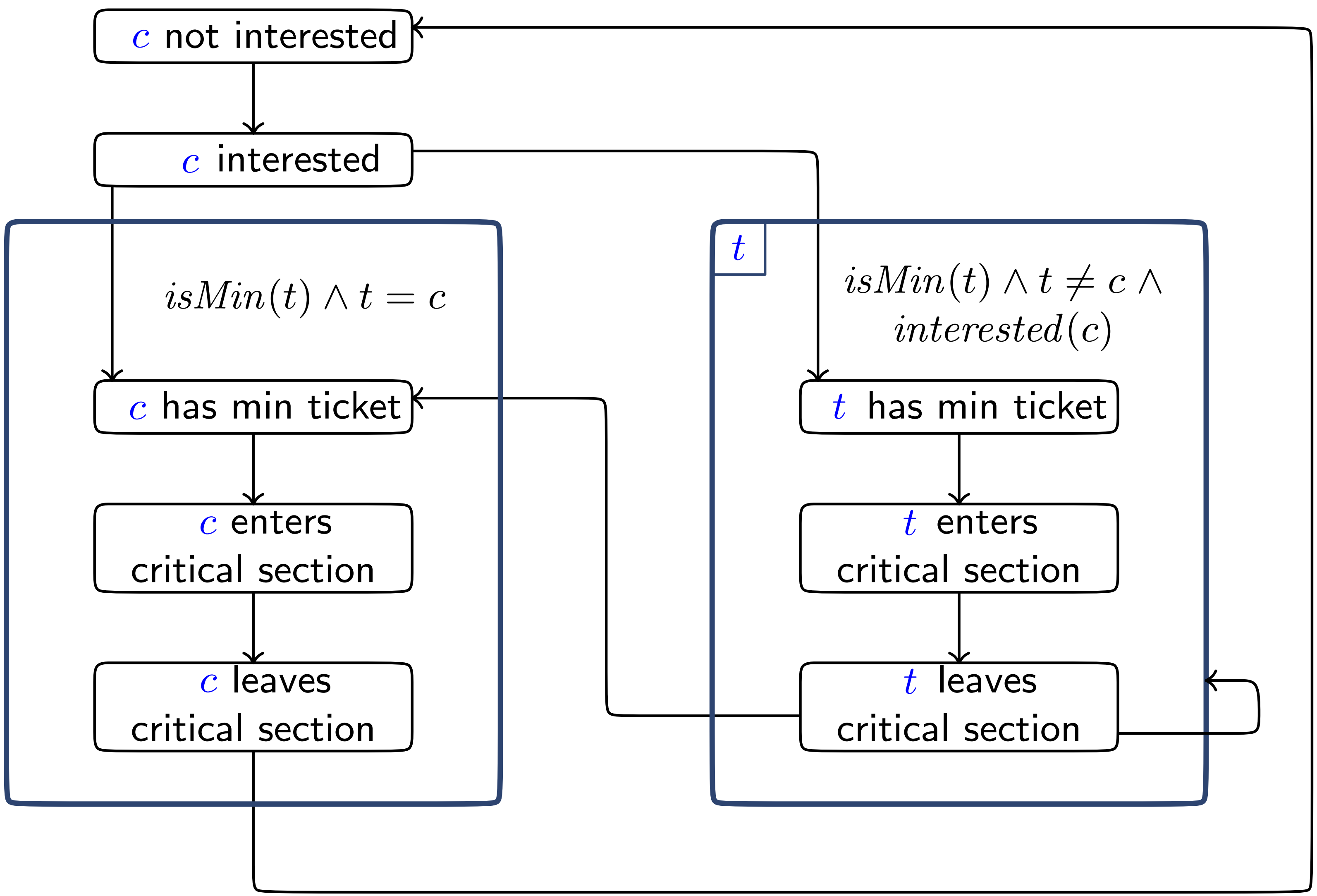
Mutual Exclusion Algorithm (revisited)

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- By **symmetry**: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$



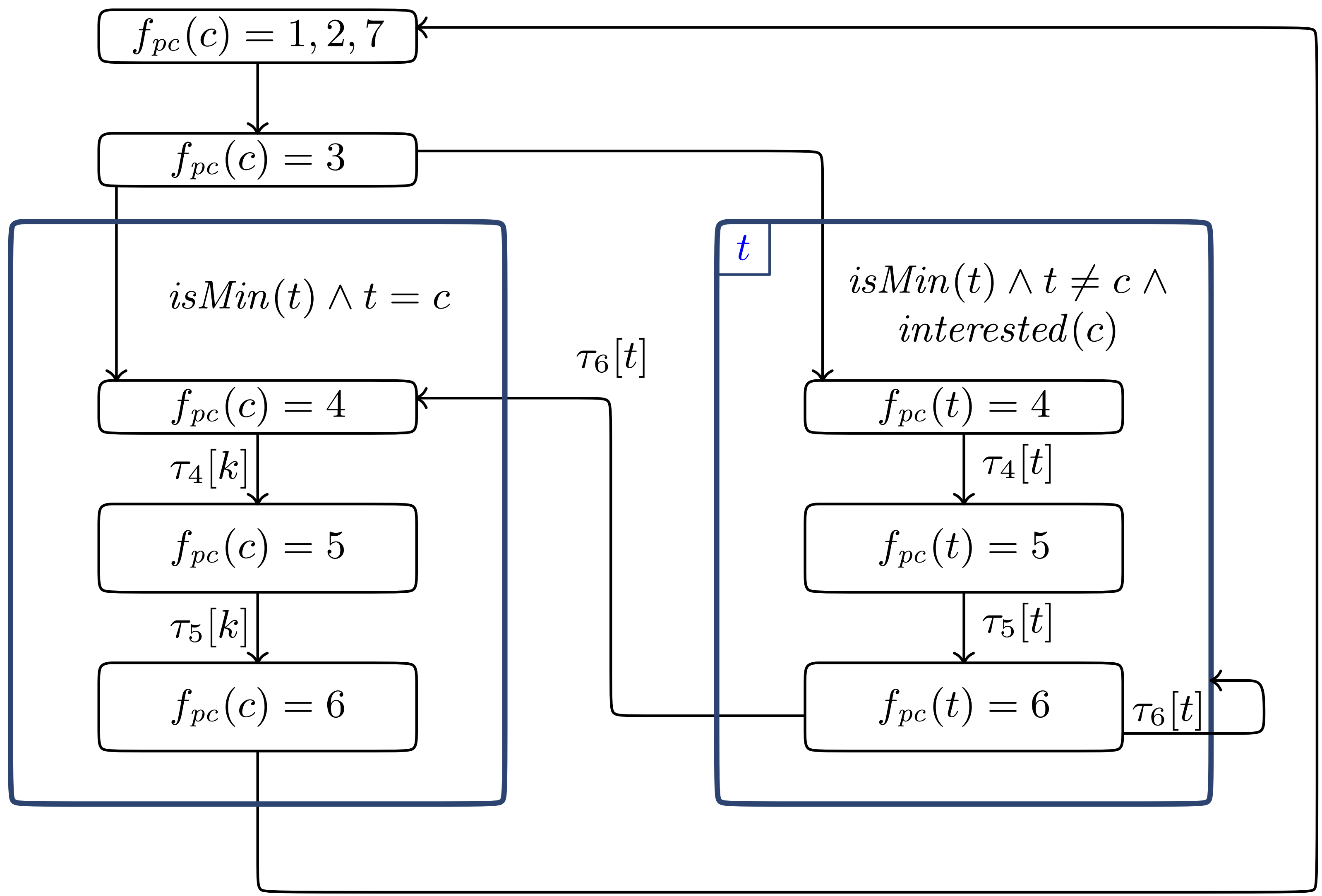
Mutual Exclusion Algorithm (revisited)

- For all k, $\varphi(k) = \Box(announced(k) \rightarrow \Diamond access\_critical(k))$
- By **symmetry**: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$



Mutual Exclusion Algorithm (revisited)

- For all k, $\varphi(k) = \square(announced(k) \rightarrow \diamond access\_critical(k))$
- By **symmetry**: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$



Conclusions

- Sound deductive method for concurrent **parametric systems**
- By now, works over **symmetric systems**
- A **unique diagram** for any arbitrary number of threads
- Proofs based on a **finite number** of **verification conditions**
- Posiblility of **combination** with **decision procedures**
- **Current and future** work:
  - Use of parametrized diagrams for the verification of concurrent list, skiplists, hashmaps...
  - Nested parametrized verification diagrams
  - Extension for non symmetric systems