

Deductive Temporal Verification of Parametrized Concurrent Systems

Alejandro Sánchez¹

César Sánchez^{1,2}

¹IMDEA Software Institute, Spain

²Spanish Council for Scientific Research (CSIC), Spain

SVARM'11, Saarbrücken, 2 April 2011

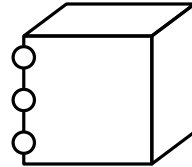
Verification of Concurrent Data-structures

Main Idea

Verification of Concurrent Data-structures

Main Idea

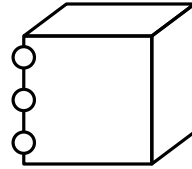
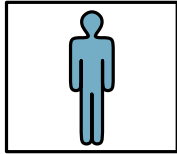
Concurrent DataStructure



Verification of Concurrent Data-structures

Main Idea

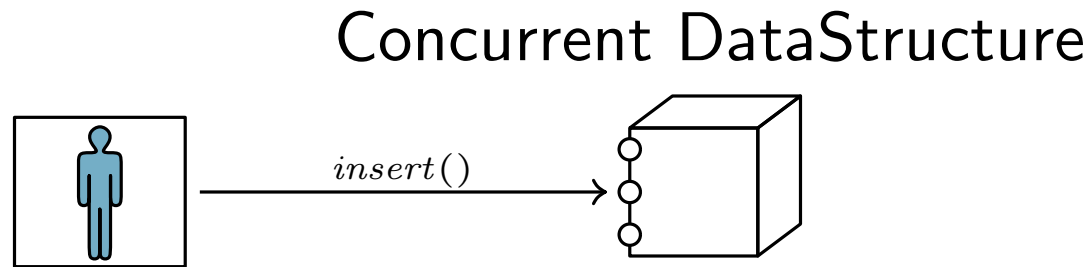
Concurrent DataStructure



Most General Client

Verification of Concurrent Data-structures

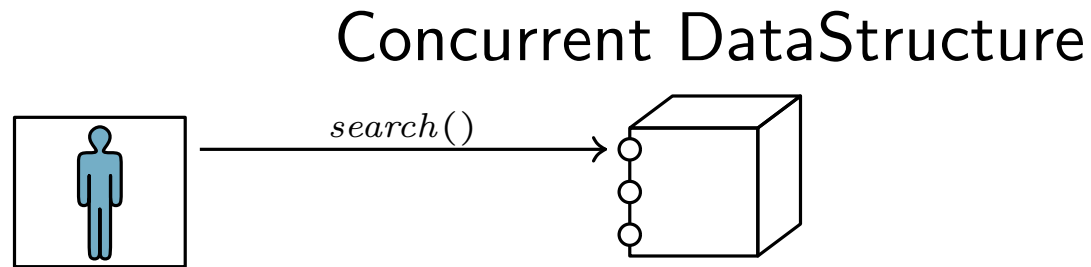
Main Idea



Most General Client

Verification of Concurrent Data-structures

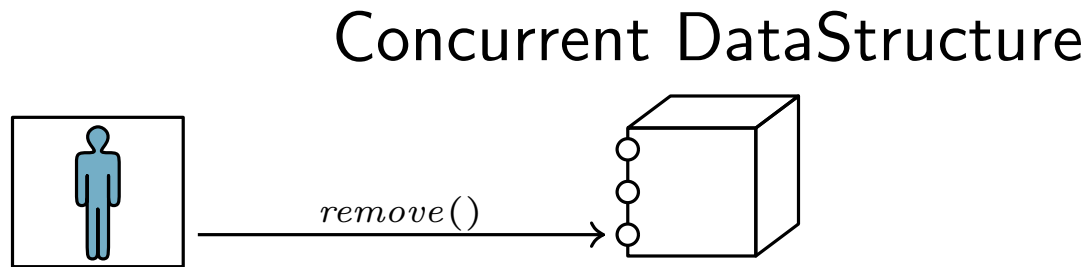
Main Idea



Most General Client

Verification of Concurrent Data-structures

Main Idea

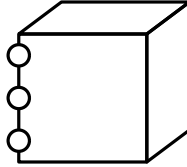
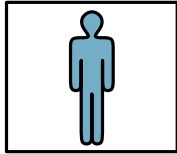


Most General Client

Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

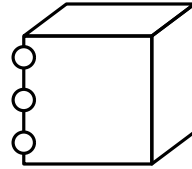


Most General Client

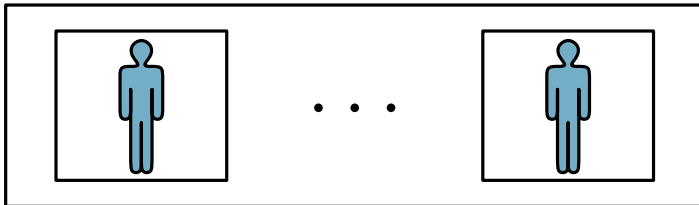
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



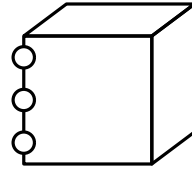
Most General Client



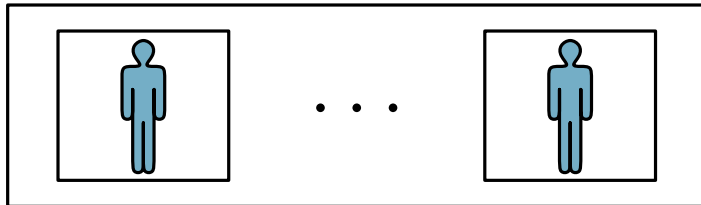
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client

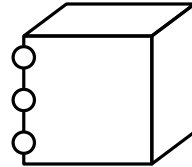


$$P[N] : P(1) || \dots || P(N)$$

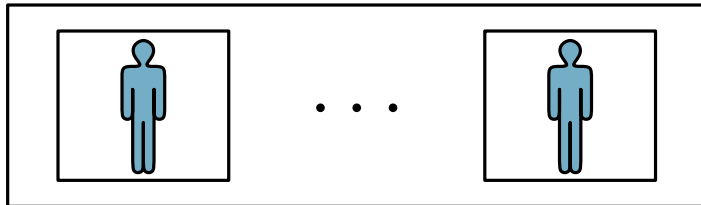
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



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

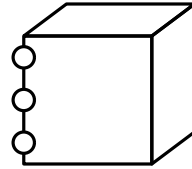
Property

$\varphi^{(k)}$

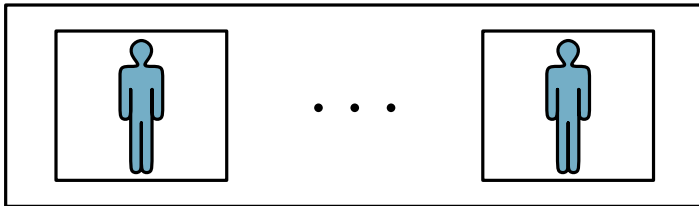
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



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

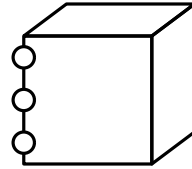
Property

$\varphi^{(k)}$
LTL ($\square, \diamond, \mathcal{U}, \dots$)

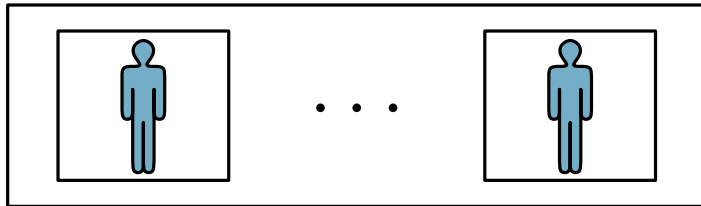
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



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

Diagram

\mathcal{D}

Property

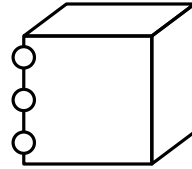
$\varphi^{(k)}$

LTL ($\square, \diamond, \mathcal{U}, \dots$)

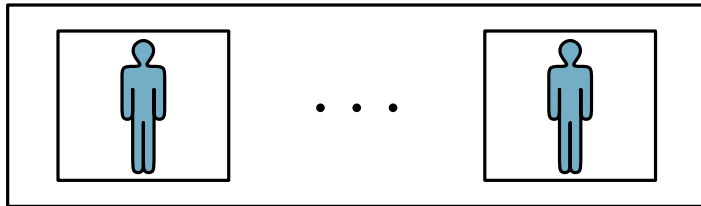
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



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

Diagram

\mathcal{D}

Property

$\varphi^{(k)}$

Verification Conditions:

- ▶ Initiation
- ▶ Consecution
- ▶ Acceptance
- ▶ Fairness

Satisfaction
(Model Checking)

Motivating Example: Mutual Exclusion Algorithm

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> announced := \emptyset

procedure MUTEXC

Int ticket

begin

1: **loop**

2: **nondet**

3: $\left\langle \begin{array}{l} \textit{ticket} := \textit{tick} ++ \\ \textit{announced.add}(\textit{ticket}) \end{array} \right\rangle$

4: **await** (*announced.min == ticket*)

5: **critical**

6: *announced.remove(ticket)*

7: **end loop**

end procedure

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> 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



Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> announced := \emptyset

procedure MUTEXC

Int ticket

begin

1: **loop**

2: **nondet**

3: $\left\langle \begin{array}{l} ticket := tick ++ \\ announced.add(ticket) \end{array} \right\rangle$

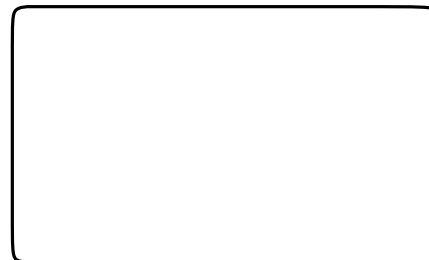
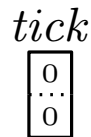
4: **await** (*announced.min == ticket*)

5: **critical**

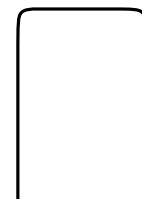
6: *announced.remove(ticket)*

7: **end loop**

end procedure



Announced Set



Critical Section

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> 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**

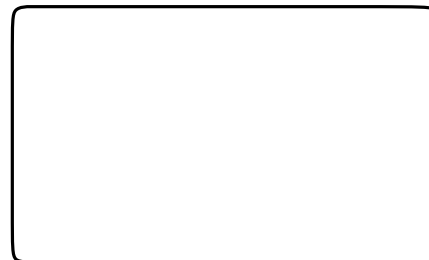
6: *announced.remove(ticket)*

7: **end loop**

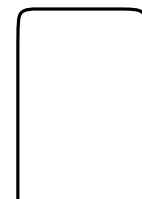
end procedure



tick
 $\begin{array}{|c|} \hline 0 \\ \hline \vdots \\ \hline 0 \\ \hline \end{array}$



Announced Set



Critical Section

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> 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**

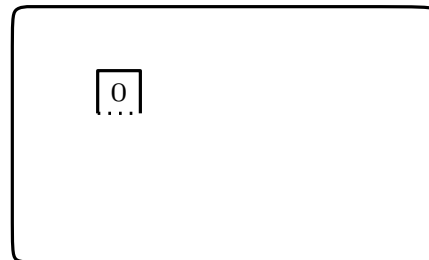
6: *announced.remove(ticket)*

7: **end loop**

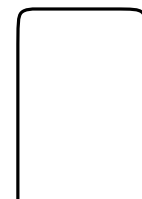
end procedure



tick
 $\begin{array}{c} 1 \\ \dots \\ 1 \end{array}$



Announced Set



Critical Section

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> 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**

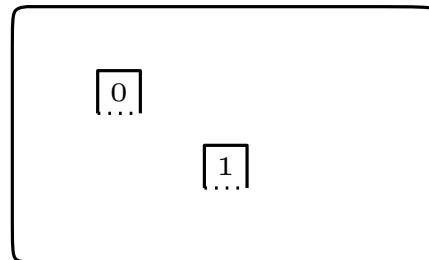
6: *announced.remove(ticket)*

7: **end loop**

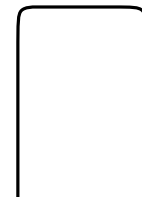
end procedure



tick
 $\begin{bmatrix} 2 \\ \dots \\ 2 \end{bmatrix}$



Announced Set



Critical Section

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> 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**

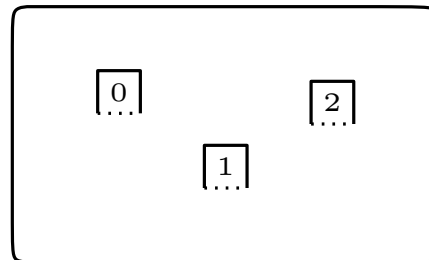
6: *announced.remove(ticket)*

7: **end loop**

end procedure



tick
 $\begin{bmatrix} 3 \\ \dots \\ 3 \end{bmatrix}$



Announced Set



Critical Section

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> announced := \emptyset

procedure MUTEXC

Int ticket

begin

1: **loop**

2: **nondet**

3: $\left\langle \begin{array}{l} ticket := tick ++ \\ announced.add(ticket) \end{array} \right\rangle$

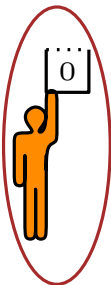
4: **await** (*announced.min == ticket*)

5: **critical**

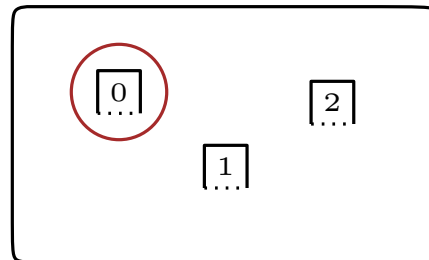
6: *announced.remove(ticket)*

7: **end loop**

end procedure



tick
 $\begin{bmatrix} 3 \\ \dots \\ 3 \end{bmatrix}$



Announced Set



Critical Section

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> announced := ∅

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**

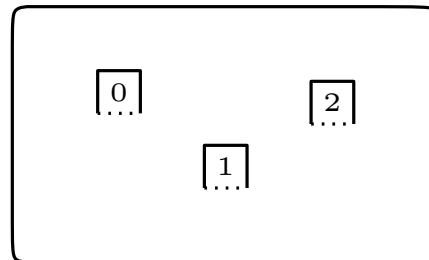
6: *announced.remove(ticket)*

7: **end loop**

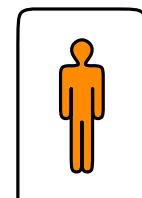
end procedure



tick
3
...
3



Announced Set



Critical Section

Motivating Example: Mutual Exclusion Algorithm

global

Int tick := 0

Set<Int> announced := \emptyset

procedure MUTEXC

Int ticket

begin

1: **loop**

2: **nondet**

3: $\left\langle \begin{array}{l} \textit{ticket} := \textit{tick} + + \\ \textit{announced.add}(\textit{ticket}) \end{array} \right\rangle$

4: **await** (*announced.min == ticket*)

5: **critical**

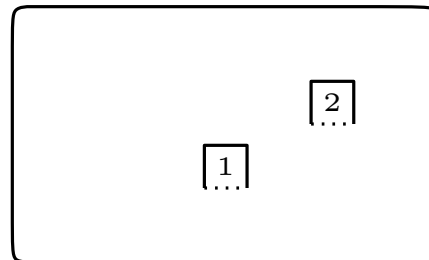
6: *announced.remove(ticket)*

7: **end loop**

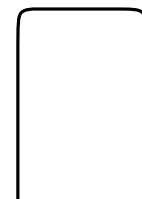
end procedure



tick
 $\begin{bmatrix} 3 \\ \dots \\ 3 \end{bmatrix}$



Announced Set



Critical Section

Verification Diagram for Mutual Exclusion Algorithm

- ▶ For all k , $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access_critical}(k))$

Verification Diagram for Mutual Exclusion Algorithm

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

Verification Diagram for Mutual Exclusion Algorithm

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

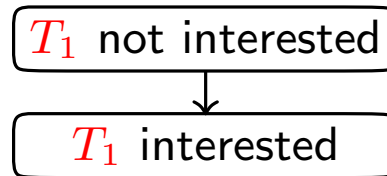
Verification Diagram for Mutual Exclusion Algorithm

- ▶ 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 $\varphi(T_1)$

T_1 not interested

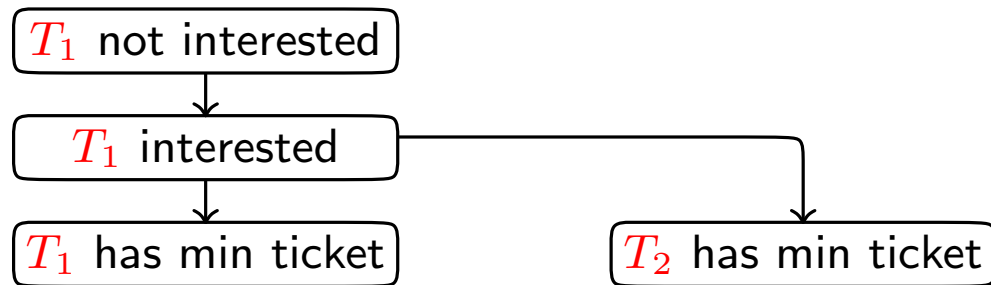
Verification Diagram for Mutual Exclusion Algorithm

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



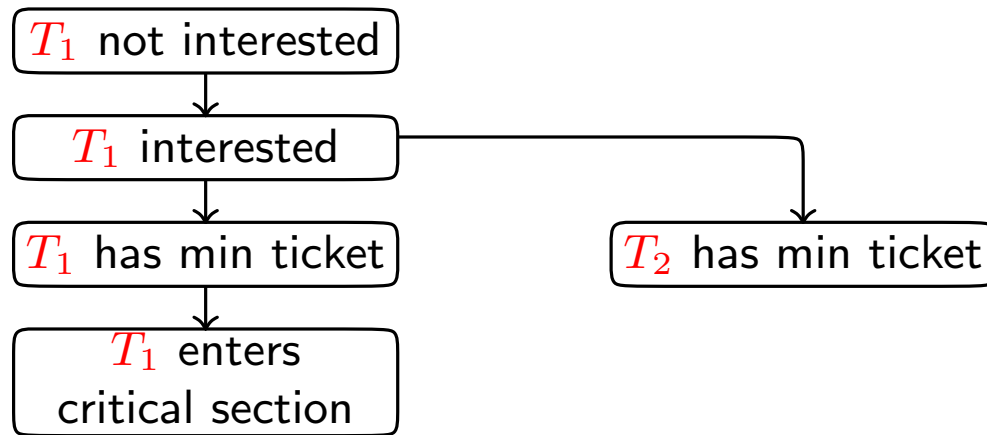
Verification Diagram for Mutual Exclusion Algorithm

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



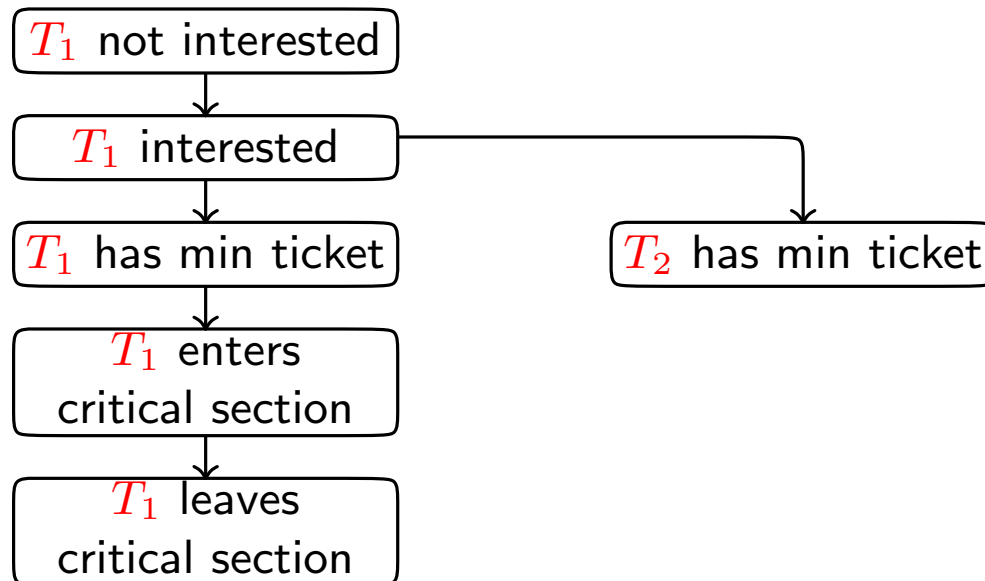
Verification Diagram for Mutual Exclusion Algorithm

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



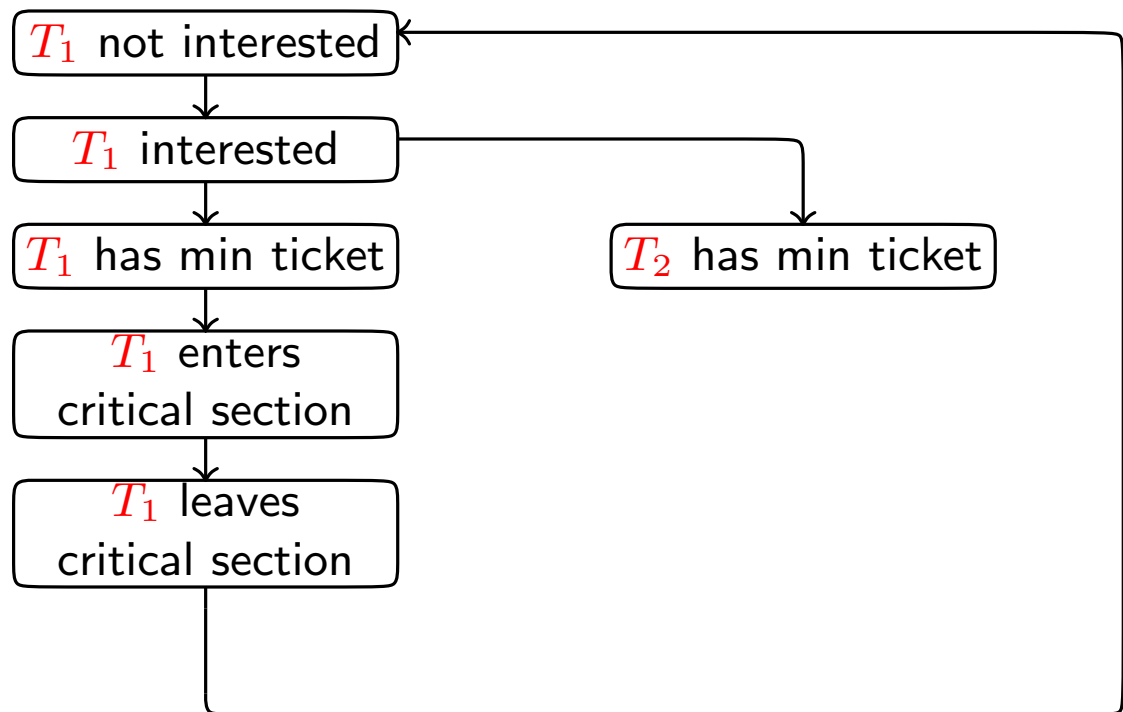
Verification Diagram for Mutual Exclusion Algorithm

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



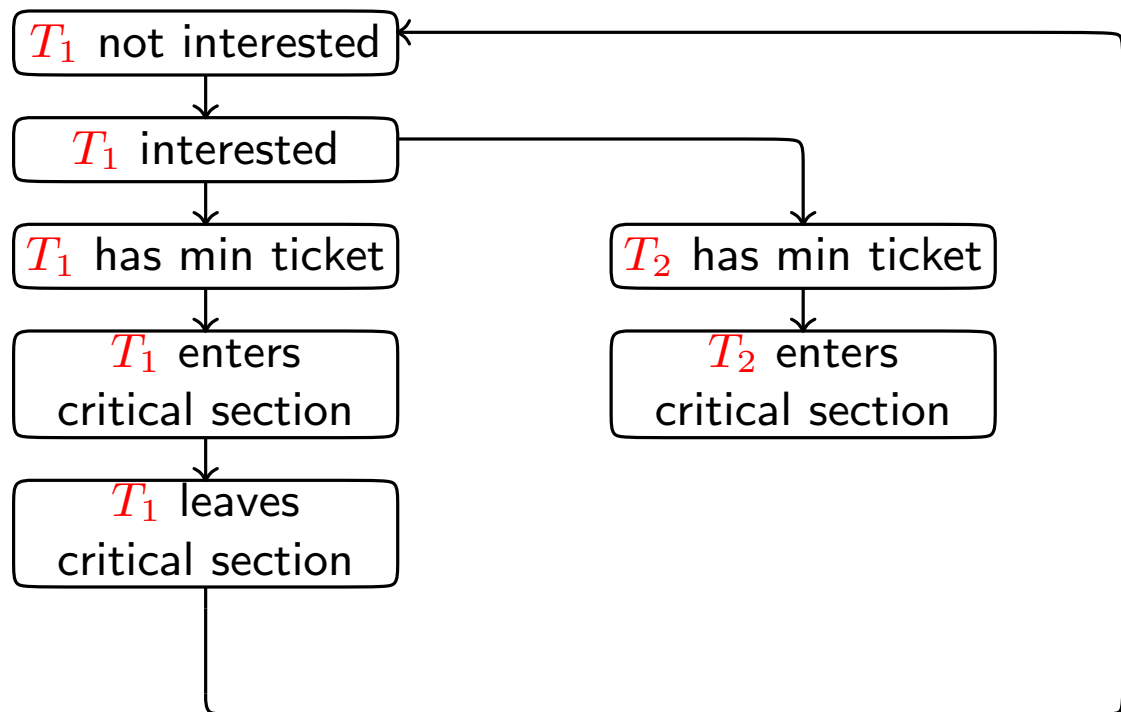
Verification Diagram for Mutual Exclusion Algorithm

- ▶ 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 $\varphi(T_1)$



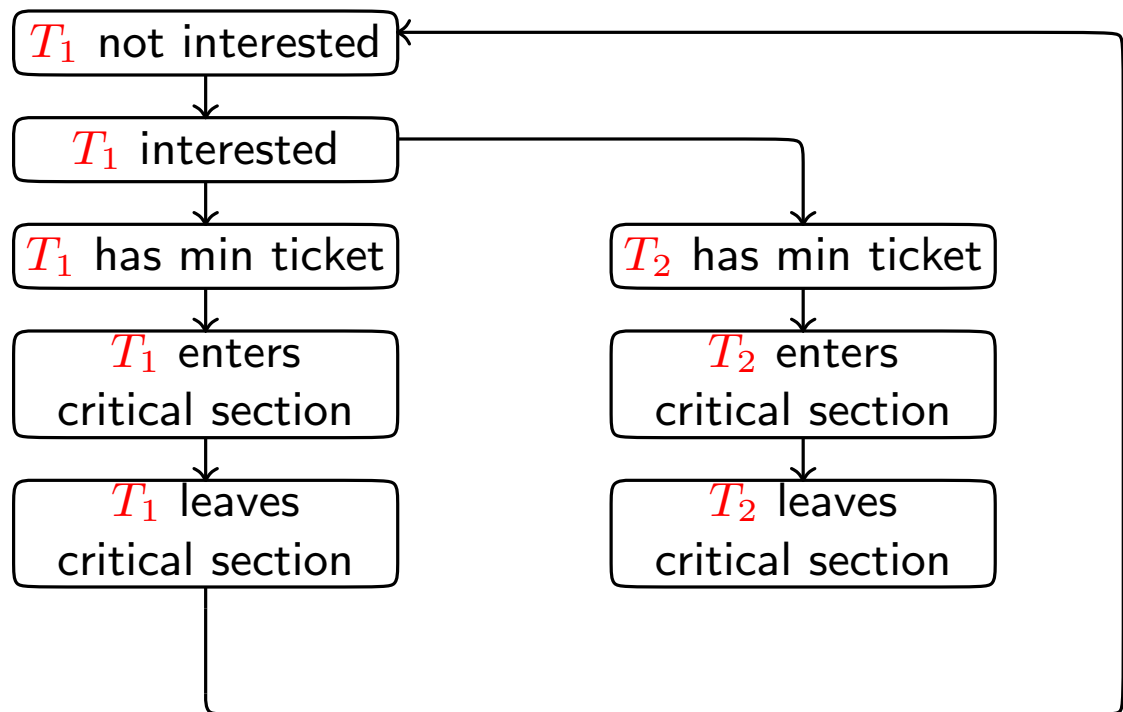
Verification Diagram for Mutual Exclusion Algorithm

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



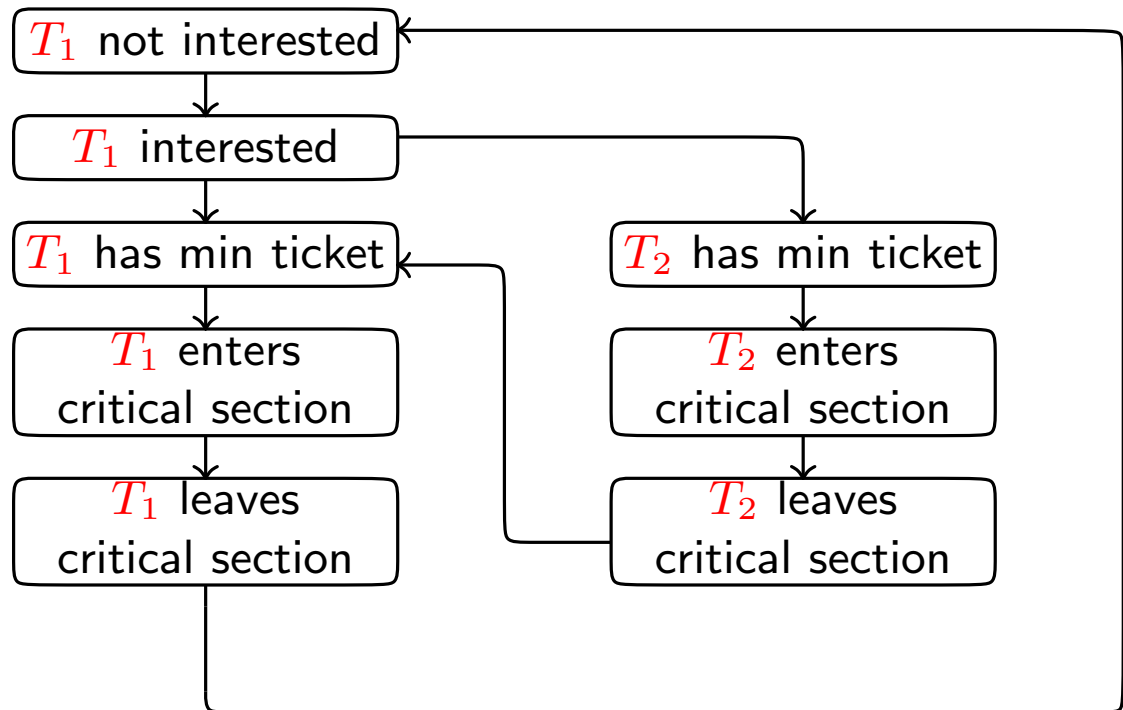
Verification Diagram for Mutual Exclusion Algorithm

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



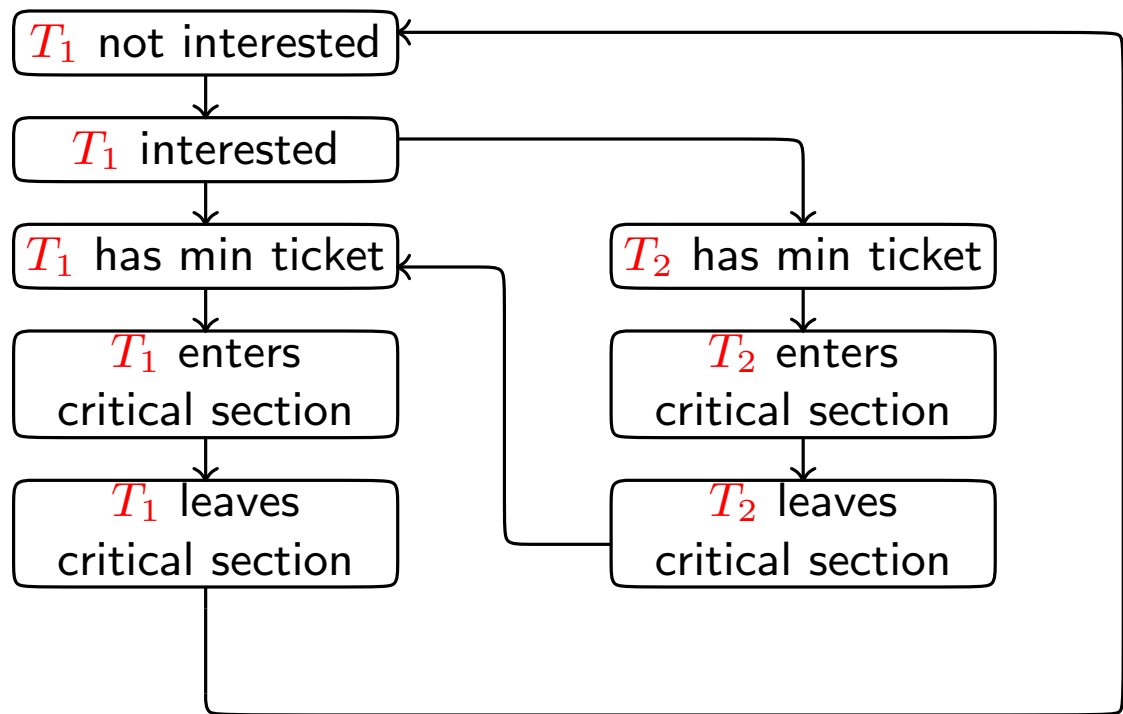
Verification Diagram for Mutual Exclusion Algorithm

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



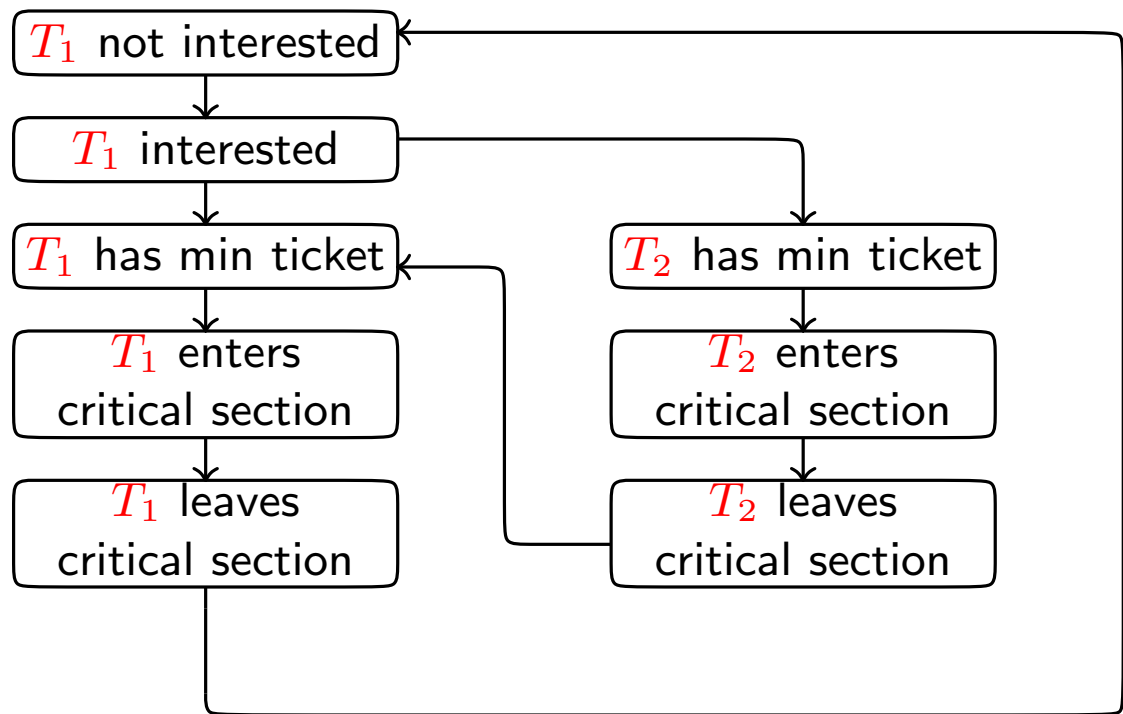
Some verification conditions

$$n \wedge \tau[1] \rightarrow succ(n)$$

$$n \wedge \tau[2] \rightarrow succ(n)$$

Verification Diagram for Mutual Exclusion Algorithm

- ▶ For all k , $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access_critical}(k))$
- ▶ 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



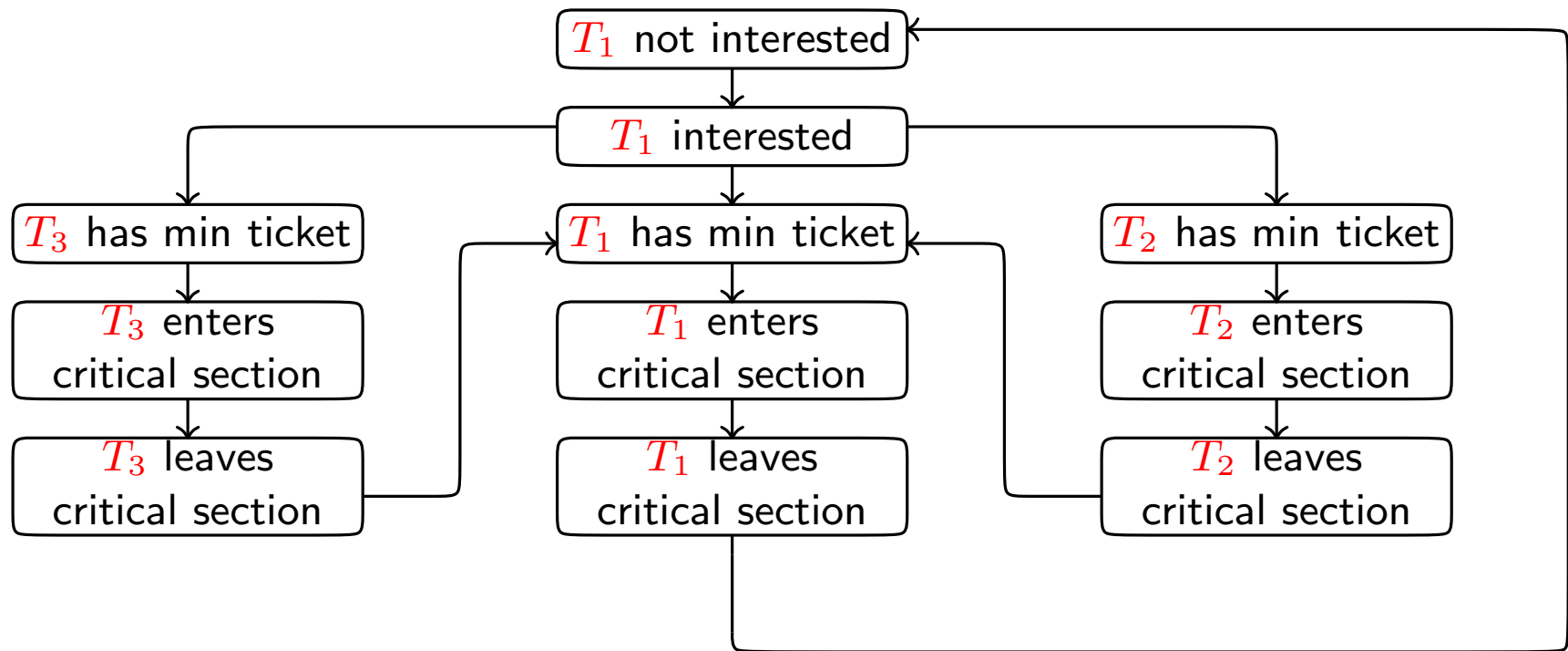
Some verification conditions

$$n \wedge \tau[1] \rightarrow \text{succ}(n)$$

$$n \wedge \tau[2] \rightarrow \text{succ}(n)$$

Verification Diagram for Mutual Exclusion Algorithm

- ▶ For all k , $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access_critical}(k))$
- ▶ 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



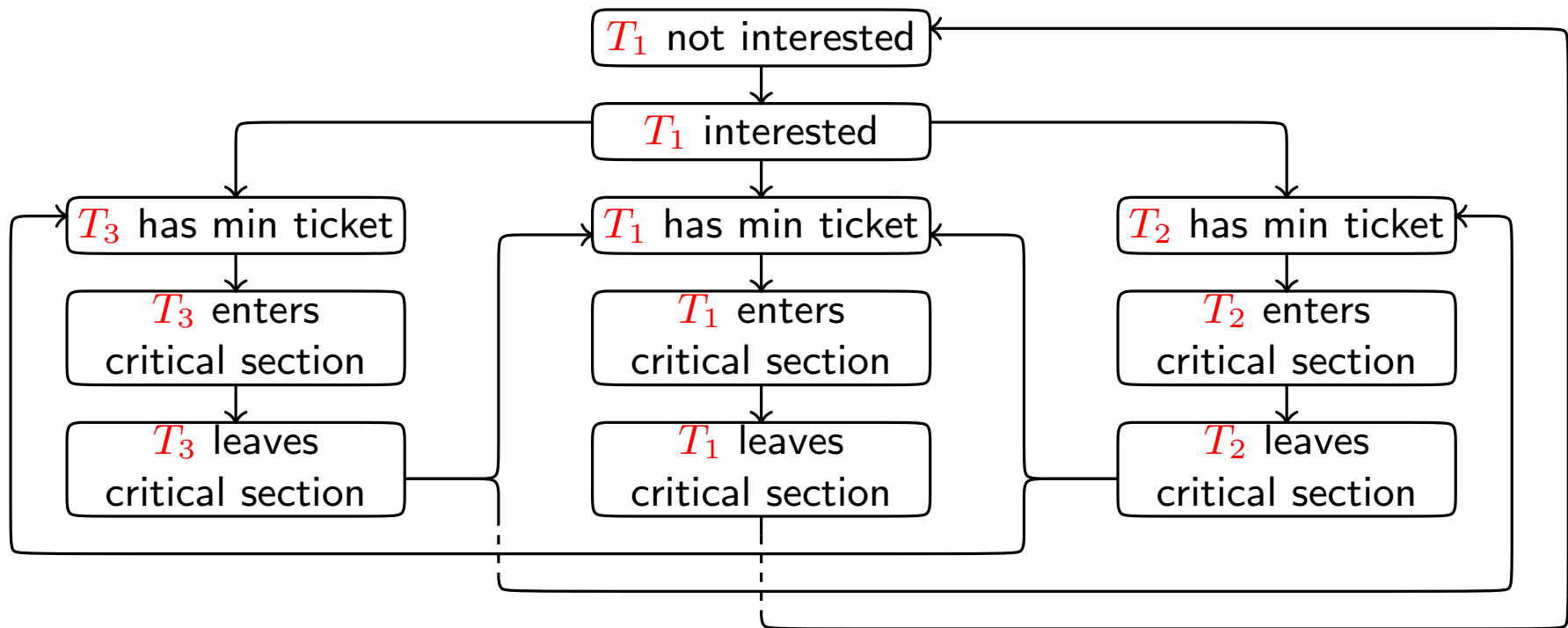
Some verification conditions

$$n \wedge \tau[1] \rightarrow \text{succ}(n)$$

$$n \wedge \tau[2] \rightarrow \text{succ}(n)$$

Verification Diagram for Mutual Exclusion Algorithm

- ▶ For all k , $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access_critical}(k))$
- ▶ 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



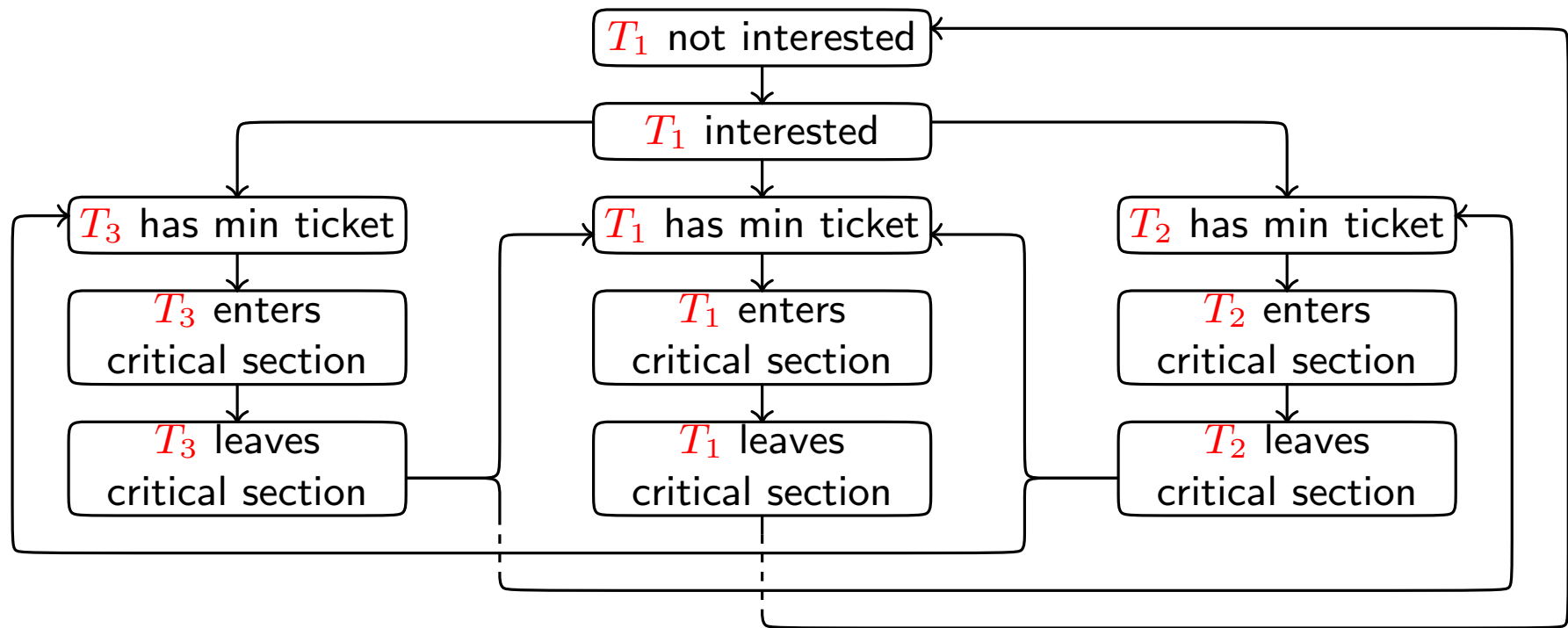
Some verification conditions

$$n \wedge \tau[1] \rightarrow \text{succ}(n)$$

$$n \wedge \tau[2] \rightarrow \text{succ}(n)$$

Verification Diagram for Mutual Exclusion Algorithm

- ▶ For all k , $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access_critical}(k))$
- ▶ 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

$$n \wedge \tau[1] \rightarrow succ(n)$$

$$n \wedge \tau[2] \rightarrow succ(n)$$

$$n \wedge \tau[3] \rightarrow succ(n)$$

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

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

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

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

$$V = V_{global}$$

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

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

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

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

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

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

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

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

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

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

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 := \emptyset

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

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

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 := ∅
```

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

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

$V = \{tick, announced\}$

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

Int tick := 0

Set<Int> announced := \emptyset

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

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

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

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 := \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 $\mathcal{S}^{[2]}$

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

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 := \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 $\mathcal{S}^{[2]}$

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

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

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 := \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 $\mathcal{S}^{[2]}$

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

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

$$\mathcal{J} = \mathcal{T}$$

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**

$\sigma = s_0 \quad s_1 \quad s_2 \quad \dots$

Symmetric Systems

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

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

Symmetric Systems

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

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

$$i, j \in [M]$$

Symmetric Systems

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

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

$i, j \in [M]$ | 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**

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

$i, j \in [M]$

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

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

Symmetric Systems

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

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

$i, j \in [M]$

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 $\Sigma_{\mathbf{tid}} = (\{\mathbf{tid}\}, \emptyset, \emptyset)$ and $T_{\mathbf{param}}$ of uninterpreted functions

Parametrized Verification Diagrams

- ▶ Prove that **all instances** of $\mathcal{S}^{[M]}$ satisfy a temporal specification
with a unique diagram
- ▶ We add $\Sigma_{\mathbf{tid}} = (\{\mathbf{tid}\}, \emptyset, \emptyset)$ and $T_{\mathbf{param}}$ of uninterpreted functions
- ▶ For each $v : \alpha$, we add $f_v : \mathbf{tid} \rightarrow \alpha$
 $f_{pc} : \mathbf{tid} \rightarrow Loc$
- ▶ $T_{\mathbf{param}}$ 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 $\Sigma_{\text{tid}} = (\{\mathbf{tid}\}, \emptyset, \emptyset)$ and T_{param} of uninterpreted functions
- ▶ For each $v : \alpha$, we add $f_v : \text{tid} \rightarrow \alpha$
 $f_{pc} : \text{tid} \rightarrow \text{Loc}$
- ▶ T_{param} is:
 - ▶ stable infinite $\xrightarrow{\text{combinable with}}$ stable infinite theories
 - ▶ polite $\xrightarrow{\hspace{1.5cm}}$ non stable infinite theories

$$T = T_{\text{prog}} + T_{\text{param}}$$

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:

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

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:

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

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

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

(a)

(b)

Parametrized Verification Diagrams

- ▶ PVDS are **an extension** of GVD

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

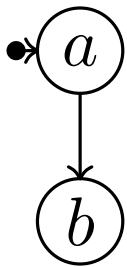
• $\circlearrowleft a$

$\circlearrowleft b$

Parametrized Verification Diagrams

- ▶ PVDS are **an extension** of GVD

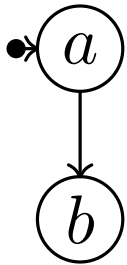
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, \mathbf{E}, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- ▶ PVDS are **an extension** of GVD

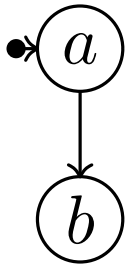
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- ▶ PVDS are **an extension** of GVD

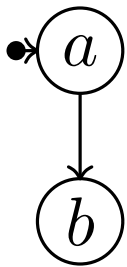
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

- ▶ PVDS are **an extension** of GVD

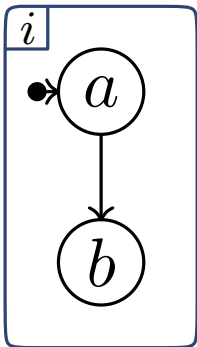
$$\mathcal{D}^{[\mathcal{M}]} : \langle N, N_0, B, E, \mu, \mathcal{F}, \eta, \Delta, f \rangle$$



Parametrized Verification Diagrams

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

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

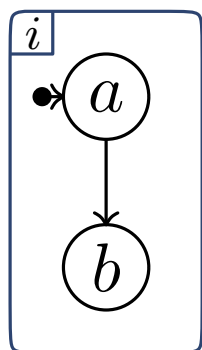


Parametrized Verification Diagrams

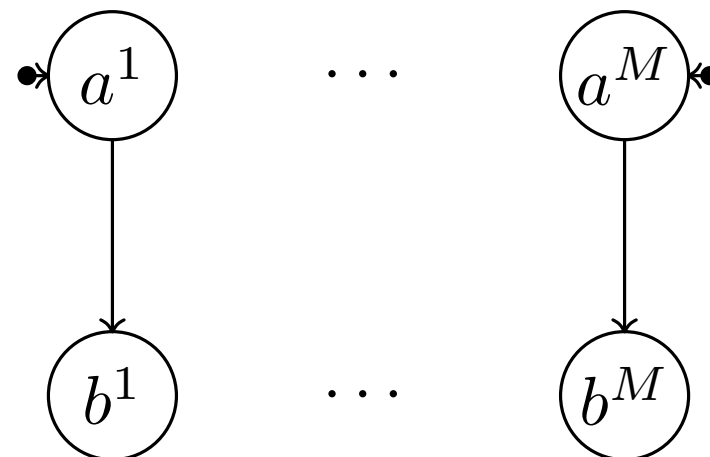
- ▶ PVDS are **an extension** of GVD
- ▶ We add the notion of **boxes**

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

For M threads



represents

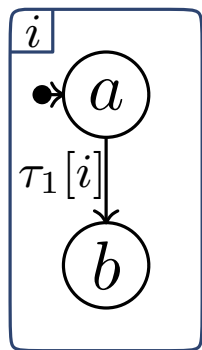


Parametrized Verification Diagrams

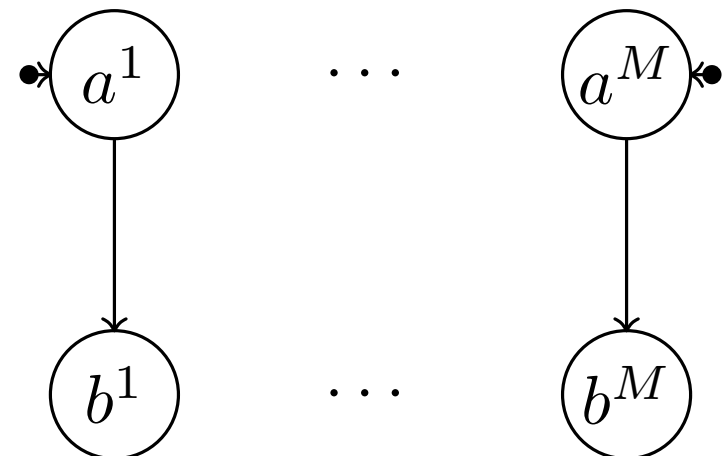
- ▶ PVDS are **an extension** of GVD
- ▶ We add the notion of **boxes**

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

For M threads



represents

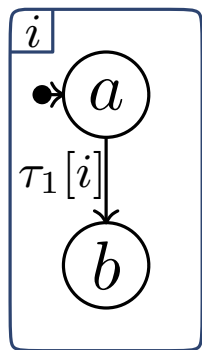


Parametrized Verification Diagrams

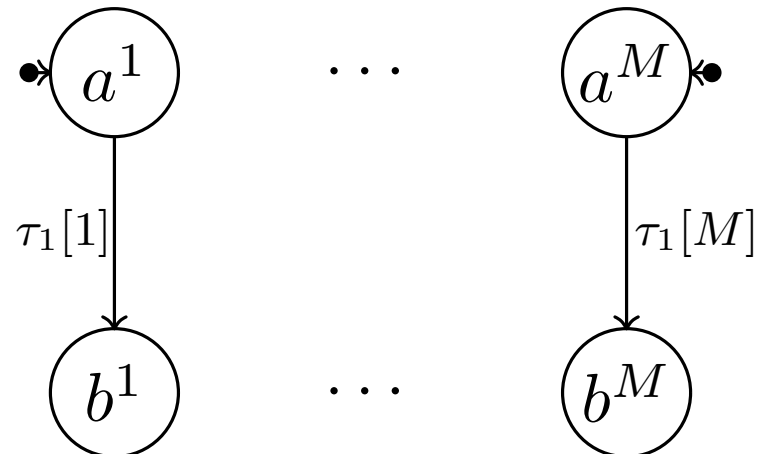
- ▶ PVDS are **an extension** of GVD
- ▶ We add the notion of **boxes**

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

For M threads



represents

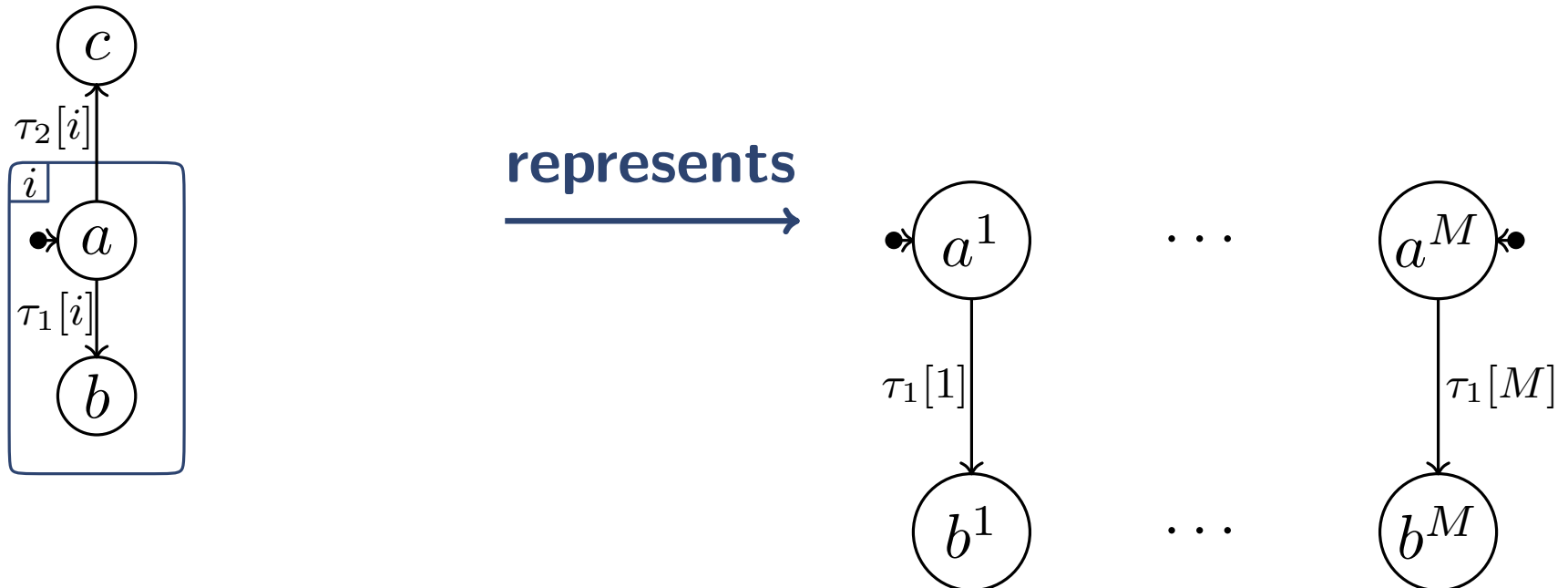


Parametrized Verification Diagrams

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

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

For M threads

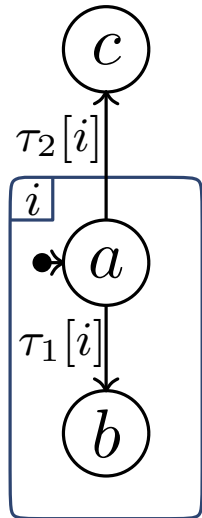


Parametrized Verification Diagrams

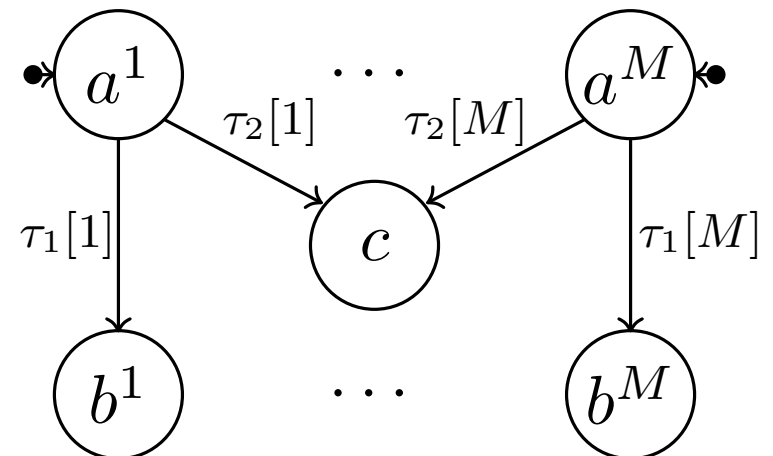
- ▶ PVDS are **an extension** of GVD
- ▶ We add the notion of **boxes**

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

For M threads



represents

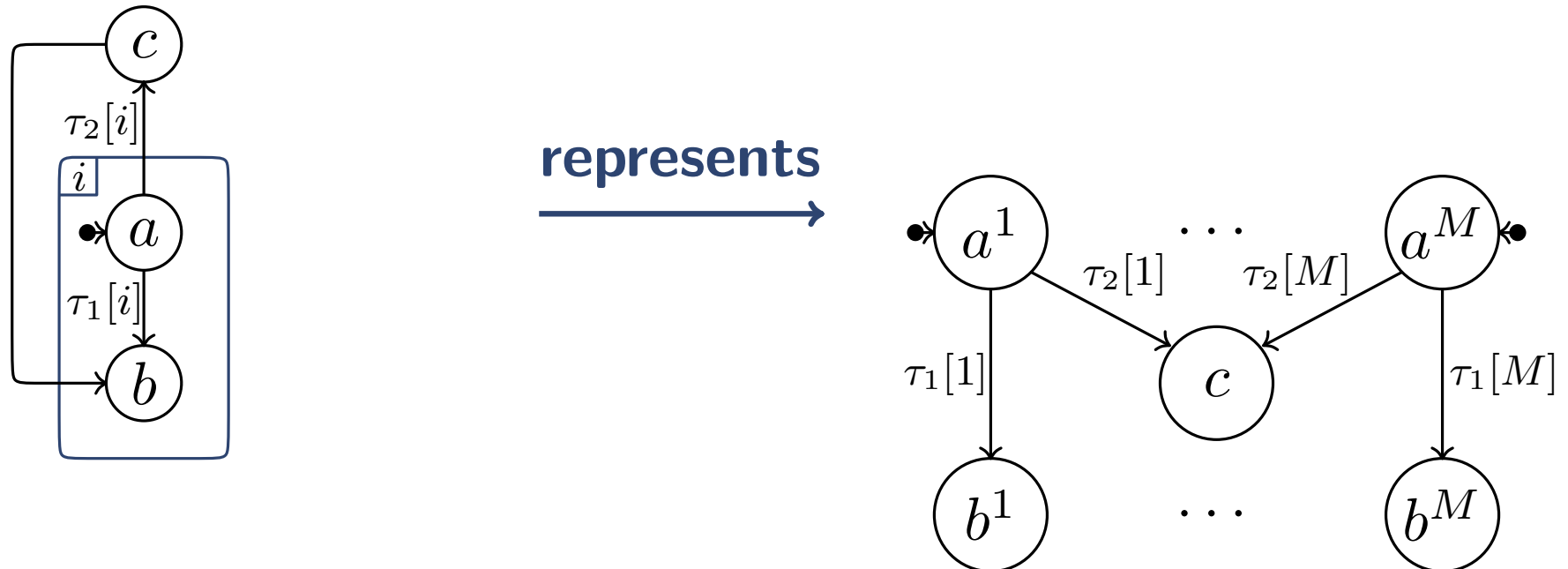


Parametrized Verification Diagrams

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

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

For M threads

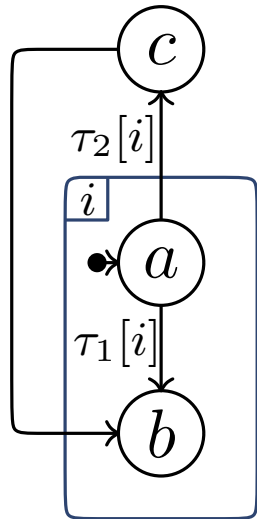


Parametrized Verification Diagrams

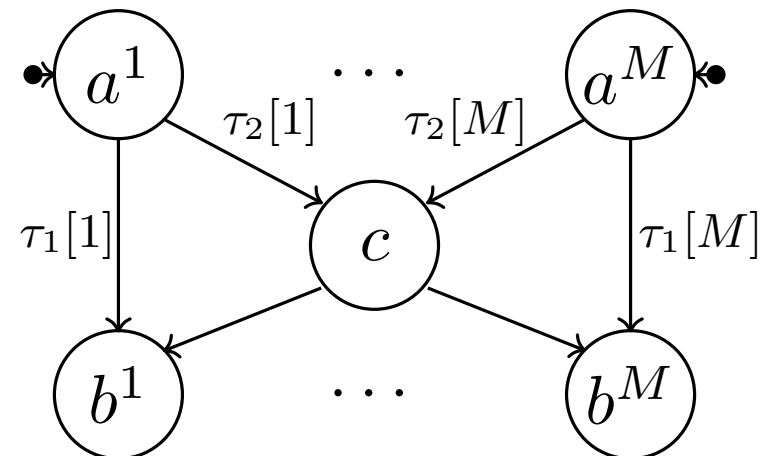
- ▶ PVDS are **an extension** of GVD
- ▶ We add the notion of **boxes**

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

For M threads



represents

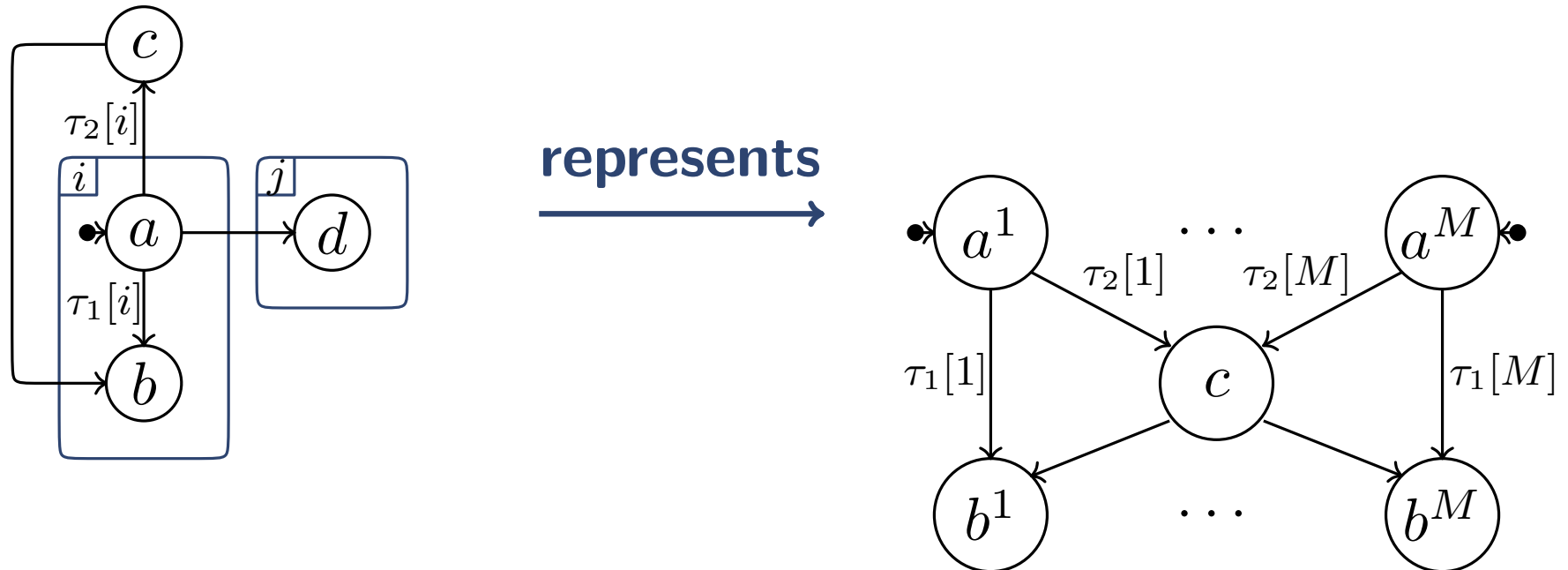


Parametrized Verification Diagrams

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

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

For M threads

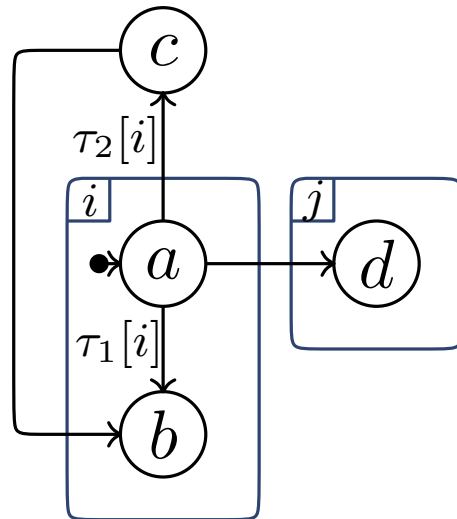


Parametrized Verification Diagrams

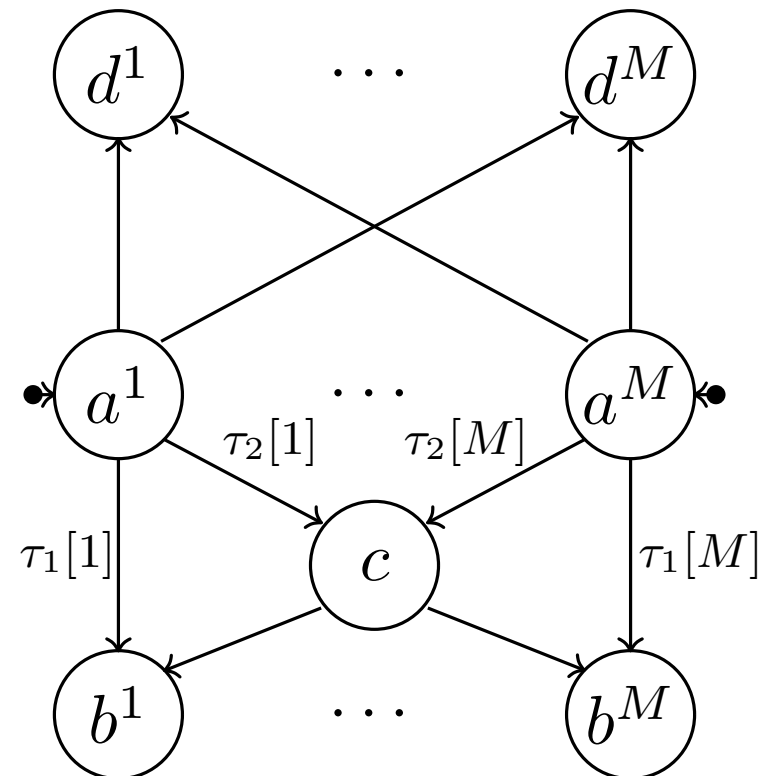
- ▶ PVDS are **an extension** of GVD
- ▶ We add the notion of **boxes**

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

For M threads



represents

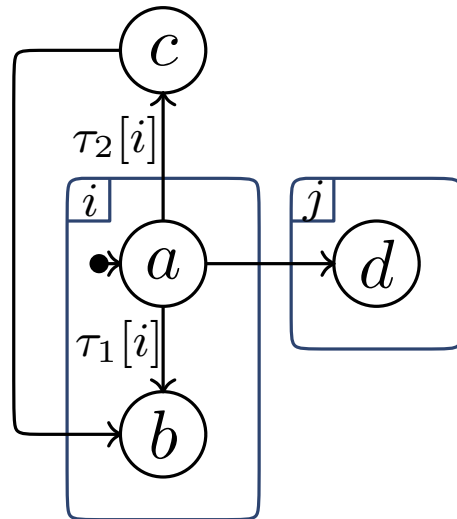


Parametrized Verification Diagrams

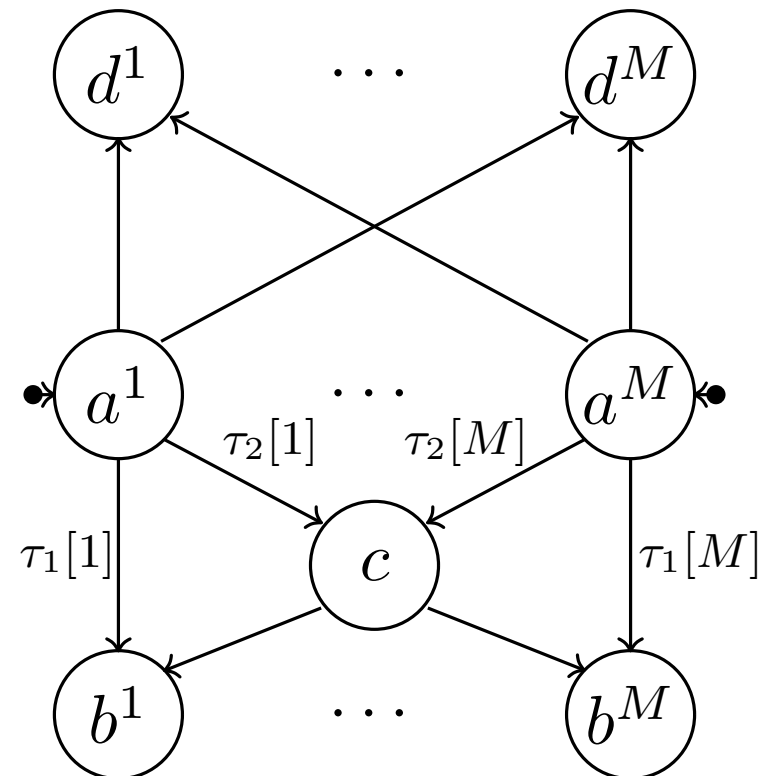
- ▶ PVDS are **an extension** of GVD
- ▶ We add the notion of **boxes**
- ▶ A PVD **abstracts all instantiations** of a parametric system

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

For M threads



represents



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 $\tau \in \mathcal{T}$,

$$n \wedge \tau \rightarrow \text{succ}(n)$$

Verification Conditions for Parametrized Diagrams

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

$$n \wedge \tau \rightarrow \text{succ}(n)$$

$$\bigwedge_l \bigwedge_i n \wedge \tau_l[i] \rightarrow \text{succ}(n)$$

Verification Conditions for Parametrized Diagrams

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

$$n \wedge \tau \rightarrow succ(n)$$

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

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 $\tau \in \mathcal{T}$,

$$n \wedge \tau \rightarrow succ(n)$$

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

$$\text{Voc}(n, succ(n)) = \{i_1, \dots, i_q\} = I$$

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 $\tau \in \mathcal{T}$,

$$n \wedge \tau \rightarrow succ(n)$$

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

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

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

Verification Conditions for Parametrized Diagrams

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

$$n \wedge \tau \rightarrow succ(n)$$

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

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

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

Verification Conditions for Parametrized Diagrams

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

$$n \wedge \tau \rightarrow succ(n)$$

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

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

$$\bigwedge_l n \wedge \tau_l[i_1] \rightarrow succ(n)$$

$$\vdots$$

$$\bigwedge_l n \wedge \tau_l[i_q] \rightarrow succ(n)$$

$$\bigwedge_l n \wedge \tau_l[i] \wedge \bigwedge_{j \in I} i \neq j \rightarrow succ(n)$$



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 $\tau \in \mathcal{T}$,

$$n \wedge \tau \rightarrow succ(n)$$

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

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

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

unbounded



L × M

Before

bounded



L × (q + 1)

Now

verification conditions

Verification Conditions for Parametrized Diagrams

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

► **Consecution:** For every $n \in N$, let $I = \text{Voc}(n, \text{next}(n))$,

$$(C1) \quad \mu(n)(s) \wedge \rho_{\tau[i]}(s, s') \rightarrow \mu(\text{next}(n))(s') \quad , \text{ for each } i \in I$$

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

► **Acceptance:** If $(n_1, n_2) \in P \setminus R$, let $I = \text{Voc}(n_1, n_2)$,

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

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

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

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

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

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

$$(F1) \quad \mu(n_1)(s) \wedge \tau[i] \in \eta(e) \rightarrow \text{En}(\tau[i])$$

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

Verification Conditions for Parametrized Diagrams

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

► **Consecution:** For every $n \in N$, let $I = \text{Voc}(n, \text{next}(n))$,

$$(C1) \quad \mu(n)(s) \wedge \rho_{\tau[i]}(s, s') \rightarrow \mu(\text{next}(n))(s') \quad , \text{ for each } i \in I$$

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

► **Acceptance:** If $(n_1, n_2) \in P \setminus R$, let $I = \text{Voc}(n_1, n_2)$,

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

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

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

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

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

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

$$(F1) \quad \mu(n_1)(s) \wedge \tau[i] \in \eta(e) \rightarrow \text{En}(\tau[i])$$

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

Mutual Exclusion Algorithm (revisited)

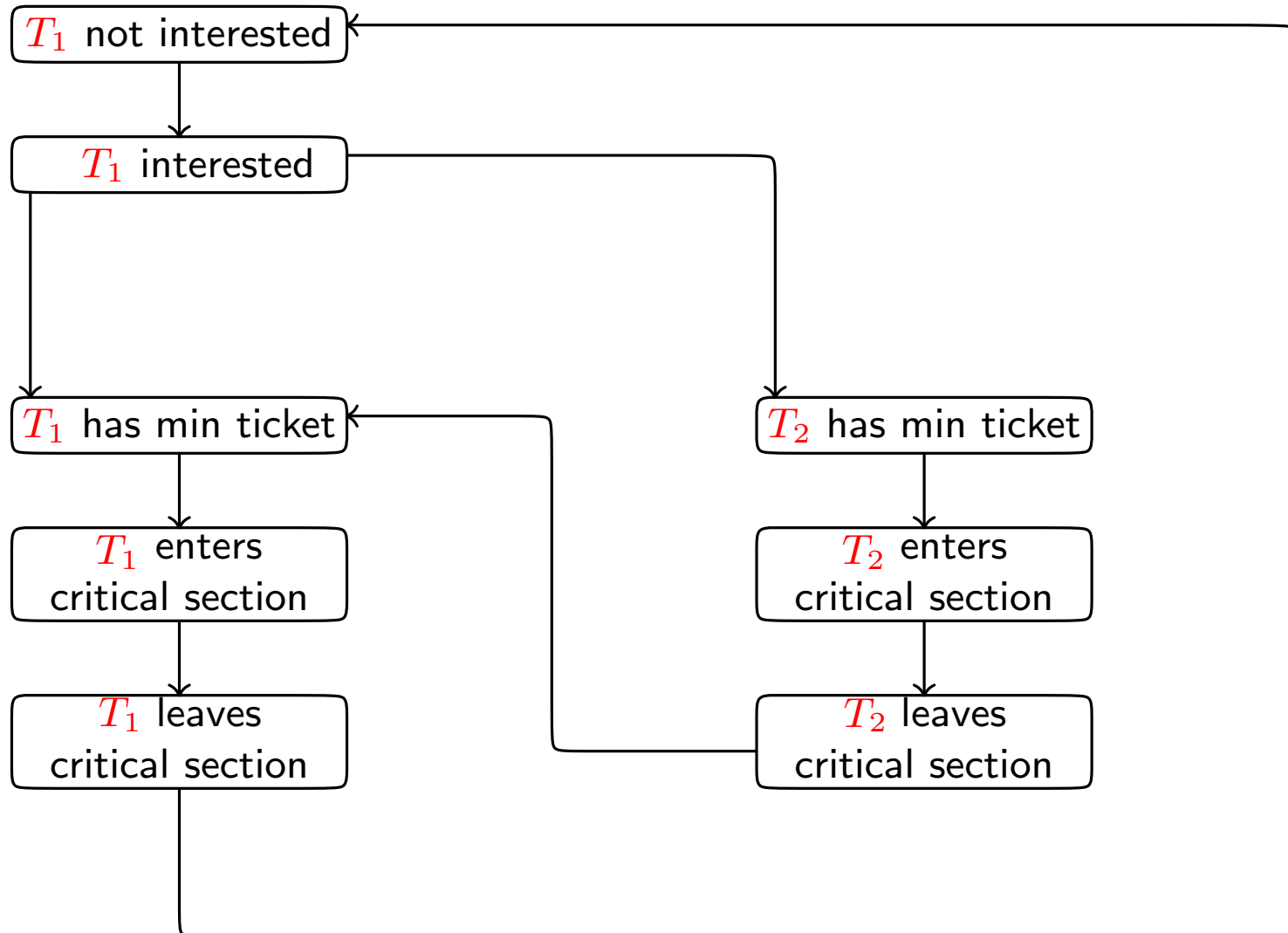
- ▶ For all k , $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access_critical}(k))$

Mutual Exclusion Algorithm (revisited)

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

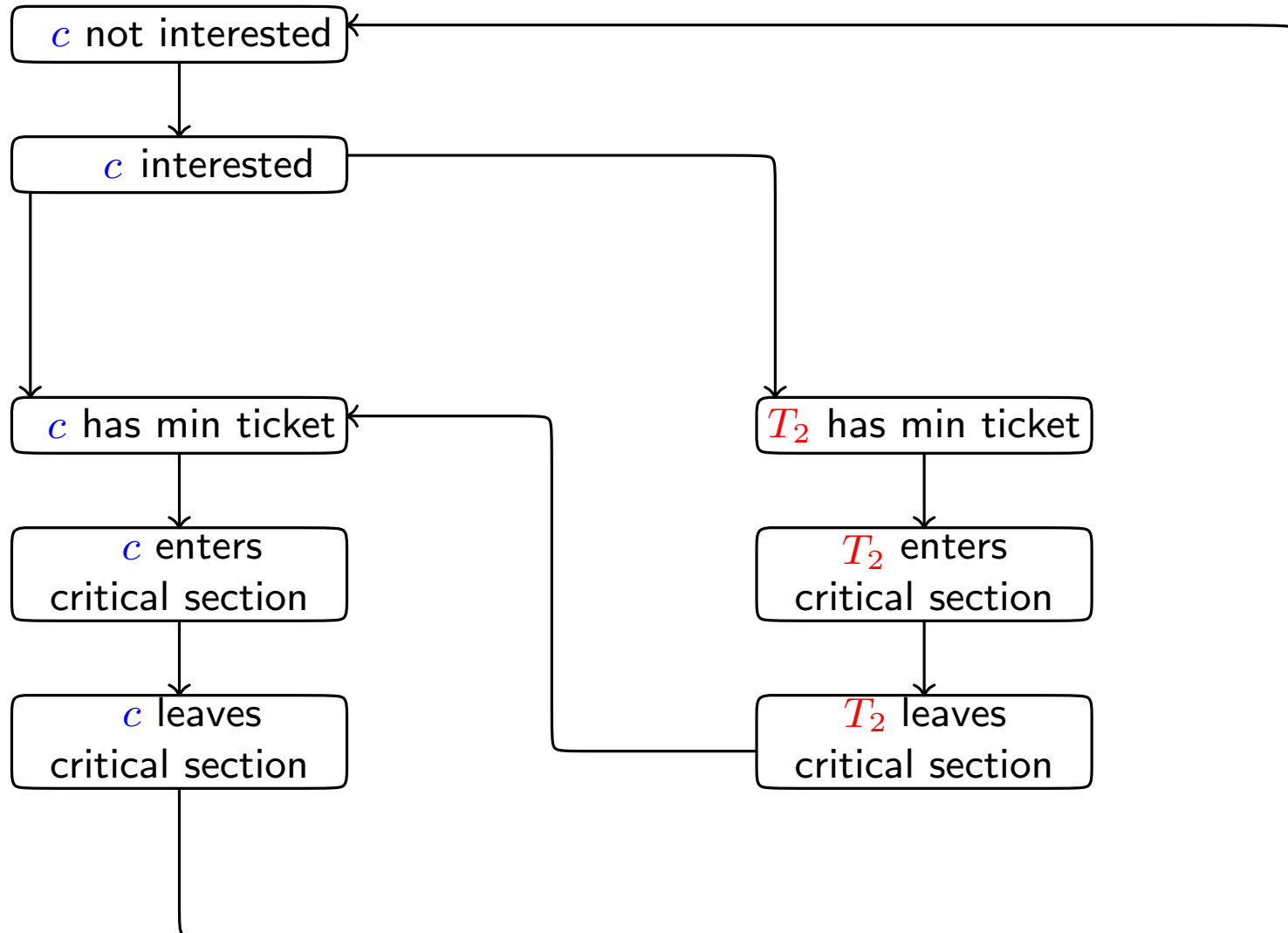
Mutual Exclusion Algorithm (revisited)

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



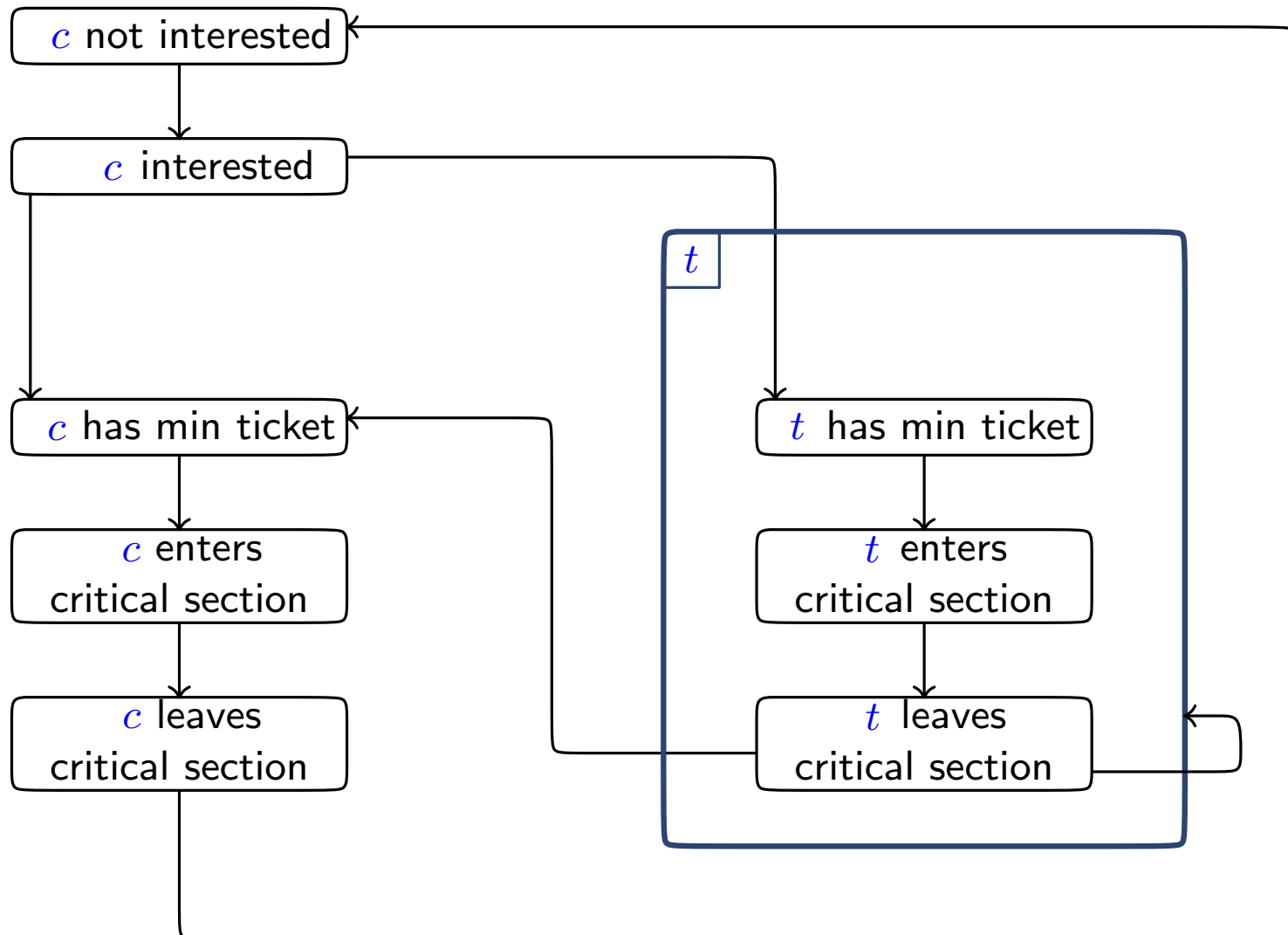
Mutual Exclusion Algorithm (revisited)

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



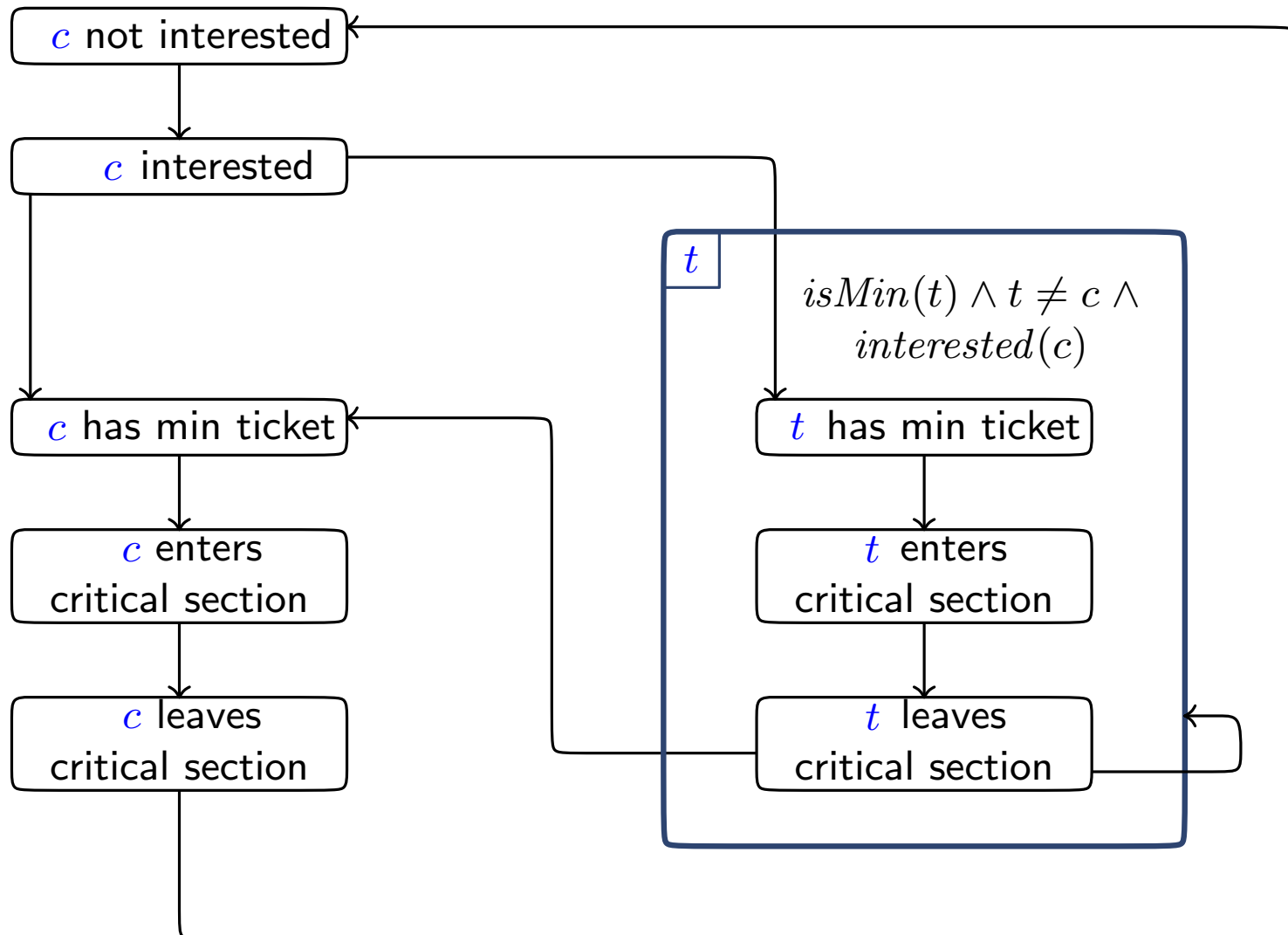
Mutual Exclusion Algorithm (revisited)

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



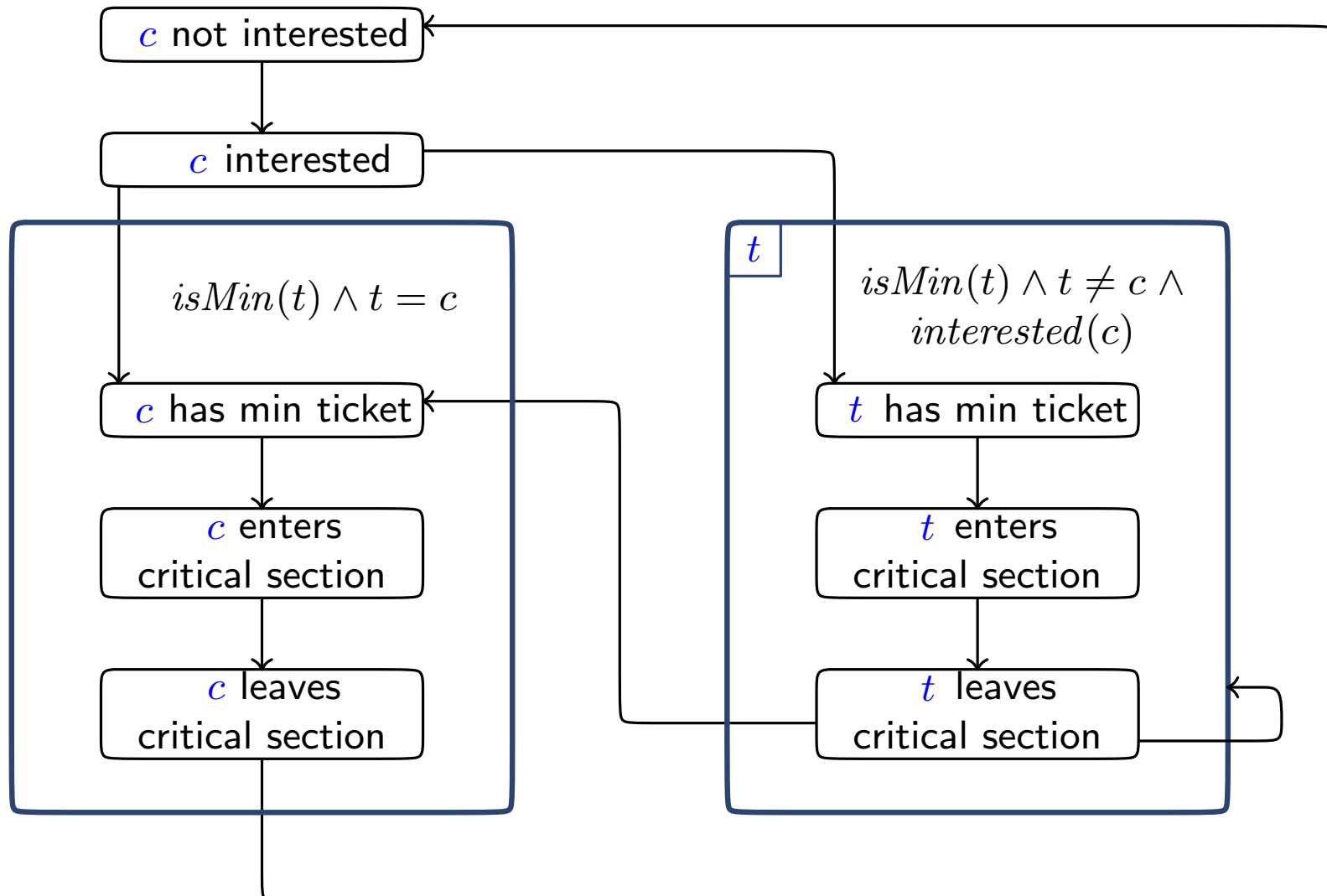
Mutual Exclusion Algorithm (revisited)

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



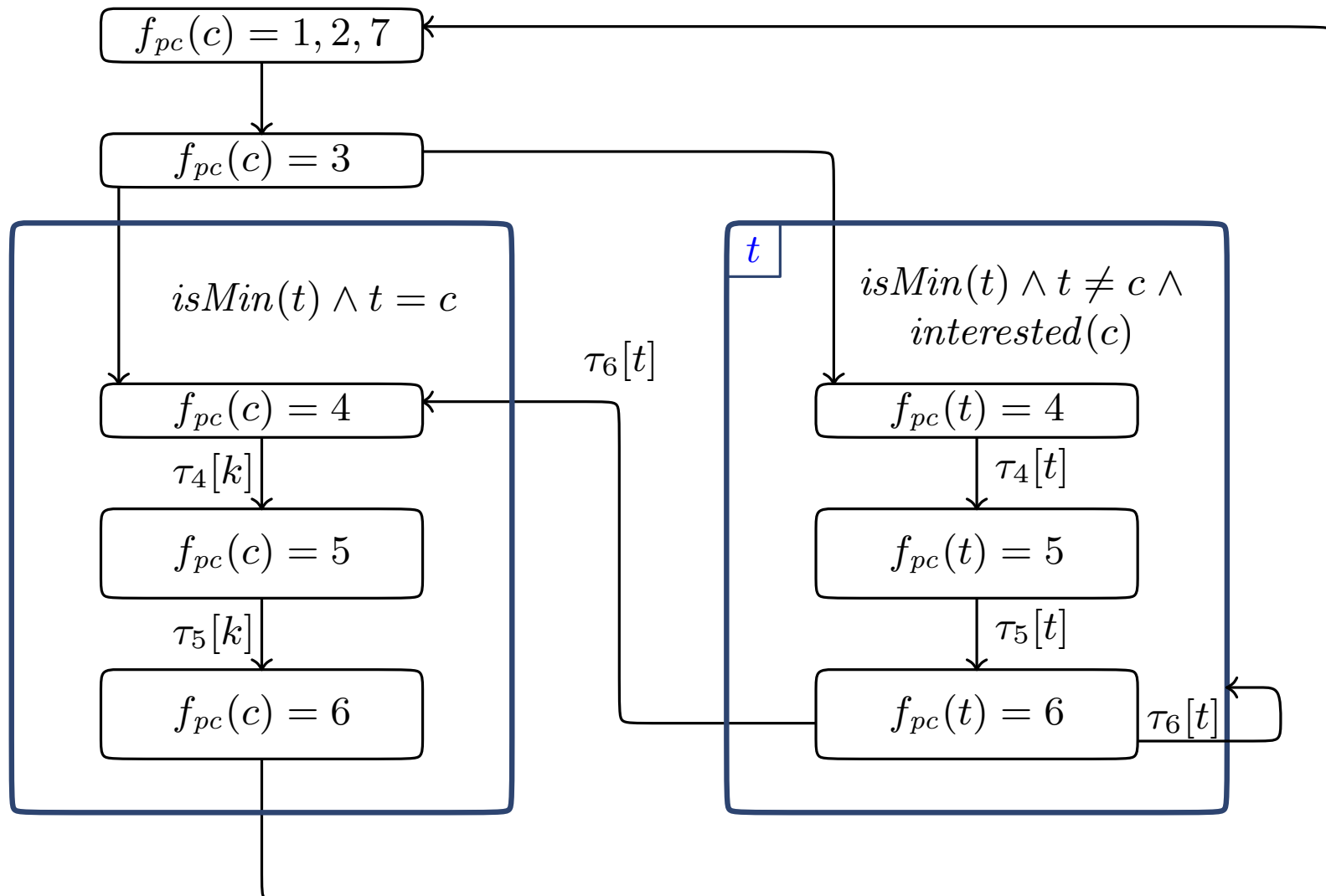
Mutual Exclusion Algorithm (revisited)

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



Mutual Exclusion Algorithm (revisited)

- ▶ For all k , $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access_critical}(k))$
- ▶ 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**
- ▶ Possibility 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