Deductive Temporal Verification of Parametrized Concurrent Systems

Alejandro Sánchez\textsuperscript{1} \hspace{1cm} César Sánchez\textsuperscript{1,2}

\textsuperscript{1}IMDEA Software Institute, Spain
\textsuperscript{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

remove()
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

\[ P[N] : P(1) \parallel \cdots \parallel P(N) \]

Property

\[ \varphi^{(k)} \]
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

Most General Client

\[ P[N] : P(1) \parallel \cdots \parallel P(N) \]

Property

\[ \varphi^{(k)} \]

LTL \( (\Box, \Diamond, U, \ldots) \)
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

Diagram

Property

Most General Client

\[ P[N] : P(1) \| \cdots \| P(N) \]
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

Most General Client

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

Diagram

$\models D$

Verification Conditions:
- Initiation
- Consecution
- Acceptance
- Fairness

Property

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

Satisfaction (Model Checking)
Motivating Example: Mutual Exclusion Algorithm
Motivating Example: Mutual Exclusion Algorithm

global
   Int tick := 0
   Set(Int) announced := Ø

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(Int) announced := \emptyset

procedure MutExc
    Int ticket

begin
    loop
        nondet
        ticket := tick +++
        \langle announced.add(ticket) \rangle
        await (announced.min == ticket)
        critical
        announced.remove(ticket)
    end loop
end procedure
```
Motivating Example: Mutual Exclusion Algorithm

```plaintext
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
```

![Diagram of the mutual exclusion algorithm]

- **tick**: Represents the current tick count.
- **Announced Set**: Tracks the tickets that have been announced.
- **Critical Section**: The section where resources are accessed exclusively.

---

2/11
Motivating Example: Mutual Exclusion Algorithm

global
    Int tick := 0
    Set(Int) announced := ∅

**procedure** MutExc
    Int ticket

begin
1:   loop
2:     nondet
3:     ⟨ ticket := tick ++ 
3:          announced.add(ticket) ⟩
4:     await (announced.min == ticket)
5:     critical
6:     announced.remove(ticket)
7:   end loop
end procedure
global
\[
\begin{align*}
\text{Int } & \; \text{tick} := 0 \\
\text{Set(Int) } & \; \text{announced} := \emptyset
\end{align*}
\]

procedure MutExc
\[
\begin{align*}
\text{Int } & \; \text{ticket} \\
begiend
1: \; \text{loop} \\
2: \; \text{nondet} \\
3: \; \langle \text{ticket} := \text{tick} + +, \text{announced}.\text{add(ticket)} \rangle \\
4: \; \text{await} (\text{announced}.\text{min} == \text{ticket}) \\
5: \; \text{critical} \\
6: \; \text{announced}.\text{remove(ticket)} \\
7: \; \text{end loop}
\end{align*}
\]
end procedure
global

\[
\text{Int } \text{tick} := 0
\]

\[
\text{Set(\text{Int}) } \text{announced} := \emptyset
\]

\begin{verbatim}
procedure MutExc
  \text{Int ticket}
  begin
  1: loop
  2: \text{nondet}
  3: \langle
  4: \text{ticket} := \text{tick} + +
  5: \text{announced.add(ticket)}
  6: \text{await (announced.min} == \text{ticket)}
  7: \text{critical}
  8: \text{announced.remove(ticket)}
  9: \text{end loop}
  end procedure
\end{verbatim}
Motivating Example: Mutual Exclusion Algorithm

global
\[ \text{Int } \text{tick} := 0 \]
\[ \text{Set(} \text{Int} \text{) } \text{announced} := \emptyset \]

procedure MutExc
\[ \text{Int } \text{ticket} \]
begin
1: \text{loop}
2: \text{nondet}
3: \begin{cases}
\text{ticket} := \text{tick} + + \\
\text{announced}.\text{add}() \\
\end{cases}
4: \text{await} (\text{announced}.\text{min} == \text{ticket})
5: \text{critical}
6: \text{announced}.\text{remove}() \\
7: \text{end loop}
end procedure
Motivating Example: Mutual Exclusion Algorithm

global

\[ \text{Int } \text{tick} := 0 \]
\[ \text{Set(Int) announced} := \emptyset \]

\textbf{procedure MutExc}

\[ \text{Int ticket} \]

\textbf{begin}

1: \textbf{loop}

2: \textbf{nondet}

3: \begin{align*}
& \text{ticket} := \text{tick} + + \\
& \text{announced.add(ticket)}
\end{align*}

4: \textbf{await} \ (\text{announced.min} == \text{ticket})

5: \textbf{critical}

6: \text{announced.remove(ticket)}

7: \textbf{end loop}

\textbf{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
```
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 all $k$, $\varphi(k) = \square(announced(k) \rightarrow \diamond access\_critical(k))$
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 \( T_2 \)
Verification Diagram for Mutual Exclusion Algorithm

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

![Diagram showing $T_1$ not interested and $T_1$ 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 \( k \), \( \varphi(k) = \Box(\text{announced}(k) \rightarrow \Diamond \text{access\_critical}(k)) \)
- Let’s assume a system with 2 threads: \( T_1 \) and \( T_2 \)
- Let’s verify \( \varphi(T_1) \)
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 $k$, $\varphi(k) = \Box(\text{announced}(k) \rightarrow \Diamond \text{access_critical}(k))$
- 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 $\varphi(T_1)$

Some verification conditions

\begin{align*}
  n \land \tau[1] & \rightarrow \text{succ}(n) \\
  n \land \tau[2] & \rightarrow \text{succ}(n)
\end{align*}
Verification Diagram for Mutual Exclusion Algorithm

- For all $k$, $\varphi(k) = \square(\text{announced}(k) \rightarrow \Diamond \text{access}_{\text{critical}}(k))$
- 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

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

$n \land \tau[2] \rightarrow \text{succ}(n)$
Verification Diagram for Mutual Exclusion Algorithm

- For all \( k \), \( \varphi(k) = □(announced(k) → ◇access\_critical(k)) \)
- 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

\[
\begin{align*}
n \land \tau[1] & \rightarrow \text{succ}(n) \\
n \land \tau[2] & \rightarrow \text{succ}(n)
\end{align*}
\]
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 $\varphi(T_1)$
- Imagine now a system with 3 threads: $T_1$, $T_2$ and $T_3$

Some verification conditions

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

$n \land \tau[2] \rightarrow \text{succ}(n)$
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 $\varphi(T_1)$
- Imagine now a system with 3 threads: $T_1$, $T_2$ and $T_3$

Some verification conditions

- $n \land \tau[1] \rightarrow \text{succ}(n)$
- $n \land \tau[2] \rightarrow \text{succ}(n)$
- $n \land \tau[3] \rightarrow \text{succ}(n)$
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 symmetric 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 $P$

\[ S^{[M]} = \langle V, \Theta, T, J \rangle \]
Let $P$ be a program consisting of $L$ lines of code.

Assuming $M$ threads running program $P$.

Let $V_{\text{global}}$ be the set of global variables of program $P$.

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

\[ V = V_{\text{global}} \]
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 $P$

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

\[ V = V_{global} \cup (V_{local} \times [M]) \]
Let $P$ be a program consisting of $L$ lines of code.
Assuming $M$ threads running program $P$.
Let $V_{\text{global}}$ be the set of global variables of program $P$.
Let $V_{\text{local}}$ be the set of local variables of program $P$.

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

$$V = V_{\text{global}} \cup (V_{\text{local}} \times [M]) \cup pc[M]$$
Parametrized Fair Transition Systems

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

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

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

\[ \mathcal{T} = \bigcup_{l \in 1..L} \bigcup_{i \in [M]} \tau_l[i] \]
Parametrized Fair Transition Systems

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

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

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

For $S^{[2]}$
```
Let $P$ be a program consisting of $L$ lines of code

Assuming $M$ threads running program $P$

Let $V_{\text{global}}$ be the set of global variables of program $P$

Let $V_{\text{local}}$ be the set of local variables of program $P$

$$\text{Int} \ \text{tick} := 0$$
$$\text{Set}(\text{Int} \ \text{announced}) := \emptyset$$

procedure MutExc

begin

1: loop

2: nondet

3: $\langle $ $\langle \text{ticket} := \text{tick} + +$

4: $\langle $ $\langle \text{announced}.\text{add}($ticket$) \rangle$

5: await ($\langle \text{announced}.\text{min} ==$ ticket$) \rangle$

6: critical

7: $\langle \text{announced}.\text{remove}(\text{ticket}) \rangle$

end loop

end procedure

For $S^{[2]}$

$V = \{\text{tick, announced}\}$
Let $P$ be a program consisting of $L$ lines of code

Assuming $M$ threads running program $P$

Let $V_{\text{global}}$ be the set of global variables of program $P$

Let $V_{\text{local}}$ be the set of local variables of program $P$

```
Int tick := 0
Set\langle Int\rangle announced := \emptyset

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

For $S^{[2]}$

$$V = \{\text{tick, announced}\} \cup \{\text{ticket[1], ticket[2]}\}$$
Parametrized Fair Transition Systems

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

```
Int tick := 0
Set\langle Int\rangle \text{ announced} := \emptyset

procedure \text{MutExc}
  Int ticket
begin
  loop
    nondet
    \langle ticket := tick ++
    announced.add(ticket) \rangle
    await (announced.min == ticket)
  critical
  announced.remove(ticket)
end loop
end procedure
```

For $S^{[2]}$

$V = \{\text{tick, announced}\} \cup \{\text{ticket[1], ticket[2]}\} \cup \{\text{pc[1], pc[2]}\}$
Let $P$ be a program consisting of $L$ lines of code

Assuming $M$ threads running program $P$

Let $V_{\text{global}}$ be the set of global variables of program $P$

Let $V_{\text{local}}$ be the set of local variables of program $P$

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

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

For $S^{[2]}$

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

\[
\mathcal{T} = \bigcup_{l \in 1..7} \{\tau_l[1], \tau_l[2]\}
\]
Let \( P \) be a program consisting of \( L \) lines of code

Assuming \( M \) threads running program \( P \)

Let \( V_{\text{global}} \) be the set of global variables of program \( P \)

Let \( V_{\text{local}} \) be the set of local variables of program \( P \)

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

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 \( S^{[2]} \)

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

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

\[
\mathcal{J} = \mathcal{T}
\]
Symmetric Systems
Symmetric Systems

- Assume a parametrized transition system $S^M$
- All threads execute the same program
- Only equality and inequality between thread identifiers
Symmetric Systems

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

\[
\sigma = s_0 \quad s_1 \quad s_2 \quad \ldots
\]
Symmetric Systems

- Assume a parametrized transition system $S^M$
- All threads execute the same program
- Only equality and inequality between thread identifiers

$$\sigma = s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} \ldots$$
Symmetric Systems

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

\[
\sigma = s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} \ldots
\]

\[
i, j \in [M]
\]
Symmetric Systems

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

$$
\sigma = S_0 \xrightarrow{\tau_0} S_1 \xrightarrow{\tau_1} S_2 \xrightarrow{\tau_2} \ldots
$$

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

- Assume a parametrized transition system $S^M$
- All threads execute the same program
- Only equality and inequality between thread identifiers

\[ \sigma = S_0 \xrightarrow{\tau_0} S_1 \xrightarrow{\tau_1} S_2 \xrightarrow{\tau_2} \ldots \]

\[ i \leftrightarrow j \]

\( i, j \in [M] \)  
switch \( v[i] \) with \( v[j] \)

switch \( \tau[i] \) with \( \tau[j] \)
Symmetric Systems

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

\[
\begin{align*}
\sigma & = \quad S_0 \xrightarrow{\tau_0} S_1 \xrightarrow{\tau_1} S_2 \xrightarrow{\tau_2} \ldots \\
\sigma^{i \leftrightarrow j} & = \quad S_0^{i \leftrightarrow j} \xrightarrow{\tau_0^{i \leftrightarrow j}} S_1^{i \leftrightarrow j} \xrightarrow{\tau_1^{i \leftrightarrow j}} S_2^{i \leftrightarrow j} \xrightarrow{\tau_2^{i \leftrightarrow j}} \ldots \\
\end{align*}
\]

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

\[\text{switch } v[i] \text{ with } v[j]\]

\[\text{switch } \tau[i] \text{ with } \tau[j]\]
Parametrized Verification Diagrams

- Prove that all instances of $S^{[M]}$ satisfy a temporal specification with a unique diagram.
Parametrized Verification Diagrams

- Prove that all instances of $S^M$ satisfy a temporal specification with a unique diagram

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

- Prove that all instances of $S^M$ satisfy a temporal specification with a unique diagram

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

- For each $v : \alpha$, we add $f_v : \text{tid} \to \alpha$
  $f_{pc} : \text{tid} \to \text{Loc}$

- $T_{\text{param}}$ is:
  - stable infinite
  - polite
Parametrized Verification Diagrams

▶ Prove that all instances of $S^M$ satisfy a temporal specification with a unique diagram

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

▶ For each $v : \alpha$, we add $f_v : \text{tid} \rightarrow \alpha$

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

▶ $T_{\text{param}}$ is:
  ◀ stable infinite combinations with
  ◀ polite non stable infinite theories

$ T = T_{\text{prog}} + T_{\text{param}} $
Soundness of Parametrized Verification Diagrams

**Theorem:**

Let $S^M$ be a symmetric parametrized FTS and $\varphi(k)$ a temporal formula.

If there exists a $(M, k)$-valid PVD $D^M$, then:

$$S^M \models \varphi(k)$$
Theorem:

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

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

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

for all $M$
Parametrized Verification Diagrams

- PVDs are an extension of GVD

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

- PVDs are an extension of GVD

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

- PVDs are an extension of GVD

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

- PVDs are an extension of GVD

\[ D[M] : \langle N, N_0, B, E, \mu, F, \eta, \Delta, f \rangle \]
Parametrized Verification Diagrams

- PVDs are **an extension** of GVD

\[ D[M] : \langle N, N_0, B, E, \mu, F, \eta, \Delta, f \rangle \]
Parametrized Verification Diagrams

- PVDs are an extension of GVD

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

- PVDs are an extension of GVD

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

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

\[ D[^M] : \langle N, N_0, B, E, \mu, F, \eta, \Delta, f \rangle \]
Parametrized Verification Diagrams

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

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

For \( M \) threads

```
3
```

```
2
```

```
1
```

represents

```
3
```

```
2
```

```
1
```
Parametrized Verification Diagrams

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

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

For \( M \) threads

![Diagram](image)
Parametrized Verification Diagrams

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

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

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

For \( M \) threads
Parametrized Verification Diagrams

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

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

For \( M \) threads

For \( M \) threads
PVDs are an extension of GVD

We add the notion of boxes

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

For \( M \) threads

represents
Parametrized Verification Diagrams

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

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

For \( M \) threads

```
represents
```

![Diagram showing the parametrized verification diagram for \( M \) threads]
Parametrized Verification Diagrams

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

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

For \( M \) threads
Parametrized Verification Diagrams

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

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

For \( M \) threads

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

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

For \( M \) threads

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 $\tau \in T$,

  $$n \land \tau \rightarrow \text{succ}(n)$$
Verification Conditions for Parametrized Diagrams

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

\[
\forall l \forall i \quad n \land \tau_l[i] \rightarrow \text{succ}(n)
\]
Verification Conditions for Parametrized Diagrams

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

- **Consecution:** For every \( n \in N \) and \( \tau \in \mathcal{T} \),

\[
    n \land \tau \rightarrow \text{succ}(n)
\]

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

only tid appearing in \( n \) and \( \text{succ}(n) \) are relevant
Verification Conditions for Parametrized Diagrams

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

- **Consecution:** For every $n \in N$ and $\tau \in T$,

\[
\forall i, n \land \tau[i] \rightarrow succ(n)
\]

\[
\forall l \land_i \forall n \land \tau[l][i] \rightarrow succ(n)
\]

$\forall n, suc(n)$ = $\{i_1, \ldots, i_q\}$ = $I$

"tid appearing on $n$ and $suc(n)$"
Verification Conditions for Parametrized Diagrams

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

\[
\begin{align*}
    n \land \tau & \rightarrow \text{succ}(n) \\
\end{align*}
\]

\[
\bigwedge_l \bigwedge_i n \land \tau_l[i^*] \rightarrow \text{succ}(n)
\]

\[
\text{Voc}(n, \text{succ}(n)) = \{i_1, \ldots, i_q\} = I
\]

\[
\begin{align*}
    \bigwedge_l n \land \tau_l[i_1] & \rightarrow \text{succ}(n) \\
    \vdots & \quad \vdots \\
    \bigwedge_l n \land \tau_l[i_q] & \rightarrow \text{succ}(n)
\end{align*}
\]
Veriﬁcation Conditions for Parametrized Diagrams

▶ Initialization: \( \Theta \to \mu(N_0) \)

▶ Consecution: For every \( n \in N \) and \( \tau \in \mathcal{T} \),

\[
\begin{align*}
n \land \tau & \to succ(n) \\
\prod_l \prod_i n \land \tau_l[i] & \to succ(n) \\
Voc(n, succ(n)) &= \{i_1, \ldots, i_q\} = I \\
\prod_l n \land \tau_l[i_1] & \to succ(n) \\
\vdots & \vdots \\
\prod_l n \land \tau_l[i_q] & \to succ(n)
\end{align*}
\]
Verification Conditions for Parametrized Diagrams

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

- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

\[
\begin{align*}
\forall l \quad \forall i \quad n \land \tau_l[i] & \rightarrow succ(n) \\
Voc(n, succ(n)) & = \{i_1, \ldots, i_q\} = I \\
\forall l \quad n \land \tau_l[i_1] & \rightarrow succ(n) \\
\vdots & \\
\forall l \quad n \land \tau_l[i_q] & \rightarrow succ(n) \\
\forall l \quad n \land \tau_l[i] & \land \bigwedge_{j \in I} i \neq j \rightarrow succ(n)
\end{align*}
\]

abstracts all other cases thanks to symmetry
Verification Conditions for Parametrized Diagrams

- **Initialization:** \( \Theta \rightarrow \mu(N_0) \)
- **Conseution:** For every \( n \in N \) and \( \tau \in T \),

\[
\begin{align*}
\Lambda_l \, \Lambda_i \, n \land \tau_l[i] & \rightarrow \text{succ}(n) \\
Voc(n, \text{succ}(n)) & = \{i_1, \ldots, i_q\} = I
\end{align*}
\]

\[
\begin{align*}
\Lambda_l \, n \land \tau_l[i_1] & \rightarrow \text{succ}(n) \\
& \vdots \\
\Lambda_l \, n \land \tau_l[i_q] & \rightarrow \text{succ}(n) \\
\Lambda_l \, n \land \tau_l[i] & \land \bigwedge_{j \in I} i \neq j \rightarrow \text{succ}(n)
\end{align*}
\]

\( \mathbf{L} \times \mathbf{M} \) \hspace{1cm} \( \mathbf{L} \times (q + 1) \)

verification conditions
Verification Conditions for Parametrized Diagrams

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

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

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

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

\[
\begin{align*}
(a) & \quad \begin{bmatrix} \rho_{\tau[i]}(s,s') \land \\
\mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \quad \rightarrow \delta_{j,n_1}(s) \succeq \delta_{j,n_2}(s') \quad , \text{for each } i \in I \\
(b) & \quad \begin{bmatrix} \rho_{\tau[i]}(s,s') \land \bigwedge_{j \in I} i \neq j \land \\
\mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \quad \rightarrow \delta_{j,n_1}(s) \succeq \delta_{j,n_2}(s')
\end{align*}
\]

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

\[
\begin{align*}
(a) & \quad \begin{bmatrix} \rho_{\tau[i]}(s,s') \land \\
\mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \quad \rightarrow \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s') \quad , \text{for each } i \in I \\
(b) & \quad \begin{bmatrix} \rho_{\tau[i]}(s,s') \land \bigwedge_{j \in I} i \neq j \land \\
\mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \quad \rightarrow \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s')
\end{align*}
\]

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

\[
\begin{align*}
(F1) & \quad \mu(n_1)(s) \land \tau[i] \in \eta(e) \quad \rightarrow En(\tau[i]) \\
(F2) & \quad \mu(n_1)(s) \land \tau[i] \in \eta(e) \land \rho_{\tau[i]}(s,s') \rightarrow \mu(\tau[i](n_1))(s')
\end{align*}
\]
Verification Conditions for Parametrized Diagrams

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

- **Consecution:** For every $n \in N$, let $I = Voc(n, next(n))$,
  
  (C1) $\mu(n)(s) \land \rho_{\tau[i]}(s, s') \rightarrow \mu(next(n))(s')$, for each $i \in I$

  (C2) $\mu(n)(s) \land \rho_{\tau[i]}(s, s') \land \land_{j \in I} i \neq j \rightarrow \mu(next(n))(s')$

- **Acceptance:** If $(n_1, n_2) \in P \setminus R$, let $I = Voc(n_1, n_2)$,
  
  (a) $\begin{bmatrix} \rho_{\tau[i]}(s, s') \land \\ \mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \rightarrow \delta_{j,n_1}(s) \geq \delta_{j,n_2}(s')$, for each $i \in I$

  (b) $\begin{bmatrix} \rho_{\tau[i]}(s, s') \land \land_{j \in I} i \neq j \land \\ \mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \rightarrow \delta_{j,n_1}(s) \geq \delta_{j,n_2}(s')$

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

  (a) $\begin{bmatrix} \rho_{\tau[i]}(s, s') \land \\ \mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \rightarrow \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s')$, for each $i \in I$

  (b) $\begin{bmatrix} \rho_{\tau[i]}(s, s') \land \land_{j \in I} i \neq j \land \\ \mu(n_1)(s) \land \mu(n_2)(s') \end{bmatrix} \rightarrow \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s')$

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

  (F1) $\mu(n_1)(s) \land \tau[i] \in \eta(e) \rightarrow En(\tau[i])$

  (F2) $\mu(n_1)(s) \land \tau[i] \in \eta(e) \land \rho_{\tau[i]}(s, s') \rightarrow \mu(\tau[i](n_1))(s')$
Mutual Exclusion Algorithm (revisited)

- For all $k$, $\varphi(k) = \Box(\text{announced}(k) \rightarrow \Diamond \text{access\_critical}(k))$
Mutual Exclusion Algorithm (revisited)

- For all $k$, $\varphi(k) = \Box(\text{announced}(k) \rightarrow \Diamond \text{access\_critical}(k))$
- By symmetry: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$
Mutual Exclusion Algorithm (revisited)

- For all $k$, $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access}_{\text{critical}}(k))$
- By symmetry: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$
For all $k$, $\varphi(k) = \Box(\text{announced}(k) \rightarrow \Diamond\text{access critical}(k))$

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 $k$, $\varphi(k) = \square(\text{announced}(k) \rightarrow \diamond \text{access}_{\text{critical}}(k))$
- By symmetry: $\varphi(c)$ for arbitrary $c \in [M]$, implies $\varphi(k)$, $\forall k \in [M]$
Mutual Exclusion Algorithm (revisited)

- For all $k$, $\varphi(k) = \Box(\text{announced}(k) \rightarrow \Diamond \text{access\_critical}(k))$
- 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 \( \varphi(k), \forall k \in [M] \)

\[
\begin{align*}
\text{isMin}(t) \land t = c & \quad \text{for arbitrary } c \in [M] \\
f_{pc}(c) = 1, 2, 7 & \\
f_{pc}(c) = 3 & \\
isMin(t) \land t \neq c \land \text{interested}(c) & \\
f_{pc}(c) = 4 \\
\tau_4[k] \\
f_{pc}(c) = 5 \\
\tau_5[k] \\
f_{pc}(c) = 6 & \\
\end{align*}
\]
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