The Method of Network Invariants

Consider a parameterized FDS (fair discrete system), \( S(n) \). We wish to verify property \( P \) for which \( d \equiv (u)(S(n)) \) for every \( n \). Assume that \( P \) are identical (up to renaming) and that processes \( P_1, \ldots, P_k \) are observable to the observable variables of \( P_1 \). We propose the following strategy ([BCC86], [WLR89], [SC89], [CJ95]):

1. Device a network invariant \( I \), which is an FDS intended to provide a modular abstraction for the parallel composition \( P_1 \parallel \cdots \parallel P_k \).

2. Confirm that \( I \) is indeed a network invariant by verifying that \( (I)_{P_1 \parallel \cdots \parallel P_k} \subseteq (I)_{P_1} \). and that \( (I)_{P_1} \subseteq (I)_{P_1} \).

3. Model check \( (I)_{P_1} \parallel \cdots \parallel (I)_{P_k} \).

4. Conclude that \( S(n) \parallel \cdots \parallel (u)(S(n)) \) for every \( n < 1 \).
Consider a process in program $MUX-SEM$:

Assume that among the system variables, we distinguish $\mathcal{O} \overset{\text{Observable variables}}{\leftarrow} \Lambda$, the set of owned variables, and $\mathcal{W}$, the set of owned variables.

An open computation of a process consists of an infinite sequence of states obtained by strict interleaving of environment and system steps. A system step follows the transition relation. An environment step may change all variables except for the owned variables.

Consider the process of this program:

Following is an open computation of this process:

\[
\begin{align*}
\langle I: \bar{r}, L: \psi \rangle & \leftrightarrow S \langle I: \bar{r}, N: \psi \rangle \\
\langle I: \bar{r}, C: \psi \rangle & \leftarrow S \langle I: \bar{r}, \mathcal{O}: \psi \rangle \\
\langle I: \bar{r}, \mathcal{O}: \psi \rangle & \leftarrow E \langle I: \bar{r}, L: \psi \rangle \\
\langle I: \bar{r}, \mathcal{O}: \psi \rangle & \leftarrow E \langle I: \bar{r}, N: \psi \rangle \\
\langle I: \bar{r}, \mathcal{O}: \psi \rangle & \leftarrow E \langle I: \bar{r}, C: \psi \rangle
\end{align*}
\]
We refer to $\mathcal{A}$ and $\mathcal{A}^\varphi$ as the concrete and abstract systems, respectively.

\[
\forall \mathcal{A} \text{ and } \mathcal{A}^\varphi, (\varphi)\mathcal{A}\mathcal{O}^\varphi \subseteq (\varphi)\mathcal{A}\mathcal{O}.
\]

If

\[
\forall \mathcal{A} \mathcal{O} \subseteq \mathcal{A}
\]

The FDS $\mathcal{A}$ is a (modular) abstraction of the comparable FDS $\mathcal{A}$, denoted by $\mathcal{A}$. The FDS $\mathcal{A}$ is defined to be comparable with the FDS $\mathcal{A}$ if $\forall \mathcal{A} \mathcal{O} = \mathcal{A} \mathcal{O}$. For the observables of $\mathcal{A}$, let $\mathcal{O}$ denote the set of all observations of system $\mathcal{A}$, where $\mathcal{O}$ is the projection of state $s$, on $s_0$:

\[
\mathcal{O} = \mathcal{O}^\varphi
\]

An open computation of the FDS $\mathcal{A}$ is an open computation of the FDS $\mathcal{A}$. For $\mathcal{O}$:

\[
\mathcal{O}^\varphi
\]

Modular Abstraction
The following can be shown:

Properties of the Modular Abstraction Relation

- Modular abstraction is reflexive and transitive.
- Modular abstraction is property restricting. That is, if \( D \subseteq_{\text{prop}} M D A \) then \( \forall a . D \sqsubseteq_{\text{prop}} (\emptyset \parallel a) \sqsubseteq_{\text{prop}} (\emptyset \parallel a) \).
- Modular abstraction is compositional. That is, whenever \( \forall a . D \sqsubseteq_{\text{prop}} \bigcirc \forall a \sqsubseteq_{\text{prop}} \bigcirc \forall a \).

\[ \vdash p \implies \bigcirc A \quad \text{implies} \quad \vdash \bigcirc A \]

A. Pnueli

Advanced Topics in Reactive Verification, NYU, Spring, 2004
There are two properties we wish to verify for program $\text{MUX-SEM}(u)$.

### Application to Program $\text{MUX-SEM}(u)$

As a first step, we abstract into:

\[ \phi \vdash \Box (C_1 \triangleleft 1) \Box : \text{acc} \phi \quad \text{and} \quad (0 = \phi \leftarrow 1) \Box : \text{sem} \phi \]

There are two properties we wish to verify for program $\text{MUX-SEM}(u)$. 

\[ \phi \vdash \Box (C_1 \triangleleft 1) \Diamond : \text{req} \phi \]

### Network Invariants

A. Pnueli

**Application to Program $\text{MUX-SEM}(u)$**

As a first step, we abstract into:

\[ \phi \vdash \Box (C_1 \triangleleft 1) \Box : \text{acc} \phi \quad \text{and} \quad (0 = \phi \leftarrow 1) \Box : \text{sem} \phi \]
Thus, the sequence $I^1, I^2, I^3, \ldots$ does not converge.

Three times in succession, critical section twice in succession. Presumably, $I^3$ will be able to enter and exit in this behavior, $I^2$ enters a critical section twice in succession, and then exits a critical section twice in succession. Presumably, $I^3$ will be able to enter and exit in this behavior, $I^2$ enters a critical section twice in succession, and then exits a critical section twice in succession.

The following modular observation can be generated by $I^2$ but not by $I^1$:

\[
\begin{align*}
\text{enter} & \quad \langle 0 : \bar{I} \rangle \leftarrow \langle 1 : \bar{I} \rangle \leftarrow \langle 0 : \bar{I} \rangle \\
\text{exit} & \quad \langle 1 : \bar{I} \rangle \leftarrow \langle 0 : \bar{I} \rangle \leftarrow \langle 1 : \bar{I} \rangle
\end{align*}
\]

Fails in the current case!

and hoping that it converges.

\[
\begin{align*}
\text{enter} & \quad \langle 0 : \bar{I} \rangle \leftarrow \langle 1 : \bar{I} \rangle \leftarrow \langle 0 : \bar{I} \rangle \\
\text{exit} & \quad \langle 1 : \bar{I} \rangle \leftarrow \langle 0 : \bar{I} \rangle \leftarrow \langle 1 : \bar{I} \rangle
\end{align*}
\]

\[
\begin{align*}
I^1 & \parallel I^2 \quad \langle 1 : \bar{I} \rangle \leftarrow \langle 0 : \bar{I} \rangle \\
I^3 & \parallel I^2 \quad \langle 1 : \bar{I} \rangle \leftarrow \langle 0 : \bar{I} \rangle \\
I^3 & \parallel I^2 \quad \langle 1 : \bar{I} \rangle \leftarrow \langle 0 : \bar{I} \rangle
\end{align*}
\]

A useful heuristic which often works attempts to solve the fix-point inclusions:

**A Failure of a Useful Heuristic**
The two properties.

It can be verified that for any system transition, the chaotic behavior allows any state sequence, with arbitrary values of $y$ even.

The distinctions among the different approximations is observed only under an environment.

Introduce an abstraction which discards improper behavior of the environment.

Order is restored by Chaos.
The above abstraction can be viewed as automata (FSA) representation of an assume-guarantee specification of the form: Assume-Guarantee Specification.

The system behaves according to the non-chaotic part of the diagram spanning states $\langle C, 0 \rangle$, $\langle 0, 1 \rangle$, $\langle N, 1 \rangle$, and $\langle 0, 0 \rangle$. The environment never raises $\alpha$ while the system is at $C$. This is because the implication is logically equivalent to:

\[
\neg \exists \alpha. \neg \forall \alpha. \neg \forall \alpha.
\]

Where:

\[
\neg \exists \alpha. \leftrightarrow \neg \exists \alpha.
\]
Consider $n$ philosophers arranged around a table.

The life of a philosopher alternates between a thinking phase (a non-critical activity) and an eating phase. In order to eat, a philosopher needs both forks.
It is not difficult to verify the following safety property:

\[
\begin{array}{l}
[1 \cup f] f \quad \text{release} : 6_j \\
\vdots \\
\sqrt{\text{Critical}} : 4_j \\
[1 \cup f] f \quad \text{request} : 3_j \\
\vdots \\
\sqrt{\text{Non-Critical}} : 1_j \\
\text{loop forever do} : 0_j
\end{array}
\]

A first attempt yields the following program:

\[
\begin{array}{l}
\text{Dine:} \\
\text{Program Dine}
\end{array}
\]
Unfortunately, Dine Cannot Ensure Accessibility for $P[I]$, Specifiable by

Because all philosophers may deadlock together:

$$\Diamond \quad \square \quad \text{(at } \neg \phi_3^{[1]} \text{)}$$

$P'$ $P_2$ $P_3$ $P_4$ $P_5$ $P_6$

Accessibility not Guaranteed
Wish to establish accessibility, expressible by

\[ \phi, \text{acc} \]

Solution: One Contrary Philosopher

\[
\begin{array}{c}
\text{Critical} \\
\text{Non-Critical}
\end{array}
\]

local array \( f \) of natural initially \( f_0 \): loop forever do

1.\ \text{Non-Critical}
2.\ \text{request} \( f[j] \)
3.\ \text{request} \( f[n_1] \)
4.\ \text{Critical}
5.\ \text{release} \( f[j] \)
6.\ \text{release} \( f[n_1] \)

\[
\begin{array}{c}
\text{Critical} \\
\text{Non-Critical}
\end{array}
\]

local array \( f \) of natural initially \( f_0 \): loop forever do

1.\ \text{Non-Critical}
2.\ \text{request} \( f[1] \)
3.\ \text{request} \( f[n] \)
4.\ \text{Critical}
5.\ \text{release} \( f[1] \)
6.\ \text{release} \( f[n] \)
A natural abstraction with hand-crafted proof.

A hand-crafted abstraction with simple proof.

There is a tradeoff:

We will consider two approaches to the construction of the desired invariant.

By a corresponding (left, right)-observation of \( I(\text{left}, \text{right}) \), for every \( k \geq 2 \) this means that any (left, right)-observation of \( S[2..k] \) is matched.

\[
\begin{array}{c}
\begin{array}{c}
\text{array } [2..k] \text{ of boolean where } f_i = 1 \text{ for every } i \geq 2.
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{in-out left \_ right \_ boolean where left = right = 1.}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{local f : } [2..k] \text{ where } f_i = 1.
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{[left, right] } P(\text{left}, \text{f}[2]) P(\text{f}[2], \text{f}[3]) \cdots P(\text{f}[k], \text{right}).
\end{array}
\end{array}
\]

We are searching for an invariant \( I(\text{left}, \text{right}) \) which is a modular abstraction of the philosophers string.

We are searching for a system \( P(\text{left}, \text{right}) \) which can be viewed as a system in which the semaphores left and right are the only observables.

A regular philosopher can be viewed as a system where the

**Designing a Network Invariant**
Lecture 7: Network Invariants

A. Pnueli

An Ad-Hoc Hand-Crafted Construction

Observing how the two border members of \( S \) manipulate the observables

\[
I^N \models (I \parallel p) \quad \text{and} \quad I^N \models [2 \cdots 3]S
\]

In this case, the abstraction conditions were:

\[
(\forall j \geq 3 \left( \text{at } j \right) \lor \text{at } j^r)
\]

Additional compassion:

<table>
<thead>
<tr>
<th>m0: Loop forever do</th>
<th>m1: Loop forever do</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Critical</td>
<td>Critical</td>
</tr>
<tr>
<td>Request Right</td>
<td>Release Right</td>
</tr>
<tr>
<td>Idle</td>
<td>Skip</td>
</tr>
<tr>
<td>Critical</td>
<td>Request Right</td>
</tr>
<tr>
<td>Non-Critical</td>
<td>Release Left</td>
</tr>
<tr>
<td>Idle</td>
<td>Release Left</td>
</tr>
<tr>
<td>Critical</td>
<td>Request Right</td>
</tr>
<tr>
<td>Non-Critical</td>
<td>Release Left</td>
</tr>
<tr>
<td>Idle</td>
<td>Request Right</td>
</tr>
<tr>
<td>Critical</td>
<td>Loop forever do</td>
</tr>
</tbody>
</table>

Advanced Topics in Reactive Verification, NYU, Spring, 2004
Using the Network Invariant

To show that an arbitrary regular philosopher never starves, it is sufficient to verify

\[ \text{where } R \text{ is a contrary philosopher and the locations in the property refer to } P. \]

\[ (\Diamond \Leftarrow \Box) \quad \text{at location } 4 \]

\[ (R \parallel I \parallel P \parallel I) \]
Consider a concrete and abstract systems:

\[ V \mathcal{A} \equiv \mathcal{A} \]

Verifying

Proof.

Streett automaton is hopelessly high. Thus is why we have to resort to a deductive algorithmically solved. However, the complexity of complementing a general Streett automaton, comparing two finite-state systems, can, in theory, be the problem non-deterministic Streett Automaton.

With no loss of generality, assume that there exists a 1

\[ \emptyset = V \Lambda \cup \Phi \Lambda \]

Consider a concrete and abstract systems:

\[ V \mathcal{A} \equiv \mathcal{A} \]
A Proof rule Based on Abstraction Mapping

Let the user devise an abstraction mapping

\[ \forall A \equiv \forall A \]

\[ ([v] \forall \bigwedge \bigcirc \leftarrow [v] \forall \bigwedge \bigcirc, \forall \exists (b \cdot d) \forall \bigcirc \bigwedge \forall \exists \bigwedge \bigcirc \bigwedge \forall \bigcirc \bigwedge \forall \bigcirc ) \implies \forall A \]

A4.

\[ [v] \forall O = \forall O \]

A3.

\[ [v] [v] \forall d \leftarrow \forall d \]

A2.

\[ [v] \forall \Theta \leftarrow \forall \Theta \]

A1.

This rule is adequate for verifying the conditions generated for the two-ends philosophers abstractions.

This following rule can be used to prove model checking:

\[ \forall A \equiv \forall A \]

\[ \forall A \leftarrow [v] \forall \bigwedge \bigcirc \]
A proof rule based on abstraction mapping.

Let the user devise an abstraction mapping \( \varphi \) on the concrete variables and abstract variables. They suggested using history and prophecy variables. However, as shown by Abadi and Lamport [AL91], such a rule is complete only if we allow the mapping \( \varphi \) to depend on the full past and future of the concrete computation. This rule is adequate for verifying the conditions generated for the two-ends philosophers' abstractions.

\[ v A \implies \varphi A \]

\[ ([v]b \diamond \Box \leftarrow [v]d \diamond \Box) v \exists (b \cdot d) v \forall v \exists \Box \]

\[ [v]f \diamond \Box \leftarrow [v]f \diamond \Box \]

\[ [v]O \varphi = \varphi O \]

\[ [v][v]d \leftarrow vO \]

\[ [v]O \leftarrow O \]

\[ [v]A \leftarrow A \]

The following rule can be used to prove model checking.

\[ v A \implies \varphi A \]

Using model checking, \( v A \) – substituting \( \varphi \) for \( v \) in \( \varphi \) – substituting \( \varphi \) variables for the abstract variables, expressing the abstract mapping \( (\varphi A) v = v A \) on abstraction mapping.
This natural mapping is sufficient in order to establish all the abstraction conditions.

\[
\begin{align*}
\left[2\right] \mu &= \left[4\right] \Pi \\
\left[1\right] \mu &= \left[3\right] \Pi \\
\left[\text{Left} = \text{Right} = \text{Left} = \text{Left} \right] : \alpha \\
\end{align*}
\]

An appropriate abstraction mapping is:

\[
\]

To prove \(\left[\right] \overset{w}{\supseteq} [1] \cdot 2\), we have to establish

For Example
An alternative: A Natural Invariant with Complex Proof

\[ S = \mathcal{I} \]

The Alternative: A Natural Invariant with Complex Proof

Obviously, \([d]p_5\) should mimic \([d]p_1\) and \([d]p_6\) should mimic \([d]p_4\) and \([d]p_3\). What should the abstraction mapping be? I.e., that 3 philosophers can faithfully emulate 4 philosophers. What should the string 

\(([d]p_1 \parallel [d]p_2 \parallel [d]p_3) \equiv ([d]p_4 \parallel [d]p_5 \parallel [d]p_6 \parallel [d]p_7)\)

prove that this is an invariant, it is sufficient to establish

An alternate simpler invariant can be obtained by taking the string 

\[ S = \mathcal{I} \]
An alternative simplr invariant can be obtained by taking
\[ I = S^{[1 \cdots 3]} \], i.e., a string
of 3 (unmodiied) philosophers.


To prove that this is an invariant, it is sufficient to establish

\[ \text{The Alternative: A Natural Invariant with Complex Proof} \]
In the "preflop.smv" file, we place:

```
MODULE main
VAR h: boolean -- history variable
CS: system Concrete;
AS: system Abstract;
DEFINE
  deadlock := CS.p1.loc = 2 & CS.p2.loc = 2 &
  CS.p3.loc = 2 & CS.p4.loc = 2 &
  !CS.mid1 & !CS.mid2 & !CS.mid3 & !CS.right;
  predict := [] deadlock;
ASSIGN init(h) := 0; -- The history variable
next(h) := case
  next(p1.loc) != p1.loc | next(p4.loc) != p4.loc:
    h = 0 & next(predict); 1;
  h = 1 & AS.mid1: 2;
  h = 2 & CS.p3.loc = 2 & CS.p4.loc = 2 &
    CS.mid1 & CS.mid2 & CS.mid3 & CS.right:
  h = 3 & CS.p3.loc = 2 & CS.p2.loc = 2 &
    CS.mid1 & CS.mid2 & CS.mid3 & CS.right:
next(h) := case
DEFINE
  AS : system Abstract;
  CS : system Concrete;
VAR h : boolean -- history variable
MODULE main
```

Advanced Topics in Reactive Verification, NYU, Spring 2004
MODULE Concrete
VAR left, right : boolean;
    mid1, mid2, mid3 : boolean
own;
    p1 : Phil(left, mid1);
p2 : Phil(mid1, mid2);
p3 : Phil(mid2, mid3);
p4 : Phil(mid3, right);

MODULE Abstract
VAR Left, Right : Boolean;
    mid1, mid2 : boolean
own;
p5 : process Phil(Left, mid1);
p6 : process Phil(mid1, mid2);
p7 : process Phil(mid2, Right);

...
Abstraction Mapping

\[ \text{Let corr} = \text{Left} = \text{left} \land \text{Right} = \text{right}; \]

\[ \text{-- correspondence between observables} \]

\[ \text{Let } \alpha = \text{corr} \land \text{loc} = \text{p1.loc} \land \text{loc} = \text{p6.loc} \land \text{loc} = \text{p7.loc} \land \text{loc} = \text{p5.loc}; \]

In the file `petsp3.pt`, we place:

**Abstract Mapping**