We will consider a general approach to abstraction of systems in order to simplify their verification.

Abstraction-Aided Verification
Often, both approaches to verification can be simplified by using abstraction.

To verify a reactive system $S$,

1. If it is finite state, model check it.
2. Otherwise, prove it by temporal deduction, using a temporal deductive system such as $\text{MP}$ or $\text{TLA}$, supported by theorem provers, such as $\text{STeP}$, $\text{TLP}$, or $\text{PVS}$.
An Obvious Idea:

\[ \text{AAV: Abstraction-Aided Verification} \]

Lecture 5: Abstraction-Aided Verification

Advanced Topics in Reactive Verification, NYU, Spring, 2004

The question considered here is whether we can find instantiations of this general methodology which are sound and (relatively) complete.

An Obvious idea:

Abstract system \( S \) into \( S_A \) — a simpler system, but admitting more behaviors.

Verify property for the abstracted system \( S_A \).

Conclude that property holds for the concrete system.

Model check \( \phi \models \exists \alpha \).

To prove (VFA) as follows:

Technically, define the methodology of Verification by Finitary Abstraction (VFA) as follows:

Propositional LTL formula \( \phi \models \exists \alpha \).

Abstract into a finite-state system \( D_A \) and the specification into a propositional LTL formula \( \phi \models D \).

Approach is particularly impressive when abstracting an infinite-state system into a finite-state one.
Finitary Abstraction

Based on the notion of abstract interpretation (CC77).

Finitary Abstraction

Let $\mathcal{A}$ be a mapping of concrete into abstract states. $\mathcal{A}$ is finitary if $\forall \mathcal{X} \subset \mathcal{A} \colon \mathcal{A}$ denotes the set of states of an FDS $D$ – the concrete states. Let $\mathcal{X}$ denote the set of states of an FDS $D$ – the concrete states.

We consider abstraction mappings which are presented by a set of equations:

\[ v_1 = E_1(v); \ldots; v_n = E_n(v) \]

(ormore compactly,

\[ \mathcal{V}^A = E(\mathcal{V}) \]

where $\mathcal{V}^A$ are the abstract state variables and $\mathcal{V}$ are the concrete variables.

Let $\mathcal{D}$ denote the set of states of an FDS $D$ – the concrete states.

Finitary Abstraction

A. Pnueli
Example: Program ANY-\text{Y}

Consider the program:

\[
0 = x, \quad x \text{ 's integer initially}
\]

Assume we wish to verify the property for system ANY-\text{Y}:

\[
\begin{align*}
[ & \begin{array}{l}
1 \equiv x\quad \text{initially} \\
0 = x
\end{array} & \text{while} ] \\
\begin{array}{l}
0 = x \quad \text{while}\ 0 = x
\end{array}
\end{align*}
\]

The abstraction mapping \( \alpha \) is specified by the following list of defining expressions:

\[
\{ \begin{array}{l}
X = (0 \neq x) = \bigwedge
Y = \text{sign}(y)
\end{array} \}
\]

where \( \text{sign}(y) \) is defined to be \(-1\), or \(1\), according to whether \( y \) is negative, zero, or positive, respectively.

\[
((\text{sign} = \lambda \ (0 \neq x) = X) : \alpha \\
\end{align*}
\]

Advanced Topics in Reactive Verification, NYU, Spring, 2004
The abstracted version with the mapping, we can obtain the abstract version of ANY-y, called ANY-Y.

\[
\begin{align*}
&\{\text{zero}\} \in X \\
&0 < f
\end{align*}
\]

The original invariance property \((y_0)\), is abstracted into:

\[
\begin{align*}
&\{\text{zero, pos}\} \in X \\
&0 = X
\end{align*}
\]

With the mapping \(\alpha\), we can obtain the abstract version of ANY-y, called ANY-Y.
When is such an Abstraction Sound?

Reconsider program \( \text{ANY} - Y \), but this time the property

\[ (0 \leq y \leq 10) \]
Lecture 5: Abstraction-Aided Verification

A. Pnueli

Lifting a State Abstraction to Assertions

For an abstraction mapping $V_A = E(V)$ and an assertion $p(V)$, there are two ways we can abstract $d$.

1. The expanding abstraction (over approximation) is given by

   \[ \|d\| \equiv \exists s \quad \exists (S')_{I-x} \subseteq S \quad \| (d)_{\overline{x}} \| \text{ to } S \]

   \begin{align*}
   \{ \|d\| \subseteq (S')_{I-x} \mid S \} = \| (d)_{\overline{x}} \|
   \end{align*}

   \[
   ((\Lambda)d \leftarrow (\Lambda)^{o}A = V:\Lambda) \Lambda A \quad \lor \quad ((\Lambda)^{o}A = V:\Lambda) \Lambda E \quad : (d)_{\overline{x}}
   \]

   Obviously, an abstract state $S$ belongs to $k(p)^k$ iff there exists some concrete state such that $(S')_{I-x} \subseteq S$.

2. The contracting abstraction (under approximation) is given by

   \[ \|d\| \equiv s \quad \exists (S')_{I-x} \subseteq S \quad \| (d)_{\overline{x}} \| \text{ to } S \]

   \begin{align*}
   \{ \|d\| \subseteq s \mid (s)_{o} \} = \| (d)_{\overline{x}} \| \\
   ((\Lambda)d \lor (\Lambda)^{o}A = V:\Lambda) \Lambda E \quad : (d)_{\overline{x}}
   \end{align*}

   For an abstraction mapping $V_A = E(V)$, there are two ways we can abstract $d$.
Claim 8. Let $L_1$ and $L_2$ be two sets of concrete states. The correct sound abstraction of set inclusions is:

\[(\overline{L_2}) \supseteq (\overline{L_1})\overline{v}\]
Visual Illustration of the Two Abstraction Transformers
The Existential (expanding) Abstraction
The Universal (contracting) Abstraction

Abstract state $S$ belongs to $(d)\overline{\varphi}$ if all concrete states $\varphi$-mapped into $S$ satisfy $(d)\varphi$. This is when $\varphi$ does not distinguish between two concrete states which are mapped by $d$ to the same abstract state. In such cases, the abstraction $\varphi$ is precise with respect to the assertion $(d)\overline{\varphi} = (d)\varphi$. 

\[(d)\overline{\varphi} = (d)\varphi\]
By transitivity of the diagram, abstract satisfaction implies concrete satisfaction.

\[ \| \phi \| \subseteq \| \sigma \| \]

Satisfaction, e.g., can be viewed as the inclusion

\[ \phi \models \sigma \]

\[ \forall \sigma \models \phi \]

The Diagram View
Recall the definition of fair discrete systems.

A fair discrete system (FDS) consists of:

\[ \mathcal{C}, \mathcal{L}^d, \Theta, \Lambda \]
A. Pnueli  

Lecture 5: Abstraction-Aided Verification

Soundness

If $\varphi$ is an abstraction mapping and $D$ and $\varphi(D)$ are abstracted according to the recipes presented above, then

$$\varphi \models \alpha \implies \varphi(D) \models \alpha$$

Sound Joint Abstraction

For an FDS $D = (V; J; C; \Lambda)$, where $\langle \alpha D, \alpha \varphi D, \alpha \theta \varphi D, \alpha \theta V \varphi \Lambda \rangle = \alpha D$

Every (maximal) state sub-formula $\varphi$ by $\varphi \in (d)$. For a temporal formula $\varphi$, we define the $\alpha$-abstracted version to be the formula obtained by replacing

$$(d)\varphi$$
$I = \forall \{I^-, 0\} \subseteq \mathcal{A} \land \{0, I^-\} \subseteq \mathcal{A} \land I^- = \mathcal{A} : (I + \bar{f} = \bar{f})\forall$

This enumeration yields the following abstraction:

\[
\begin{align*}
I + \bar{f} = \bar{f} \lor 0 & < \bar{f} \lor 0 > \bar{f} \iff I = (I+I^-)(d)\forall \\
I + \bar{f} = \bar{f} \lor 0 & = \bar{f} \lor 0 > \bar{f} \iff I = (0, I^-)(d)\forall \\
I + \bar{f} = \bar{f} \lor 0 & > \bar{f} \lor 0 > \bar{f} \iff I = (I-, I^-)(d)\forall 
\end{align*}
\]

In many cases, it is possible to break the computation of \((d)\forall\) into a set of decision

\[
(I + \bar{f} = \bar{f} \lor (\bar{f})\text{sign} = \mathcal{A} \lor (\bar{f})\text{sign} = \mathcal{A}) : \bar{f}, \bar{f}_E = (\mathcal{A}, \mathcal{A})(I + \bar{f} = \bar{f})\forall
\]

For example,

\[
(\mathcal{A}, \mathcal{A}d) \lor (\mathcal{A})^\vee 3 = \mathcal{V} \mathcal{A} \lor (\mathcal{A})^\vee 3 = \mathcal{V} \mathcal{A}) : \mathcal{A}, \mathcal{A}E \quad : (d)\forall
\]

Technically,

Computing the Abstract
Proving Soundness of the Method

Claim 9. If \( \varnothing \) is a computation of \( D \), then \( \varnothing \) is a computation of \( D \).

Claim 10. Let \( \alpha \) be a state sequence and \( \varnothing \) be a positive temporal formula. If

\[ \frac{\varnothing \models \varnothing}{\varnothing \models \varnothing} \]

and

\[ \frac{\varnothing \models \varnothing}{\varnothing \models \varnothing} \]
The mapping is called a predicate abstraction if it contains the boolean equation \( B \) for each atomic state formula. The abstraction mapping \( \alpha \) is defined by:

\[
\begin{align*}
\{ \forall d = \forall d \, B_1, \ldots, \forall d \, B_2, \forall d \, B_3 = d, \forall d \, B_4 = d \} & : \alpha \\
\end{align*}
\]

Following [BBM95] and [GS97], define abstract boolean variables \( B_1, B_2, \ldots, B_k \), one for each atomic state formula. Let \( \Phi \) be the set of all atomic state formulas referring to the data (non-control) variables appearing within conditions in the program \( P \) and within the temporal formula \( \phi \). Let \( d_1, d_2, \ldots, d_k \) be the set of all atomic state formulas referring to the data (non-control) variables appearing within conditions in the program \( P \) and within the temporal formula \( \phi \). Let \( \Theta \), \( \Omega \), \( \Phi \), \( L \), \( \Theta \), \( \Omega \), and \( \Phi \) for each atomic state formula \( d \) occurring in \( \phi \), \( \Theta \), and \( \Omega \).

The mapping \( \alpha \) is called a predicate abstraction if it contains the boolean equation. The Example of Predicate Abstraction.
The temporal properties for program BAKERY-2 are:

\[
\begin{align*}
&\text{Critical: } m_2 : \neg \phi_2 \\
&\text{await: } \psi_2 = \psi_1 \land \psi_1 > \psi_2 \\
&\text{Non-Critical: } m_2'(t) = m_2(t) + 1 \\
&\text{Loop forever do } m_2
\end{align*}
\]

\[
\begin{align*}
&\text{Critical: } m_1 : \neg \phi_1 \\
&\text{await: } \psi_1 = \psi_2 \land \psi_1 > \psi_2 \\
&\text{Non-Critical: } m_1'(t) = m_1(t) + 1 \\
&\text{Loop forever do } m_1
\end{align*}
\]

Example: Program BAKERY-2
### Abstracting Program BAkERY-2

**Define abstract variables**

\[
\begin{align*}
(0,0) & := \left( z_i > 1 \Leftrightarrow B_i = 0, z_i = 0 \right) & : \text{m} \\
(1,0) & := \left( z_i > 1 \Leftrightarrow B_i = 0, z_i = 0 \right) & : \text{m} \\
(0,1) & := \left( z_i > 1 \Leftrightarrow B_i = 0, z_i = 0 \right) & : \text{m} \\
(1,1) & := \left( z_i > 1 \Leftrightarrow B_i = 0, z_i = 0 \right) & : \text{m} \\
\end{align*}
\]

**local**

\[
\begin{align*}
B_i & = 0 \\
B_i & = 0 \\
B_i & = 0 \\
B_i & = 0 \\
\end{align*}
\]

**m0**: Loop forever do

\[
\begin{align*}
& 0 = z_i > 1 \Leftrightarrow B_i = 0, z_i = 0 \\
\text{where} \quad & \text{local} \\
& B_1 = 0, B_1 = 0, B_1 = 0, \text{ and } B_1 = 0, B_1 = 0 \\
\text{Defining abstract variables} & : \text{boolean}
\end{align*}
\]

The abstracted properties can now be model-checked.