Symbolic Model Checking

Definethe existential predecessor predicate transformer:

\[ [\exists \phi]d \sim d \lor \phi : \exists \phi \]

The expression \( \phi \) is obtained from \( \exists \phi \) by replacing every occurrence of variable \( \phi \) by \( d \).

Obviously \( s \iff \) some successor of \( s \) satisfies \( \phi \).

Characterizing all the states whose \( \exists \)-successor satisfies \( \phi \):

\[
\begin{align*}
\exists x = x & \sim \exists x = 1 + x \sim \\
\exists x = x \lor 1 + x = x : x = 1 + x = x & = (\exists x = x) \diamond (1 + x = x)
\end{align*}
\]

Predecessor computation yields:

For example, for a transition relation \( R \) and assertion \( x = x : R \) and assertion \( 1 + x = x : R \), the\( \exists x = x : \phi \) and assertion \( 1 + x = x : d \) implies \( \phi \diamond d \models s \).

 Obviously

\[
(\exists \Lambda) \phi \lor (\exists \Lambda \Lambda)^d : \exists \Lambda \phi \]

Define the existential predecessor predicate transformer:

Symbolic Model Checking
A Symptomatic Algorithm for Model Checking Invariance

A. Pnueli

Symbolic Model Checking

Algorithm: INV($D$, $p$):

1. $old := 0$
2. $new := p$
3. while ($new \neq old$) ^ ($new \neq D_{new}$) do
   a. $old := new$
   b. $new := new$
4. return $(old, new) ^ (D_{new})$

The algorithm returns an assertion characterizing all the initial states from which there exists a finite path leading to violation of $p$.

Symbolic operations:

Hardware Verification, NYU, Fall 2004
We iterate as follows:

\[ ((\mathcal{C} = \overline{\nu} \lor \mathcal{C} = \overline{1}) \land \neg (\nu)) \land \mathcal{I} = \overline{1} \lor \mathcal{N} = \overline{\nu} \lor \mathcal{N} = \overline{1} : \Theta \]

If we intersect \( \nu \) with the initial condition \( \Theta \), we obtain

\[ (b \lor d) \land d \]

The last equivalence is due to the general property

\[ \forall x \ (p \land \neg (p \lor q)) \equiv p \]

Hardware Verification, NYU, Fall, 2004
Symbolic Model Checking

Layers

Symbolic Exploration Progresses in Layers
Let $V$ be the state variables for the FDS $D$. Taking a vocabulary $U = V \cup V'$, we represent the state formulas $\Theta$, $J$ for each $J \in J$, $p$, $q$, for each $\langle p_i, q_i \rangle \in C$, and the INV symbolic working variables $\text{new}$ and $\text{old}$ as BDD's over $U$ which are independent of $V'$.

The transition relation $\rho$ is represented as a BDD over $U$ which may be fully dependent on both $V$ and $V'$. All the boolean operations used in the INV algorithm can be implemented by the Apply function. Negation can be computed by $t = t \oplus 1$, where $\oplus$ is sum modulo 2.

To check for equivalence such as $\text{old} = \text{new}$ we compute $t := (\text{old} \leftrightarrow \text{new})$ and then verify that the result is the singleton BDD 1.

The existential pre-condition transformer is computed by

$$A = A \lor (\forall V : \phi(V) \land \psi(V'))$$

Priming an assertion $\psi$ is performed by

$$(\forall V : \phi(V) \land \psi(V')) = \exists V : \phi(V) = \psi(V')$$
When we wish to progress beyond the simple property of invariance, it is helpful to introduce a special (temporal) logic for the specification of properties.

There are several versions of such logics. We will start with computational tree logic (CTL).

A Temporal Logic for the Specification of Properties
Assume an underlying (first-order) assertion language $\mathcal{L}$. The predicate at a location $\ell$ abbreviates the formula $\forall \ell \in \mathcal{L}$, where $\ell$ is a location within process $P$. A CTL operator has the form $\mathcal{L}^Q$, where the path quantifier $Q$ is one of $\forall, \exists$. The temporal operator $T$ is one of $\{\diamond, \Box\}$, where $\diamond$ is eventually, $\Box$ is always, $\forall$ is until, $\exists$ is D until, $\forall$ is G until, $\exists$ is H henceforth, $\forall$ is F eventually, $\exists$ is U until, $\forall$ is W waiting-for, unless.

Every assertion is a CTL formula.

If $p$ and $q$ are CTL formulas, then so are $\forall p$, $\exists p$, $\forall p$, $\exists p$, $\forall p$, $\exists p$, $\forall p$, and $\exists p$.

The predicate at $i$, abbreviates the formula $j = i$, where $i$ is a location within process $P$.

Hardware Verification, NYU, Fall, 2004

A CTL: Syntax
\(0 \geq \not\exists f\) for all positions \(\phi \models f\) 

For every \(\phi\) an originating run, \(\not\exists\) for all positions \(\phi \models f\) 

For some \(s = s_0, s_1, \ldots\) an originating run, \(\not\exists\) for all positions \(\phi \models f\)

\(\neg\) an originating run, \(\not\exists\) for all positions \(\phi \models f\)

\(\phi \models f\) with \(\not\exists\) and \(\forall\) successors of \(s\). 

\(\phi \models f\) with \(\not\exists\) and \(\forall\) successors of \(s\). 

\(\phi \models f\) with \(\forall\) successors of \(s\). 

\(\phi \models f\) with \(\forall\) successors of \(s\). 

\(\phi \models f\) with \(\forall\) successors of \(s\).

For CTL formulas and assertion ' \(b\) and \(d\) holds' \(\phi \models f\) 

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).

\(\phi \models f\) with \(\forall\) successors of \(s\).
A CTL formula which is satisfied by all FDS’s is called valid.

\[ \phi = \exists s \phi \]

If \[ s \models \phi \] for all initial states of \( D \), we say that \( \phi \) satisfies \( D \) and write \( s \models \phi \).

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \}

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \}

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]

\[ \forall j \geq 0 \text{ for all } d \models \forall s \text{ and } 0 \prec j \text{ for some position } a \Rightarrow b \models \forall s \text{ for all } d \models \forall s \]
Consider the following system $\mathcal{D}$:

- There exists a path along which all states satisfy $p$ but have a successor satisfying $\neg p$.
- There exists a reachable state, all of whose descendants satisfy $p$.
- It is not the case that every computation contains a state all of whose descendants satisfy $p$.

**Example**

A. Pnueli
Reading Exercises

Following are some CTL formulas and what do they say about the states:

- If holds at \( s \), then all runs originating at contain a state at.
- Every state-reachable from is followed by a.
- All runs departing from contain infinitely many states.
- Along every s-originating run, following every.
- Along every s-originating run, containing a state all of whose descendants satisfy.
- Every run departing from contains a state.
- Note that is not guaranteed, but it cannot happen.
- It means that property eventually stabilizes, but is not equivalent to it.
- The precedent property, it implies that property.
- Along every s-originating run, containing a state.
- Along every s-originating run, containing a state.
- Along every s-originating run, containing a state.
- Along every s-originating run, containing a state.

A. Pnueli

Lecture 5: Symbolic Model Checking
For a CTL formula $\varphi$, we denote by $\mathcal{D}$ the set of all $\varphi$-satisfying states. For the case that $\varphi$ is an assertion, $\varphi = \mathcal{D}$.

\[ \text{Claim 2. [Model Checking CTL Formulas]} \]

Thus, the essence of model checking is the computation of $\mathcal{D}$ for the various CTL formulas. We will provide a recipe for effective computation of the statification of all CTL formulas, starting with basic CTL formulas, i.e., formulas with a single CTL operator. For a CTL formula $\varphi$, we denote by $\mathcal{D}$ the set of all $\varphi$-satisfying states. For the case that $\varphi$ is an assertion, $\varphi = \mathcal{D}$.

Statification

A. Pnueli
We start by listing functions which compute a single
Model Checking CTL Formulas

A.Pnueli

Hardware Verification, NYU, Fall 2004

Lecture 5: Symbolic Model Checking

Function \texttt{CAX}(p)
\begin{align*}
\text{— Compute } \Delta_f ^{\text{AX}}p & \text{ for assertion } f \\
\text{Return } \text{CEX}(d) & \text{ (d)}
\end{align*}

The following function computes \( \Delta_f ^{\text{AX}}p \) for assertion \( f \):
The following function computes $kE$ for an assertion $p$. This is the set of all states which can reach a $p$-state by $0$ or more $p$-steps.

Function $CEF(p)$

\[
\begin{align*}
&\text{Function } CEF(p) \\
&\text{return } CEF(p) \\
&\text{while } \text{new} \neq \text{old} \\
&\text{old} = \text{old} \\
&\text{new} = \text{new} \\
&\text{return } \text{new} \\
&\text{end — CEF}
\end{align*}
\]

The following function computes $kA$ for an assertion $p$. This is the set of all states whose descendants satisfy $p$.

Function $CAG(p)$

\[
\begin{align*}
&\text{Function } CAG(p) \\
&\text{return } CAG(p) \\
&\text{end — CAG}
\end{align*}
\]
The following function computes $kE_p$ for an assertion $p$. This is the set of all states from which all runs must encounter a $p$-state. The function is defined as:

$$d \in \text{Compute}(CEG(d)) = (1;0)$$

while $new \neq old$ do

$$new = (0, 1)$$

return $new$

end

Function CAF

The following function computes $kA_p$ for an assertion $p$. This is the set of all states from which one can trace an infinite run all of whose states satisfy $p$. The function is defined as:

$$d \in \text{Compute}(CAF(d))$$

$\square d = \text{new}$

while $new \neq old$ do

$\text{new, old} = (0, 1)$

return $\text{new}$

end

Function CEG

A. Pnueli
Formulation as Fixed Points

The assertion expression \((\delta f)^*\) is called a monotonic if it satisfies the requirement

\[\|(\delta f)^*\| \supseteq \|(\delta f)^1\| \implies \|(\delta f)^2\| \supseteq \|(\delta f)^1\|\]

that \(f\) has no solution.

Not every fixpoint equation has a solution. For example, the equation \(f\) has no solution. Such a recursive equation of the general form \(\delta y = f(y)\) represents a set of states, is called a fixpoint equation.

\[
\delta y = f(y)
\]

By successive iterations, the only difference between the two cases is that the

\[
\begin{align*}
\text{CEX(new)} & \lor d = \text{new} \\
\text{CEX(new)} & \land d = \text{new}
\end{align*}
\]

both attempt to solve recursive equations of the form

Examining the algorithms for computing

Formulation as Fixed Points
Consider a fixpoint equation

\[ f(y) = y \]

Every assertion expression \((\varphi)\) which is constructed out of the assertion variable and arbitrary constant assertions, to which we apply the boolean operators \(\land\) and the predecessor operators \(\mathsf{EX}\) and \(\mathsf{AX}\) is monotonic.

Solution to Fixpoint Equations

A.Pnueli

SolutionstoFixpointEquations

Claim 3. If \((\varphi)\) is a monotonic expression, then the fixpoint equation \(y = f(y)\) has both a minimal and a maximal solution which can be obtained by the iteration sequence

\[ y_1 = f(y_0); y_2 = f(y_1); y_3 = f(y_2); \ldots \]

where \(y_0 = 0\) for the minimal solution, and \(y_0 = 1\) for the maximal solution.

\[ \cdots \]

\[ (\varphi_1) = f(\varphi_0), (\varphi_2) = f(\varphi_1), (\varphi_0) = f(\varphi_1) = y \]
Expressing the Higher Modalities as Fixpoints

All the CTL operators, excluding the basic $\mathsf{EX}$ and $\mathsf{AX}$ can be expressed by fixpoint expressions.

$$
\begin{align*}
(\neg \mathsf{AX} \mathsf{A} \lor \lVert d \rVert) \land \lVert b \rVert \cdot \mathsf{\neg A} \mathsf{A} &= \lVert b \mathsf{A} \mathsf{A} d \rVert \\
(\neg \mathsf{AX} \mathsf{A} \lor \lVert d \rVert) \land \lVert b \rVert \cdot \mathsf{\neg A} \mathsf{X} &= \lVert b \mathsf{X} \mathsf{A} d \rVert \\
(\neg \mathsf{EX} \mathsf{A} \lor \lVert d \rVert) \land \lVert b \rVert \cdot \mathsf{\neg E} \mathsf{A} &= \lVert b \mathsf{A} \mathsf{E} d \rVert \\
(\neg \mathsf{EX} \mathsf{A} \lor \lVert d \rVert) \land \lVert b \rVert \cdot \mathsf{\neg E} \mathsf{X} &= \lVert b \mathsf{X} \mathsf{E} d \rVert \\
\mathsf{AX} \lor \lVert d \rVert \cdot \mathsf{A} &= \lVert d \mathsf{A} \rVert \\
\mathsf{AX} \land \lVert d \rVert \cdot \mathsf{A} &= \lVert d \mathsf{X} \rVert \\
\mathsf{EX} \lor \lVert d \rVert \cdot \mathsf{E} &= \lVert d \mathsf{E} \rVert \\
\mathsf{EX} \land \lVert d \rVert \cdot \mathsf{E} &= \lVert d \mathsf{X} \rVert \\
(\lVert d \rVert) \mathsf{AX} &= \lVert d \mathsf{A} \rVert \\
(\lVert d \rVert) \mathsf{EX} &= \lVert d \mathsf{X} \rVert
\end{align*}
$$

Note the abuse of notation by which we write $\mathsf{EX}$ and $\mathsf{AX}$ instead of $\mathsf{CEX}$ and $\mathsf{CA}$. These expressions can be used for computing the stuttering-invariant of the $\mathsf{CTL}$ formulas appearing on the left-hand side.
Consider a general CTL formula. Claim 4 enables us to eliminate all nested CTL formulas, starting with the innermost ones, until we end up with a basic CTL formula, and then apply Claim 2.

Claim 4, [Elimination of a nested CTL formula]

\[ \| (\exists \psi) \phi \| = \| (\exists \psi) \phi \| \]

The statement of Claim 4 can also be phrased as:

\[ \| (\exists \psi) \phi \| = \| (\exists \psi) \phi \| \]

Let us now, we have considered basic CTL formulas, which are CTL formulas with a single CTL operator. Let \( (\phi) f \) be a CTL formula containing one or more occurrences of the nested CTL formula, and let \( \| \phi \| = b \), where \( b \) is obtained by substituting all occurrences of \( \phi \) in \( f \).
Try to model check the formula $\varphi$. Following Claim 4, we compute $d \square A \Diamond A \varphi = f$. Now it remains to check $\neg k f (k' k) = k A (k = 2)$.

First, we compute $(z = \nu) = \|d \square A\| = \|\varphi\|$

Next, we compute $(0 < \nu) = \| (z = \nu) \Diamond A \| = \| (\| \varphi \| f) \|$

Finally, we compute $0 = (0 < \nu \leftarrow d \vee 0 = \nu) = \| f \| \leftarrow \Theta$

which shows that $A \Diamond A \varphi$ does not hold on $D$. 

Example