Bounded Model Checking
Motivation

Abstraction works using overapproximation

The technique considers all real executions plus additional, “spurious” ones

Question: Is a (LTL) property satisfied?

Answer: Yes (if actual system satisfies it), no (if possibly violated)

→ answer is precise on the “safe” side

Goal of overapproximation: verification (= proof of correctness)
Complementary approach: underapproximation

Considers a subset of the real executions

Answer: no (if error found), “don’t know” (if no error found)

Goal: falsification (finding errors, gaining confidence)

akin to testing methods, but more systematic
Example of an underapproximation technique: Bounded Model Checking (BMC)

Consider only runs of length $k$, for some fixed value of $k$

Introduced by Biere et al 1999

Overview

Given: (compact description of) Kripke structure $\mathcal{K}$, LTL formula $\phi$, bound $k \geq 0$

Translation of $\neg \phi$ and all paths of length $k$ of $\mathcal{K}$ into a propositional-logic formula $F$

If $F$ satisfiable, then $\mathcal{K}$ does not satisfy $\phi$.
(The reverse implication does not hold!)

Use a SAT checker for the latter.
Paths as PL formulae

In the following, let $\mathcal{K} = (S, \rightarrow, r, AP, \nu)$ be a Kripke structure, where $S$ is finite.
Assumption: $S = \{0, 1\}^m$ for some $m$ (wlog.)
Another assumption: $\mathcal{K}$ does not have deadlocks.

Any state from $S$ can be described as a vector $\vec{s}$ of $m$ atomic propositions.

Use vectors $\vec{s}_0, \ldots, \vec{s}_k$ to describe paths of length $k$
$((k + 1) \cdot m$ atomic propositions).

In the following we identify $\vec{s}_i$ with the state $s_i$ it represents.
Let $I(s_0)$ be a formula s.t. $I(s_0)$ is true iff $s_0 = r$.

Let $T(s_i, s_j)$ be true iff $s_i \rightarrow s_j$.

Then $F_P := I(s_0) \land \bigwedge_{i=1}^{k} T(s_{i-1}, s_i)$ is true iff the corresponding states form a path of length $k$. 
**k-loops**

Let $p = s_0, \ldots, s_k$ be a path of length $k$.

Let $0 \leq \ell \leq k$. We call $p$ a $(k, \ell)$-loop if $s_k \rightarrow s_\ell$.

We call $p$ a **loop** if $p$ is a $(k, \ell)$-loop for some value of $\ell$.

$p$ is **non-looping** if $p$ is not a loop.
LTL semantics for loops

Let $p$ be a $(k, \ell)$-loop, and let $\phi$ be an LTL formula in negative normal form (all negations pushed inside).

$p$ represents an infinite path, by repeating the loop over and over:
$$\pi = s_0 \ldots s_{\ell-1}(s_\ell \ldots s_k)^\omega.$$  

We shall write: $p \models^k \phi$ iff $\pi \models \phi$.

Let $\sigma(i) := i + 1$ for $i < k$ and $\sigma(k) := \ell$ (successor position).

Let $0 \leq i, j \leq k$. We define $i \preceq_\ell j$ iff $i \leq j$ or $i, j \geq \ell$.
(Intuition: $s_j$ appears after $s_j$ in $\pi$.)
Let $i \leq k$. Observation:

\[
\begin{align*}
\pi^i &\models \phi \iff \pi^{i+j \cdot (k-\ell+1)} \models \phi \text{ for any } j \geq 0 \text{ and } i \geq \ell, \\
\pi^i &\models a \iff a \in \nu(s_j) \text{ for } a \in \text{AP} \\
\pi^i &\models \neg a \iff a \notin \nu(s_j) \text{ for } a \in \text{AP} \\
\pi^i &\models \phi \lor \psi \iff \pi^i \models \phi \lor \pi^i \models \psi \\
\pi^i &\models \phi \land \psi \iff \pi^i \models \phi \land \pi^i \models \psi \\
\pi^i &\models X \phi \iff \pi^{\sigma(i)} \models \phi \\
\pi^i &\models \phi \mathbf{U} \psi \iff \pi^i \models \psi \lor (\pi^i \models \phi \land \pi^{\sigma(i)} \models \phi \mathbf{U} \psi) \\
\pi^i &\models \phi \mathbf{R} \psi \iff \pi^i \models \phi \land \psi \lor (\pi^i \models \psi \land \pi^{\sigma(i)} \models \phi \mathbf{R} \psi)
\end{align*}
\]

Conclusion: $\pi \models \phi$ (and therefore $p \models^k \phi$) depends only on $p$. 

10
LTL semantics for finite paths

Let $p$ be a path (possibly non-looping) of length $k$ and $\phi$ an LTL formula in negative normal form.

Since $p$ does not have any deadlocks (by assumption), $p$ can be extended to at least one infinite path $\pi$.

Let us define $p \models^k \phi$ in such a manner that $p \models^k \phi$ implies: $\pi \models \phi$ for any infinite extension of $p$. 
The following definition attains this goal (where $i \leq k$):

\[
\begin{align*}
    p^{k+1} \not\models^k \phi & \quad \text{for any formula } \phi \\
    p^i \models^k a & \iff a \in \nu(s_i) \quad \text{for } a \in AP \\
    p^i \models^k \neg a & \iff a \notin \nu(s_i) \quad \text{for } a \in AP \\
    p^i \models^k \phi \lor \psi & \iff p^i \models^k \phi \lor p^i \models^k \psi \\
    p^i \models^k \phi \land \psi & \iff p^i \models^k \phi \land p^i \models^k \psi \\
    p^i \models^k X \phi & \iff p^{i+1} \models^k \phi \\
    p^i \models^k \phi \mathbf{U} \psi & \iff p^i \models^k \psi \lor (p^i \models^k \phi \land p^{i+1} \models^k \phi \mathbf{U} \psi) \\
    p^i \models^k \phi \mathbf{R} \psi & \iff p^i \models^k \phi \land \psi \lor (p^i \models^k \psi \land p^{i+1} \models^k \phi \mathbf{R} \psi)
\end{align*}
\]

In the above, $p^i$ denotes the sequence $s_i \ldots s_k$. 
Properties of BMC

Let $p$ be a path of length $k$ and $\phi$ an LTL formula.

We have: If $p \models^k \neg \phi$, then $\mathcal{K} \not\models \phi$.

Moreover: If $\mathcal{K} \not\models \phi$, then there are $k \geq 0$ and $p$ s.t. $p \models^k \neg \phi$. 
Translation into propositional logic

We generate a formula $F$ with the following properties:

$F$ is satisfiable iff there exists $p$ with $p \models^k \neg \phi$.

Observation: Fix a path $p$. To check whether, $ob\ p \models^k \neg \phi$, we need to check the subformulae of $\phi$ only in finitely many places.
Translating a loop

Let $\ell \leq k$ and $p$ a $(k, \ell)$-loop, and let $\pi$ be the corresponding infinite path. Let $\psi$ be a subformula of (the NNF of) $\neg \phi$ and $i \leq k$.

We introduce atomic propositions of the form $\ell[\psi]_k^i$.

$\ell[\psi]_k^i$ should be true iff $\pi^i \models \psi$.

Generate subformulae of $F$ by exploiting the previous observations, e.g.

\[
\ell[\psi_1 \lor \psi_2]_k^i \iff \ell[\psi_1]_k^i \lor \ell[\psi_2]_k^i
\]

\[
\ell[\mathbf{X} \psi]_k^i \iff \ell[\psi]_k^{\sigma(i)}
\]

Finally, conjoin all those subformulae.
Attention: For $\mathbf{U}$, something special must be done:

$$
el[\psi_1 U \psi_2]^i_k \leftrightarrow (el[\psi_2]^i_k \lor (el[\psi_1]^i_k \land el[\psi_1 U \psi_2]^{\sigma(i)}_k))$$

$$\land \lor \sum_{i \geq j} el[\psi_2]^j_k$$

Without the last clause it may happen that the right-hand side of a $\mathbf{U}$-formula is never satisfied.

By analogy, we introduce atomic propositions $[\psi]^i_k$ for arbitrary paths $p$. 
Overview of the translation

\( F \) is defined as follows:

\[
F := l(s_0) \land \bigwedge_{i=1}^{k} T(s_{i-1}, s_i) \land \left( \left[ \neg \phi \right]_k^0 \lor \bigvee_{\ell=0}^{k} \left( T(s_k, s_\ell) \land \ell \left[ \neg \phi \right]_k^0 \right) \right)
\]

If \( F \) is satisfiable, then \( \mathcal{K} \not\models \phi \).

**Remark:** size of \( F \) is polynomial in \(|\mathcal{K}|, |\phi|, \) and \( k \).
Satisfiability checking for PL formulae

First option: generate a BDD for $F$, check whether that BDD contains only the 0-node.

Second option: use a dedicated SAT checker.
SAT checkers

Find *some* satisfying valuation, often very quickly.
(BDDs would construct *all* satisfying valuations.)

Enormous progress in this area since about 2000:

- Hundreds of thousands of variables and clauses manageable

  - Examples: zChaff, MiniSAT, . . .

Advantages: memory efficiency!
**SAT and BMC**

Advantages for software model checking: all language features can (in principle) be handled.

Usual approach: user fixed a bound for the number of loop unrollings. BMC checker starts with small value of $k$ and tries successively higher values as long as feasible.

In practice still efficient for values up to 80.

Well-known implementation: CBMC (for C programs)

http://www.cprover.org/cbmc/