**RECALL: BOUNDED MODEL CHECKING**

System model:
- open
- start empty
- close empty
- close
- start close
- start heat
- heat

Property specification:
- AG (start → AF heat)

Model Checker:
- Property satisfied
- Property violated
RECALL: BOUNDED MODEL CHECKING

System model

AG (start $\rightarrow$ AF heat)

Model Checker

Property specification

Property satisfied

Property violated
Let $M = \{S, I, R, L\}$ be Kripke structure

Define predicate $\text{Reach}(s, s') \equiv R(s, s')$

$[M]^k = \bigwedge_{i=0}^{k-1} \text{Reach}(s_i, s_{i+1})$

$[M]^k$ contains states reachable in exactly $k$ steps

Then search for counterexamples formed by $k$ states
RECALL: BOUNDED MODEL CHECKING – PROCEDURE

Input: $M, \neg \varphi$

1. $k = 0$
2. Is $\neg \varphi$ satisfiable in $[M]^k$?
   - YES: $M \models \neg \varphi$, terminate
3. Is $k < \text{threshold}$?
   - NO: $M \not\models_k \neg \varphi$, terminate
4. Increment $k$
5. Go to 2.
Unwind loops and transform each line of code into (CNF) formula.

1: int i = 4;
2: int s = 0;
3: s += i;
4: if (i > 0)
  5:   i --;
6: assert (s < 10);
7: assert (s < 10);
8: s += i;
9: if (i > 0)
10:  i --;
11: assert (s < 10);

\[ f_1 : (pc_1 = 1) \land (i_2 = 4) \land (pc_2 = 2) \]
\[ f_2 : (pc_2 = 2) \land (i_3 = i_2) \land (s_3 = 0) \land (pc_3 = 3) \]
\[ f_3 : (pc_3 = 3) \land (i_4 = i_3) \land (s_4 = s_3) \land (pc_4 = 4) \]
\[ f_4 : (pc_4 = 4) \land (i_5 = i_4) \land (s_5 = s_4 + i_4) \land (pc_5 = 5) \]
\[ f_5 : (pc_5 = 5) \land (i_6 = i_5) \land (s_6 = s_5) \land (pc_6 = 6) \]
\[ f_6 : (pc_6 = 6) \land (((i_6 > 0) \land (i_7 = i_6 - 1)) \lor \\
    ((i_6 \leq 0) \land (i_7 = i_6))) \land (s_7 = s_6) \land (pc_7 = 7) \]
\[ f_7 : (pc_7 = 7) \land (s_7 \geq 10) \land (pc_8 = 8) \]
\[ f_8 : (pc_8 = 8) \land (i_9 = i_8) \land (s_9 = s_8 + i_8) \land (pc_9 = 9) \]
\[ f_9 : (pc_9 = 9) \land (i_{10} = i_9) \land (s_{10} = s_9) \land (pc_{10} = 10) \]
\[ f_{10} : (pc_{10} = 10) \land (((i_{10} > 0) \land (i_{11} = i_{10} - 1)) \lor \\
   ((i_{10} \leq 0) \land (i_{11} = i_{10}))) \land (s_{11} = s_{10}) \land (pc_{11} = 11) \]
\[ f_{11} : (pc_{11} = 11) \land (s_{11} \geq 10) \land (pc_{12} = 12) \]
Unwind loops and transform each line of code into (CNF) formula.

\begin{align*}
1: & \textbf{int } i = 4; \\
2: & \textbf{int } s = 0; \\
3: & \\
4: & s += i; \\
5: & \textbf{if } (i > 0) \\
6: & i --; \\
7: & \textbf{assert } (s < 10); \\
8: & s += i; \\
9: & \textbf{if } (i > 0) \\
10: & i --; \\
11: & \textbf{assert } (s < 10);
\end{align*}

\begin{align*}
\text{f}_1 : (pc_1 = 1) & \land (i_2 = 4) & \land (pc_2 = 2) \\
\text{f}_2 : (pc_2 = 2) & \land (i_3 = i_2) & \land (s_3 = 0) & \land (pc_3 = 3) \\
\text{f}_3 : (pc_3 = 3) & \land (i_4 = i_3) & \land (s_4 = s_3) & \land (pc_4 = 4) \\
\text{f}_4 : (pc_4 = 4) & \land (i_5 = i_4) & \land (s_5 = s_4 + i_4) & \land (pc_5 = 5) \\
\text{f}_5 : (pc_5 = 5) & \land (i_6 = i_5) & \land (s_6 = s_5) & \land (pc_6 = 6) \\
\text{f}_6 : (pc_6 = 6) & \land (((i_6 > 0) & (i_7 = i_6 - 1))) & \lor & (i_6 \leq 0) & \land (i_7 = i_6)) & \land (s_7 = s_6) & \land (pc_7 = 7) \\
\text{f}_7 : (pc_7 = 7) & \land (s_7 \geq 10) & \land (pc_8 = 8) \\
\text{f}_8 : (pc_8 = 8) & \land (i_9 = i_8) & \land (s_9 = s_8 + i_8) & \land (pc_9 = 9) \\
\text{f}_9 : (pc_9 = 9) & \land (i_{10} = i_9) & \land (s_{10} = s_9) & \land (pc_{10} = 10) \\
\text{f}_{10} : (pc_{10} = 10) & \land (((i_{10} > 0) & (i_{11} = i_{10} - 1))) & \lor & ((i_{10} \leq 0) & (i_{11} = i_{10})) & \land (s_{11} = s_{10}) & \land (pc_{11} = 11) \\
\text{f}_{11} : (pc_{11} = 11) & \land (s_{11} \geq 10) & \land (pc_{12} = 12)
\end{align*}
Let system under verification be represented as transition system $M = (I, T)$ and set of error states $E$ over variables in $V$:

- $I(V)$ – set of initial states
- $T(V, V')$ – transition relation
- $E(V)$ – set of error states (we assume safety properties only!)

All sets are represented as logical formulae – SAT/SMT solver is used to decide upon satisfiability by model checking algorithm.

$M$ does not contain error trace of length (exactly) $k$ if the formula is unsatisfiable:

$$I(V_0) \land \left[ \bigwedge_{0 \leq i < k} T(V_i, V_{i+1}) \right] \land E(V_k)$$
UNBOUNDED MODEL CHECKING

- BMC is **limited** to error traces **up to given length** $k$ – this might be quite limiting for program verification due to huge number of iterations computing sets of reachable states
- Unbounded Model Checking attempts to overcome this by computing sequences of sets **over-approximating** reachable sets of states after $i$ steps
  - e.g., by means of **Craig’s interpolation**
- The problem is still undecidable, however, for many practical cases, this approach converges
**CRAIG’S INTERPOLANTS**

**Definition** (Craig’s interpolant): Let $A, B$ be formulae such that $A \land B \rightarrow \bot$. Formula $I$ is an interpolant for $(A, B)$ iff

1. $A \rightarrow I$,
2. $I \land B \rightarrow \bot$, and
3. $\text{Var}(I) \subseteq \text{Var}(A) \cap \text{Var}(B)$

**Theorem** (Craig, 1957):
For each pair of propositional formulae $A, B$ such that $A \land B \rightarrow \bot$, there exists an interpolant $I$ for $(A, B)$.

---

Craig’s Interpolants

For given formulae \((A, B)\), interpolant is not unique

- Variability
  - complexity
    - number of connectives
    - number of unique variables
  - logical strength

Example:

\[
A = \{a_1\bar{a}_2, \bar{a}_1\bar{a}_3, a_2\}, \quad B = \{\bar{a}_2a_3, a_2a_4, \bar{a}_4\}
\]

\[
I_1 = \bar{a}_3 \land a_2
\]

\[
I_2 = \bar{a}_3
\]

\[
I_3 = \bar{a}_3 \lor \bar{a}_2
\]
GRAPHICAL VIEW OF INTERPOLANT

A

B
GRAPHICAL VIEW OF INTERPOLANT

\[ I_1 \]

A

B
GRAPHICAL VIEW OF INTERPOLANT

\[ I_1 \subseteq I_2 \subseteq A \]

\[ B \]
GRAPHICAL VIEW OF INTERPOLANT

\[ I_1 \]

\[ I_3 \]

\[ A \]

\[ B \]
Interpolants can serve as over-approximation of sets of (reachable) states.

Why not using the exact representation of states?
- interpolant is usually simpler than precise representation
- model checking algorithm can converge faster – in less iterations (later)

Interpolant can be computed from resolution refutation proof of unsatisfiability
- in linear time wrt. proof size
- various interpolation systems exist
The idea is to use approach of bounded model checking while attempting to find **fixpoint** of over-approximation of reachable states w.r.t. transition relation.

Since state space is finite (BMC), algorithm always finishes:
- however, state space size can be huge, practically equal to unbounded
- it can take very long
- therefore, we need smart way to simplify, i.e., over-approximate sets of states

Error state can be reachable (from over-approximation) due to too coarse over-approximation
⇒ **refine** over-approximation
Safe over-approximation of set of states represented by $A$ w.r.t. $E$ is formula $O$ such that:

- $A \Rightarrow O$, and
- $O \cap E = \emptyset$

We want to find either safe over-approximation of all reachable states or real error
VERIFICATION EXAMPLE
VERIFICATION EXAMPLE

I

O

≡

Jan Kofroň: Behaviour Models and Verification
Verification Example
VERIFICATION EXAMPLE

Jan Kofroň: Behaviour Models and Verification
VERIFICATION EXAMPLE

Jan Kofroň: Behaviour Models and Verification
VERIFICATION EXAMPLE

\[ O^3_3 \equiv O^2_3 \]

\[ I \quad S^1_3 \]

\[ E \]
**APPROACH FORMALLY**

Initial checks:

- $I(V_0) \land E(V_0)$ is SAT $\implies$ error
- $I(V_0) \land T(V_0, V_1) \land E(V_1)$ is SAT $\implies$ error

General case (start with $k = 1$):

- $\psi_k(S_i) \equiv S_i(V_0) \land \left[ \bigwedge_{0 \leq i < k} T(V_i, V_{i+1}) \right] \land E(V_k)$

Splitting $\psi_k$ for interpolation:

- $A \equiv S_i(V_0) \land T(V_0, V_1)$
- $B \equiv \left[ \bigwedge_{1 \leq i < k} T(V_i, V_{i+1}) \right] \land E(V_k)$
- $S_{i+1} = \text{interpolant for } (A, B)$
- $S_i$ is $i$-th over-approximation of $I$
**Verification Algorithm**

```latex
\textbf{function} \textsc{Verify}(M, E) \\
\quad \textbf{if} I \cap E \neq \emptyset \ \textbf{then return} \ \text{Error} \\
\quad \textbf{end if} \\
\quad k := 1 \\
\quad \textbf{while} \text{true} \ \textbf{do} \\
\quad \quad \text{result} := \text{FindFP}(M, E, k) \\
\quad \quad \textbf{if} \ \text{result is unknown} \ \textbf{then} \\
\quad \quad \quad k := k + 1 \\
\quad \quad \textbf{else return result} \\
\quad \textbf{end if} \\
\quad \textbf{end while} \\
\textbf{end function}
```

```latex
\textbf{function} \textsc{FindFP}(M, E, k) \\
\quad S_0 := I \\
\quad i := 0 \\
\quad \textbf{while} \psi^M_k(S_i) \ \text{is UNSAT} \ \textbf{do} \\
\quad \quad (A, B) := \text{Split}(\psi^M_k(S_i)) \\
\quad \quad T_i := \text{Itp}(A, B)[V := \check{V}'] \\
\quad \quad \textbf{if} T_i \wedge \neg S_i \ \text{is UNSAT} \ \textbf{then return} \ \text{safe} \\
\quad \quad \textbf{end if} \\
\quad \quad S_{i+1} := S_i \lor T_i \\
\quad \quad i := i + 1 \\
\quad \textbf{end while} \\
\quad \textbf{if} S_i = I \ \textbf{then return} \ \text{unsafe} \\
\quad \textbf{else return} \ \text{unknown} \\
\textbf{end if} \\
\textbf{end function}
```
IMPLEMENTATION

- Front end transforms input program (e.g., in C) into formula representation
- Model checking algorithm implemented in model checker
- SAT checking and interpolant computation provided by (interpolating) SMT solver
  - OPENSMIT\(^2\), SMTINTERPOL\(^3\)

- Tools employing interpolation: BLAST\(^4\), CPACHECKER\(^5\), SEAHORN (SPACER)\(^6\)

\(^2\)https://github.com/usi-verification-and-security/opensmt/
\(^3\)https://github.com/ultimate-pa/smtinterpol/
\(^4\)http://mtc.epfl.ch/software-tools/blast/index-epfl.php
\(^5\)https://cpachecker.sosy-lab.org/
\(^6\)https://seahorn.github.io/