RECALL: BOUNDED MODEL CHECKING

System model

<table>
<thead>
<tr>
<th>open</th>
<th>start empty</th>
<th>close empty</th>
<th>close</th>
<th>heat</th>
</tr>
</thead>
</table>

System model:

AG (start → AF heat)

Property specification

Model Checker

Property satisfied

Property violated

Jan Kofroň: Behaviour Models and Verification
RECALL: BOUNDED MODEL CHECKING

System model

\[ \text{AG (start } \rightarrow \text{ AF heat)} \]

Property specification

Model Checker

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
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.
Recall: BMC For Programs

Unwind loops and transform each line of code into (CNF) formula.

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

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

\begin{align*}
1: f_1 &= (pc_1 = 1) \land (i_2 = 4) \land (pc_2 = 2) \\
2: f_2 &= (pc_2 = 2) \land (i_3 = i_2) \land (s_3 = 0) \land (pc_3 = 3) \\
3: f_3 &= (pc_3 = 3) \land (i_4 = i_3) \land (s_4 = s_3) \land (pc_4 = 4) \\
4: f_4 &= (pc_4 = 4) \land (i_5 = i_4) \land (s_5 = s_4 + i_4) \land (pc_5 = 5) \\
5: f_5 &= (pc_5 = 5) \land (i_6 = i_5) \land (s_6 = s_5) \land (pc_6 = 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) \\
7: f_7 &= (pc_7 = 7) \land (s_7 \geq 10) \land (pc_8 = 8) \\
8: f_8 &= (pc_8 = 8) \land (i_9 = i_8) \land (s_9 = s_8 + i_8) \land (pc_9 = 9) \\
9: f_9 &= (pc_9 = 9) \land (i_{10} = i_9) \land (s_{10} = s_9) \land (pc_{10} = 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) \\
11: 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

- $A \rightarrow I$,
- $I \land B \rightarrow \bot$, and
- $\text{Var}(I) \subseteq \text{Var}(A) \cap \text{Var}(B)$

Theorem (Craig, 1957):¹
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:

\[ A = \{a_1\bar{a}_2, \bar{a}_1a_3, a_2\}, B = \{\bar{a}_2a_3, a_2a_4, \bar{a}_4\} \]

- \(I_1 = \bar{a}_3 \land a_2\)
- \(I_2 = \bar{a}_3\)
- \(I_3 = \bar{a}_3 \lor \bar{a}_2\)
GRAPHICAL VIEW OF INTERPOLANT

$I_1$ $I_2$  

A  

B
GRAPHICAL VIEW OF INTERPOLANT

$A_1$, $A$, $A_3$, $B$
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:

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

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

![Diagram of verification example](image-url)
VERIFICATION EXAMPLE

\[ I \quad O_3^1 \quad O_3^2 \]

\[ E \]

Jan Kofroň: Behaviour Models and Verification 24
VERIFICATION EXAMPLE

\[ O_3^3 \equiv O_3^2 \]

\[ O_3^1 \]

Jan Kofroň: Behaviour Models and Verification
**APPROACH FORMALLY**

Initial checks:

\[ I(V_0) \land E(V_0) \text{ is SAT} \implies \text{error} \]
\[ I(V_0) \land T(V_0, V_1) \land E(V_1) \text{ is SAT} \implies \text{error} \]

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:

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

```plaintext
function VERIFY(M, E)
    if I ∩ E ≠ ∅ then return Error
end if
    k := 1
    while true do
        result := FindFP(M, E, k)
        if result is unknown then
            k := k + 1
        else return result
        end if
    end while
end function

function FINDFP(M, E, k)
    S₀ := I
    i := 0
    while ψ^M_k(S_i) is UNSAT do
        (A, B) := Split(ψ^M_k(S_i))
        T_i := Itp(A, B)[V := V']
        if T_i ∧ ¬S_i is UNSAT then return safe
        end if
        S_{i+1} := S_i ∨ T_i
        i := i + 1
    end while
    if S_i = I then return unsafe
else return unknown
end if
end function
```

Jan Kofroň: Behaviour Models and Verification
**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
  - OPENSMT\(^2\), 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/