NSWI101: System Behaviour Models and Verification

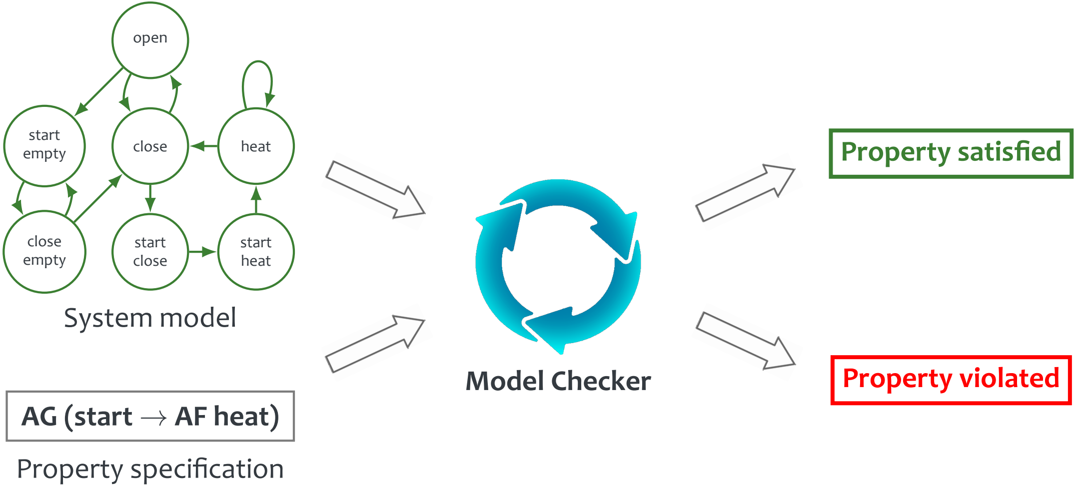
11. Unbounded Software Model Checking

Jan Kofroň

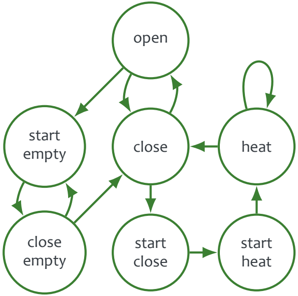
FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Department of Distributed and Dependable Systems

Recall: Bounded Model Checking



Recall: Bounded Model Checking



System model

AG (start → AF heat)

Property specification

Model Checker

Property satisfied

Property violated

Recall: Bounded Model Checking

- Let $M = \{S, I, R, L\}$ be Kripke structure
- Define predicate $Reach(s, s') \equiv R(s, s')$
- $[\![M]\!]^k = \bigwedge_{i=0}^{k-1} Reach(s_i, s_{i+1})$
- $[\![M]\!]^k$ 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 $k <$ 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;
 2: int s=0;
 3:
 4: s+=i;
 5: if (i>0)
 6:     i--;

 7: assert(s<10);
 8: s+=i;
 9: if (i>0)
10:     i--;

11: assert(s<10);
```

$$f_1 : (pc_1 = 1) \wedge (i_2 = 4) \wedge (pc_2 = 2)$$

$$f_2 : (pc_2 = 2) \wedge (i_3 = i_2) \wedge (s_3 = 0) \wedge (pc_3 = 3)$$

$$f_3 : (pc_3 = 3) \wedge (i_4 = i_3) \wedge (s_4 = s_3) \wedge (pc_4 = 4)$$

$$f_4 : (pc_4 = 4) \wedge (i_5 = i_4) \wedge (s_5 = s_4 + i_4) \wedge (pc_5 = 5)$$

$$f_5 : (pc_5 = 5) \wedge (i_6 = i_5) \wedge (s_6 = s_5) \wedge (pc_6 = 6)$$

$$f_6 : (pc_6 = 6) \wedge (((i_6 > 0) \wedge (i_7 = i_6 - 1)) \vee ((i_6 \leq 0) \wedge (i_7 = i_6))) \wedge (s_7 = s_6) \wedge (pc_7 = 7)$$

$$f_7 : (pc_7 = 7) \wedge (s_7 \geq 10) \wedge (pc_8 = 8)$$

$$f_8 : (pc_8 = 8) \wedge (i_9 = i_8) \wedge (s_9 = s_8 + i_8) \wedge (pc_9 = 9)$$

$$f_9 : (pc_9 = 9) \wedge (i_{10} = i_9) \wedge (s_{10} = s_9) \wedge (pc_{10} = 10)$$

$$f_{10} : (pc_{10} = 10) \wedge (((i_{10} > 0) \wedge (i_{11} = i_{10} - 1)) \vee ((i_{10} \leq 0) \wedge (i_{11} = i_{10}))) \wedge (s_{11} = s_{10}) \wedge (pc_{11} = 11)$$

$$f_{11} : (pc_{11} = 11) \wedge (s_{11} \geq 10) \wedge (pc_{12} = 12)$$

Recall: BMC For Programs

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

```
 1: int i=4;
 2: int s=0;
 3:
 4: s+=i;
 5: if (i>0)
 6:     i--;

 7: assert(s<10);
 8: s+=i;
 9: if (i>0)
10:     i--;

11: assert(s<10);
```

$f_1 : (pc_1 = 1) \wedge (i_2 = 4) \wedge (pc_2 = 2)$

$f_2 : (pc_2 = 2) \wedge (i_3 = i_2) \wedge (s_3 = 0) \wedge (pc_3 = 3)$

$f_3 : (pc_3 = 3) \wedge (i_4 = i_3) \wedge (s_4 = s_3) \wedge (pc_4 = 4)$

$f_4 : (pc_4 = 4) \wedge (i_5 = i_4) \wedge (s_5 = s_4 + i_4) \wedge (pc_5 = 5)$

$f_5 : (pc_5 = 5) \wedge (i_6 = i_5) \wedge (s_6 = s_5) \wedge (pc_6 = 6)$

$$f_6 : (pc_6 = 6) \wedge (((i_6 > 0) \wedge (i_7 = i_6 - 1)) \vee ((i_6 \leq 0) \wedge (i_7 = i_6))) \wedge (s_7 = s_6) \wedge (pc_7 = 7)$$

$f_7 : (pc_7 = 7) \wedge (s_7 \geq 10) \wedge (pc_8 = 8)$

$f_8 : (pc_8 = 8) \wedge (i_9 = i_8) \wedge (s_9 = s_8 + i_8) \wedge (pc_9 = 9)$

$f_9 : (pc_9 = 9) \wedge (i_{10} = i_9) \wedge (s_{10} = s_9) \wedge (pc_{10} = 10)$

$$f_{10} : (pc_{10} = 10) \wedge (((i_{10} > 0) \wedge (i_{11} = i_{10} - 1)) \vee ((i_{10} \leq 0) \wedge (i_{11} = i_{10}))) \wedge (s_{11} = s_{10}) \wedge (pc_{11} = 11)$$

$f_{11} : (pc_{11} = 11) \wedge (s_{11} \geq 10) \wedge (pc_{12} = 12)$

Model Representation For Unbounded Model Checking

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:

$$I(V_0) \wedge \Big[\bigwedge_{0 \leq i < k} T(V_i, V_{i+1}) \Big] \wedge E(V_k)$$

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

Definition (Craig's interpolant):
Let A, B be formulae such that $A \wedge B \rightarrow \perp$. Formula I is an interpolant for (A, B) iff

- $A \rightarrow I$,
- $I \wedge B \rightarrow \perp$, and
- $Var(I) \subseteq Var(A) \cap Var(B)$

Theorem (Craig, 1957)[1]:
For each pair of propositional formulae A, B such that $A \wedge B \rightarrow \perp$, there exists an interpolant I for (A, B).

[1] William Craig. (1957). Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. Journal of Symbolic Logic, 22(3):269-285. DOI: 10.2307/2963594

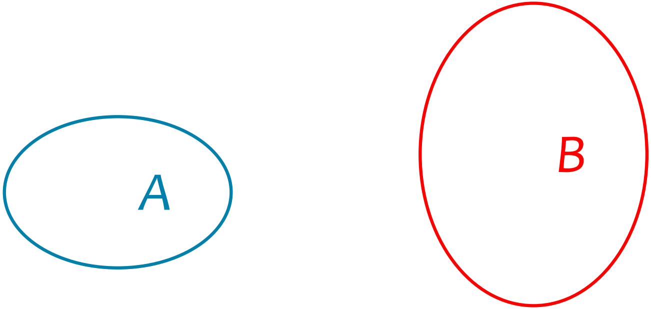
- For given formulae (A, B), interpolant is not unique
- Variability
  - complexity
    - number of connectives
    - number of unique variables
  - logical strength

Example:

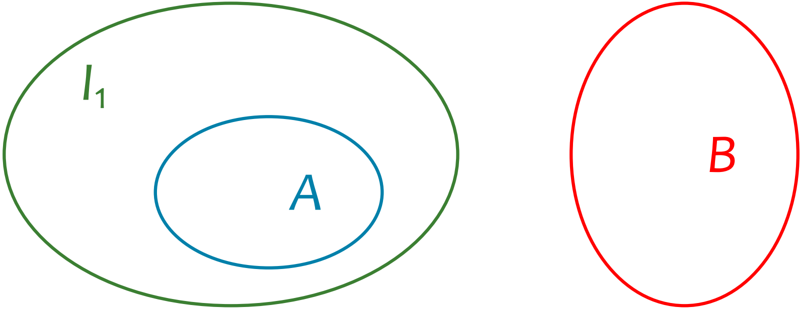
$$A = \{a_1\bar{a}_2, \bar{a}_1\bar{a}_3, a_2\}, B = \{\bar{a}_2 a_3, a_2 a_4, \bar{a}_4\}$$

- $I_1 = \bar{a}_3 \wedge a_2$
- $I_2 = \bar{a}_3$
- $I_3 = \bar{a}_3 \vee \bar{a}_2$

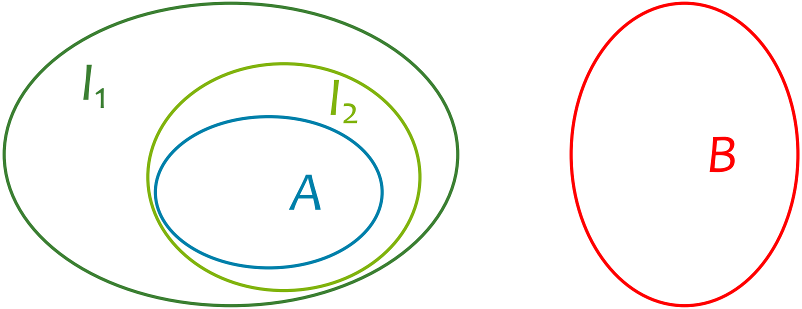
Graphical View of Interpolant



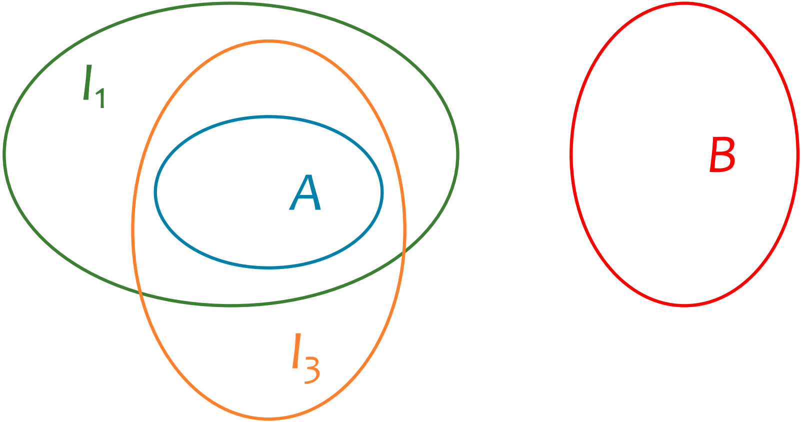
Graphical View of Interpolant



Graphical View of Interpolant



Graphical View of Interpolant



Interpolants As States Approximation

- 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

Interpolation-Based Model Checking

- 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
  $\Rightarrow$ **refine** over-approximation

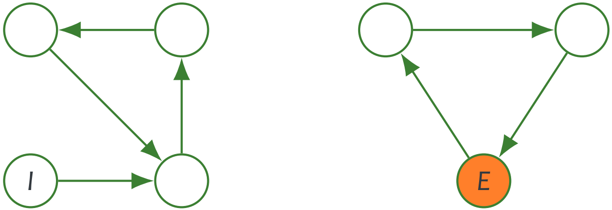
Safe Over-Approximation

Safe over-approximation of set of states represented by A w.r.t. E is formula O such that:

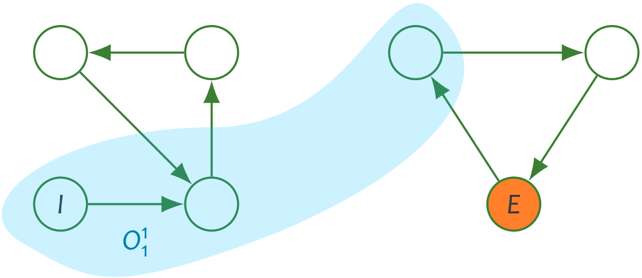
- $A \implies O$, and
- $O \cap E = \emptyset$

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

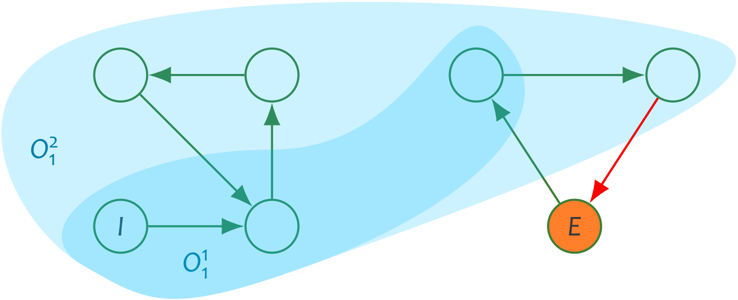
VERIFICATION EXAMPLE



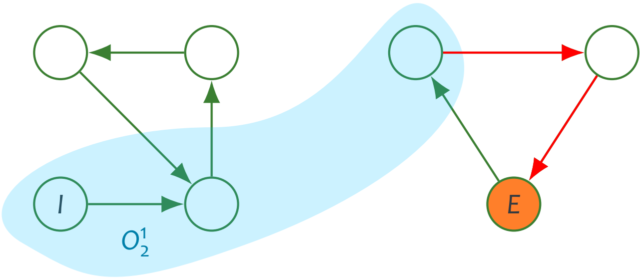
Verification Example



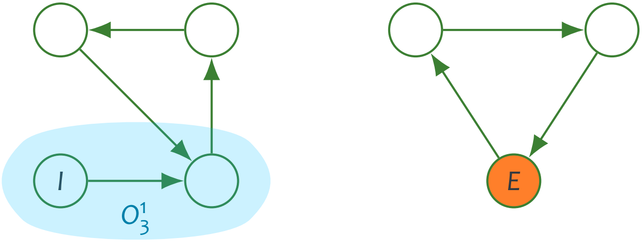
Verification Example



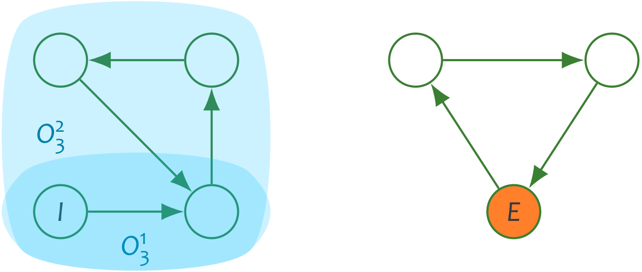
Verification Example



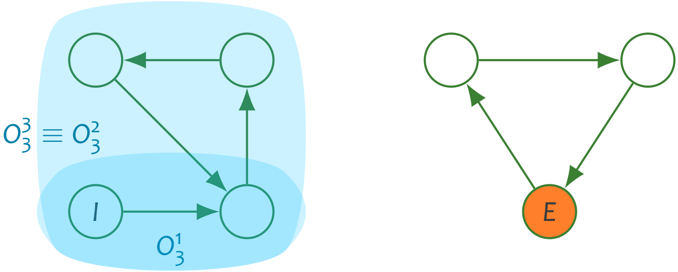
Verification Example



Verification Example



Verification Example



Approach Formally

Initial checks:

$I(V_0) \wedge E(V_0)$ is SAT $\implies$ error

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

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

$$\psi_k(S_i) \equiv S_i(V_0) \wedge \Big[\bigwedge_{0 \le i < k} T(V_i, V_{i+1}) \Big] \wedge E(V_k)$$

Splitting ψ_k for interpolation:

$$A \equiv S_i(V_0) \wedge T(V_0, V_1)$$

$$B \equiv \Big[\bigwedge_{1 \le i < k} T(V_i, V_{i+1}) \Big] \wedge E(V_k)$$

$S_{i+1} =$ interpolant for (A, B)

S_i is i-th over-approximation of I

VERIFICATION ALGORITHM

function VERIFY(M, E)
    if $I \cap E \neq \emptyset$ **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)
    $S_0 := I$
    $i := 0$
    while $\psi_k^M(S_i)$ is UNSAT **do**
        $(A, B) :=$ Split$(\psi_k^M(S_i))$
        $T_i :=$ Itp$(A, B)[V := V']$
        if $T_i \wedge \neg S_i$ is UNSAT **then return** safe
        end if
        $S_{i+1} := S_i \vee T_i$
        $i := i + 1$
    end while
    if $S_i = I$ **then return** unsafe
    else return unknown
    end if
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
  - 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/`