Formal Methods in Software Engineering

Propositional Logic — Spring 2025

Konstantin Chukharev

§1 Craig Interpolation in Module checking

Introduction into Craig Interpolation Theorem

Craig interpolation theorem states: If there's such a formulas A and B and $A \vDash B$ then exists such formula C that:

- $A \vDash C$
- $C \vDash B$
- C only consists of non-logical $A \wedge B$

Real-world examples:

- K-step BMC.
- TODO.
- TODO.

Semantic Proof of theorem

Let L_a be a language of A and L_b a language of B.

$L_{\text{AB}} = L_A \wedge L_B$

□ Example

Let $A = P(X) \wedge Q(X)$, $B = Q(Y) \vee R(Y)$, then

- $L_A = \{P, Q\}$
- $L_B = \{Q, R\}$
- $L_{\text{AB}} = \{Q\}$

□ Step 1

$G = \{\varphi \in L_{\text{AB}} \mid A \vDash \varphi\}$.

G - a set of formulas that are true if A is true.

Semantic Proof of theorem [2]

□ Step 2

We neeed to show, that if some model M satisfies all formulas from G, then it implies B automatically.

Let M be a model of G. ($\forall \gamma \in G : M \vDash \gamma$)

We are expanding M up to a M' by adding interpretation of symbols from $L_A \setminus L_{\text{AB}}$ so $M' \vDash A$

It is possible because G already contains all consequences of A in L_{AB}

□ Step 3

If $A \vDash B$, then $M' \vDash B$ because B depends only on symbols from L_B and M' equals with M on $L_{\text{AB}} \in L_B$, then $M \vDash B$

In result, **any model** of G satisfies B, means $G \vDash B$

By Theorem of compactness, if $G \vDash B$, then exists a finite set of $\varphi_1, \varphi_2, ..., \varphi_n$ from G such as that $(\varphi_1 \wedge \varphi_2 \wedge ...\varphi_n) \vDash B$

Then our **interpolant** is $C = \varphi_1 \wedge \varphi_2 \wedge ...\varphi_n$

Semantic Proof of theorem [3]

□ Checking of C

1. $A \vDash B$ because every formula in G is a consequence of A
2. $C \vDash B$ from proof
3. C only uses symbols from L_{AB}

Example: $A = P(x) \wedge Q(x), B = Q(y) \vee R(y)$

C —?

§2 SAT and usage Module checking

Definition 1 - Interpolant

- Let A and B be quantifier-free propositional formulas whose conjunction $A \wedge B$ is unsatisfiable. An interpolant is a quantifier-free propositional formula I such that $A \rightarrow I$ is valid, $I \wedge B$ is unsatisfiable, and the variables occurring in I are a subset of the variables common to A and B.

Intuitively, if A represents reachable states and B unsafe or bad states, then the interpolant I safely overapproximates A, a property frequently used in the context of fixed-point detection. (More about it later)

Definition 2 - Interpolant sequence

Let $\langle A_1, A_2, ..., A_n \rangle$ be an ordered sequence of propositional formulas such that the conjunction $\bigwedge_{i=1}^{n} A_i$ is unsatisfiable. An interpolation sequence is a sequence of formulas $\langle I_0, I_1, ..., I_n \rangle$ such that all of the following conditions hold.

1. $I_0 = T$ and $I_n = F$.
2. For every $0 \leq j < n$, $I_j \wedge A_{j+1} \rightarrow I_{j+1}$ is valid.
3. For every $0 < j < n$, it holds that the variables in I_j are a subset of the common variables of $\langle A_1, ..., A_j \rangle$ and $\langle A_{j+1}, ..., A_n \rangle$.

Definiton 3 - Finite-state Transition System

Given a *finite* transition system $M = \langle V, I, T, P \rangle$, BMC is an iterative process for checking P in all initial paths up to a given bound on the length.

- **V** - a set if boolean variables. V induces a set of states $S \stackrel{\text{def}}{=} \mathbb{B}^{|V|}$, and a state $s \in S$ is an assignment to V and can be represented as a conjunction of literals that are satisfied in s. More generally, a formula over V represents the set of states in which it is satisfiable.

Given a formula F over V, we use F' to denote the corresponding formula in which all variables v∈V have been replaced with their counterparts $v' \in V'$. In the context of multiple steps of the transition system, we use V^i instead of V' to denote the variables in V after i steps. Given a formula F over V^i, the formula $F[V^i \leftarrow V^j]$ is identical to F except that for each variable $v \in V$, each occurrence of v^i in F is replaced with v^j. This substitution allows us to change the execution step to which a formula refers.

Definiton 3 - Finite-state Transition System [2]

- **Initial states (I)** - A formula over V describing all possible starting configurations of the system.
- **Transition relation (T)** - A formula defining how the system moves from one state to the next. $(T(V, V'))$ is a formula over the variables V and their primed counterparts $V' = \{v' | v \in V\}$, representing starting states and successor states of the transition.
- **Safe states (P)** - A formula over V describing all possible safe states of the system.

Definition 4 - Forward reachability sequence

FRS - forward reachability sequence, denoted as such $\overline{F}_{[k]}$ is a sequence $\langle F_0, ..., F_k \rangle$ of propositional formulas over V such that the following holds:

- $F_0 = I$
- $F_i \wedge T \rightarrow F'_{i+1}$ for $0 \leq i < k$
- A reachability sequence $\overline{F}_{[k]}$ is monotonic if $F_i \rightarrow F_{i+1}$ for $0 \leq i < k$ and safe if $F_i \rightarrow P$ for $0 \leq i \leq k$.
- The individual propositional formulas F_i are called elements or frames of the sequence.

An element F_i in an FRS $\overline{F}_{[k]}$ represents an overapproximation of states reachable in i steps of the transition system. If the FRS $is\ monotonic$, then F_i is an overapproximation of all states reachable in at most i steps.

- **Fixpoint** - $\overline{F}$ is a fixpoint, if there is $0 < i \leq k$ and $F_i \rightarrow \bigvee_{i=0}^{k-1} F_i$

Monotonic FRS arise in the context of IC3 algorithm.

Definition 5 - Inductive invariant, Consecution

A set of states characterized by the formula F is inductive (satisfies consecution, respectively) if $F \wedge T \rightarrow F'$ holds. F is inductive relative to a formula G if $G \wedge F \wedge T \rightarrow F'$ holds. F satisfies initiation if $I \rightarrow F$, i.e., if F comprises all initial states. F is safe with respect to P if $F \rightarrow P$.

We say that F is an inductive invariant if it satisfies initiation and is inductive.

□ Lemma 1

Let $M = \langle V, I, T, P \rangle$ be a transition system and let F be a propositional formula representing a set of states. If F is an inductive invariant that implies P (i.e., $F \rightarrow P$ is valid), then P holds in M and M is said to be safe.

The following lemma highlights the relationship between inductive invariants and FRS.

□ Lemma 2

Let M be a transition system $\langle V, I, T, P \rangle$ and let $\overline{F}_{[k]}$ be an FRS. Let us define $F \stackrel{\text{def}}{=} \bigvee_{j=0}^{i} F_j$ where $0 \leq i < k$. Then, F is an inductive invariant if $F_{i+1} \rightarrow P$. In addition, if $\overline{F}_{[k]}$ is safe, then $F \rightarrow P$ holds, and thus M is safe.

Model checking algorithms

SAT-solver - checks satisfiability of a certain boolean formula.

Module checking - verification if model of a system is correct.

Module checking and SAT are connected by methods of **Symbolic verification and symbolic model checking**.

Symbolic model checking - an important hardware model checking technique. In symbolic model checking, sets of states and transition relations of circuits are represented as formulas of explicit sates.

Current design blocks with well-defined functionality have many thousands of state elements. To handle such a scale, researchers deployed SAT solvers, starting with the invention of SAT-based BMC.

Bounded model checking (BMC) relies on SAT solvers to exhaustively check hardware designs up to a limited depth.

BMC, Bounded model checking

A SAT solver either finds a satisfying assignment for a propositional formula or proves its absence. Using this terminology, BMC determines whether a transition system has a counterexample of a given length k or proves its absence.

BMC uses a SAT solver to achieve this goal. Given a transition system M, BMC translates the question "Does M have a counterexample of length k?" into a propositional formula and uses a SAT solver to determine if the formula is satisfiable or not.

If the solver finds a satisfying assignment, a counterexample exists and is represented by the assignment. If the SAT solver proves that no satisfying assignment exists, then BMC concludes that no counterexample of length k exists.

BMC, Bounded model checking [2]

In order to search for a counterexample of length k, the following propositional formula is built:

□ Easy formula

- $\text{BMC}(k) = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge ... \wedge T(s_{\text{k-1}}, s_k) \wedge (\psi(s_0) \vee \psi(s_1) \vee ... \vee \psi(s_k))$
  - $if\ satisfiable$: there's a counterexample
  - $if\ unsatisfiable$: there's no errors that are reachable in k steps

□ Hard one

- $\text{path}^{\text{i,j}} \stackrel{\text{def}}{=} T(V^i, V^{\text{i+1}}) \wedge ... \wedge T(V^{\text{j-1}}, V^j)$, where $0 \leq i < j$ and $j - i = k$. An initial path of length k is defined using the formula $I(V^0) \wedge \text{path}^{0,\text{k}}$.

In order to search for a counterexample of length k, the following propositional formula is built:

- $\varphi^k \stackrel{\text{def}}{=} I(V^0) \wedge \text{path}^{0,\text{k}} \wedge \left(\neg P(V^k)\right)$

BMC, Bounded model checking [3]

If BMC is **unsatisfiable**, then it may be splitted into two formulas:

1. $A = I(s_0) \wedge T(s_0, s_1)$
2. $T(s_{\text{k-1}}, s_k) \wedge (\psi(s_0) \vee \psi(s_1) \vee ... \vee \psi(s_k))$

Since $A \wedge B$ is **false**, then (by Craig Theorem) there's an interpolant A' such as:

1. $A \rightarrow A'$
2. $A' \wedge B$ is unsatisfiable
3. A' only uses common symbols from A and B (e.g. state variables s_i)

PBA, Proof-Based Abstraction

Abstraction is a widely-used method to mitigate the state explosion problem. Since the root of the problem is the need of model checking algorithms to exhaustively traverse the entire state space of the system, abstraction aims at reducing the state space by removing irrelevant details from the system. The irrelevant details of the system are usually determined by analyzing the checked property and finding which parts of the system are not necessary for the verification or refutation of that property.

A well-known SAT-based abstraction technique is **PBA**. One of the main advantages of SAT solvers is their ability to "zoom in" on the part that makes a CNF formula unsatisfiable. This part is referred to as the **unsatisfiable core (UC)** of the formula. If an unsatisfiable CNF formula is a set of clauses, then its UC is an unsatisfiable subset of this set of clauses. PBA uses the UC of an unsatisfiable BMC formula to derive an abstraction.

PBA, Proof-Based Abstraction [2]

Let φ^k be an unsatisfiable formula. Let's define a set

- $V_a = \{v | v^i \in \mathrm{Vars}(\mathrm{UC}(\varphi^k)), 0 \leq i \leq k\}$ as the set of variables from the transition system that appears in UC of φ^k.
- M_a - Abstract transition system derived from M by making all variables $v \in V \backslash V_a$ nondetermenistic.

PBA, Proof-Based Abstraction [3]

PBA is based on the BMC loop.

1. At each iteration, a BMC formula φ^k is checked. If the formula is satisfiable, then a counterexample is found. Otherwise, a UC is extracted, and an abstract transition system M_a is computed.
2. M_a is passed to a complete model checking algorithm for verification. If the property is proved using the abstract model, then the algorithm terminates concluding that the property holds. Otherwise, a counterexample is found. Note that if a counterexample is found in M_a, it may not exist in M (due to the abstraction).
3. A counterexample that exists in M_a but not in M is referred to as **spurious**. In the case of a **spurious** counterexample, PBA executes the next iteration of the BMC loop with a larger k.

Algorithms

Complete SAT-based model checking algorithms are predominantly based on a search for an inductive invariant. A popular approach is **k-induction**, which aims to find a bound k such that all states reachable via an initial path $I(V^0) \wedge \mathrm{path}_{0,\mathrm{k}}$ are safe, and that whenever P holds in k consecutive steps of the transition system, then P also holds in the subsequent step.

The model checking algorithms discussed below search for inductive invariants by means of FRS computation and the use of Lemma 2. In case an inductive invariant that implies P is found, M is reported to be safe.

ITP, Interpolation-Based Model Checking

ITP is a complete SAT-based model checking algorithm that relies on interpolation to compute the FRS and uses interpolation to synthesize an inductive invariant during the exploration.

□ Revisiting BMC

As we know, BMC formulates the question: "Does M have a counterexample of length k?" as a propositional formula φ^k. In a similar manner, BMC can also be formulated using the question "Does M have a counterexample of length i such that $1 \leq i \leq k$?" by using the following propositional formula:

- $\psi^k \stackrel{\text{def}}{=} I(V^0) \wedge \text{path}^{0,\,k} \wedge \left(\bigvee_{i=1}^{k} \neg P(V^i)\right)$

ITP uses nested loops where the inner loop computes a $safe$ FRS by repeatedly checking formulas of the form ψ^k with a fixed k, and the outer loop increases the bound k when needed. The safe FRS is computed inside the inner loop by extracting interpolants from unsatisfiable BMC formulas.

Inner loop of ITP

In general, the inner loop checks a fixed-bound BMC formula. At the first iteration, ψ^k is checked. If this BMC formula is satisfiable, then a counterexample exists and the algorithm terminates. If it is unsatisfiable, then the following expressions defined:

- $A \stackrel{\text{def}}{=} I(V^0) \wedge T(V^0, V^1)$
- $B \stackrel{\text{def}}{=} \text{path}^{1,\,\text{k}} \wedge \left(\bigvee_{\text{i}=1}^{k} \neg P(V^i)\right)$

By Definition 1 and interpolant I_1^k is extracted. To make sure it's an interpolant indeed:

1. It represents an overapproximation of the states from I after one transition ($A \rightarrow I_1^k$)
2. No counterexample can be reached from I_1^k in $k-1$ transitions or less ($I_1^k \wedge B$ is unsatisfiable), which also guarantees that $I_1^k \rightarrow P$.

This, the sequence $\left\langle I, I_1^k[V^1 \leftarrow V]\right\rangle$ is a valid FRS.

In the subsequent iterations, the formula $\psi^k\left[I \leftarrow I_{\text{j}-1}^k\right]$ is checked, where j is the iteration of the inner loop. Thus, in the jth iteration, if $\psi^k\left[I \leftarrow I_{\text{j}-1}^k\right]$ is unsatisfiable, an interpolant I_j^k is extracted with respect to the (A, B) pair where $A = I_{\text{j}-1}^k(V^1 \leftarrow V^0) \wedge T(V^0, V^1)$ and B is as before. Following this definition, I_j^k is an overapproximation of states reachable from $I_k^{\text{j}-1}$ in one transition and $\left\langle I, I_1^k, ..., I_j^k\right\rangle$ is a safe FRS.

Inner loop of ITP [2]

The inner loop terminates either when the BMC formula it checks is satisfiable, or when an inductive invariant is found. In the latter case, the algorithm terminates concluding that the transition system is safe. In the former case, there are two cases to handle: If the BMC formula is satisfiable in the first iteration, a counterexample exists and the algorithm terminates, otherwise, the control is passed back to the outer loop, which increases k.

Outer loop of ITP

After the first iteration of the inner loop, overapproximated sets of reachable states are used as the initial condition of the checked BMC formulas. Even if formula is satisfiable, it is not clear if it is due to the existence of a counterexample or due to the overapproximation. To make calculation more precise, puter loop increases the bound k used for the BMC queries. Increasing k helps to either find a real counterexample or to increase the precision of the overapproximation.

Interpolation Sequence-Based Model Checking

ISB was suggested for a computation of a safe FRS as a part of the main BMC loop. Unlike ITP, ISB has no nested loops and integrated into BMC's main loop.

It's pretty much the same with ITP in concept so there's a shortened version of how algorithm works.

- Starting point is $\langle F_0 = I \rangle$ as an FRS. At first operation it solves φ^1.
- At kth iteration, the FRS is $\langle F_0, \ldots, F_{\text{k-1}} \rangle$ and φ^k is checked. The goal of this step is to extend FRS with new element F_k. If φ^k is satisfiable, a counterexample is found and the algorithm terminates
- If it is not, an interpolation sequence $\langle I_0^k, \ldots, I_k^k \rangle$ is extracted and used to extend current FRS. The ith element of existing FRS is updated by defining:
  - $F_i = F_i \wedge I_i^k[V^i \leftarrow V]$ for $i \leq i < k$ and F_k to be $I_k^k\left(F_k = I_k^{k[V^k \leftarrow V]}\right)$.
- The result is a safe FRS of length k. If at the end of kth iteration an inductive invariant is found (Lemma 2), the algorithm terminates concluding that M is safe.

IC3

TODO(IC3)