Craig Interpolation in Model Checking

Propositional Logic — Spring 2025

Zabrodin Ivan

§1 Craig Interpolation in Model checking

Introduction into Craig Interpolation Theorem

Theorem 1 (Craig interpolation theorem): 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.
- ISB.
- IC3.

Semantic Proof of theorem

□ Step 1

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

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

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

Semantic Proof of theorem [3]

□ 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 [4]

□ Checking of C

1. $A \vDash C$ 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 —?

Semantic Proof of theorem [5]

```
def craig_interpolate(A, B):
    """
    A, B: Input formulas where A ∧ B is unsatisfiable
    Returns: Interpolant I satisfying:
        1. A → I
        2. I ∧ B is unsat
        3. vars(I) ⊆ vars(A) ∩ vars(B)
    """
    if sat_solver(A ∧ B) == SAT:
        raise Exception("A ∧ B must be unsatisfiable")
    proof = get_proof(A ∧ B)
    I = extract_interpolant(proof)

    assert implies(A, I), "A does not imply I"
    assert unsat(I ∧ B), "I ∧ B is satisfiable"
    assert (vars(I) <= vars(A) & vars(B)), "I has extra variables"

  return I
```

§2 SAT and usage Model checking

Definition 1 - Interpolant

Definition 1: 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
- 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 - Interpolation sequence

Definition 2: 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. $I_0 = \mathbb{T}$ and $I_n = \mathbb{F}$.
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

Definition 3: Given a *finite* transition system $M = \langle V, I, T, P \rangle$:

- **V** - a set of 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 $v \in V$ 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 (an invariant or a condition that must always hold).

Example *(Simple transition system)*:

- $V = \{x, y\}$
- $I = (x = 0 \wedge y = 0)$
- $T = (x' = x + 1 \wedge y' = y + x)$
- $P = (y \geq 0)$

Definition 4 - Forward reachability sequence

Definition 4: **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

Definition 5:

- 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.

Definition 5 - Inductive invariant, Consecution [2]

Example *(Initiation)*: Let $I = (x = 0)$ (system starts with $x = 0$).

If $F = (x \geq 0)$, then $I \to F$ holds because $x = 0$ implies $x \geq 0$.

But if $F = (x > 0)$, initiation fails because $x = 0$ does not satifsy $x > 0$.

Example *(Inductivness/Consecution)*: Let $F = (x \geq 0)$ and $T = (x' = x + 1)$.

Check: $(x \geq 0) \wedge (x' = x + 1) \to (x' \geq 0)$?

- Yes, because if $x \geq 0$ then $x' = x + 1 \geq 0$.

Thus, F is inductive.

Example *(Inductive Relative)*: Let $F = (x \leq 10)$, $T = (x' = x + 1)$ and $G = (x \leq 9)$.

Check: $(x \leq 9) \wedge (x \leq 10) \wedge (x' = x + 1) \to (x' \leq 10)$?

- Yes, because if $x \leq 9$ then $x' = x + 1 \leq 10$.

Here, F is inductive relative to G, but not fully inductive.

Definition 5 - Inductive invariant, Consecution [3]

Example *(Safety)*: Let $F = (x \leq 5)$, $P = (x \leq 10)$.

Then $F \rightarrow P$ holds because $x \leq 5$ implies $x \leq 10$.

But if $F = (x \leq 15)$, then $F \rightarrow P$ fails.

Example *(Inductive Invariant)*: Let $I = (x = 0)$, $T = (x' = x + 1)$ and $P = (x \geq 0)$.

Let $F = (x \leq 10)$.

- **Initiation:** $I \rightarrow F$ holds $(x = 0 \geq 0)$.
- **Consecution:** $(x \geq 0) \wedge (x' = x + 1) \rightarrow (x' \geq 0)$ holds.

Thus, F is inductive invariant.

Lemmas

□ 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 $F_{i+1} \rightarrow F$. 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.

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

Model 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 states.

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

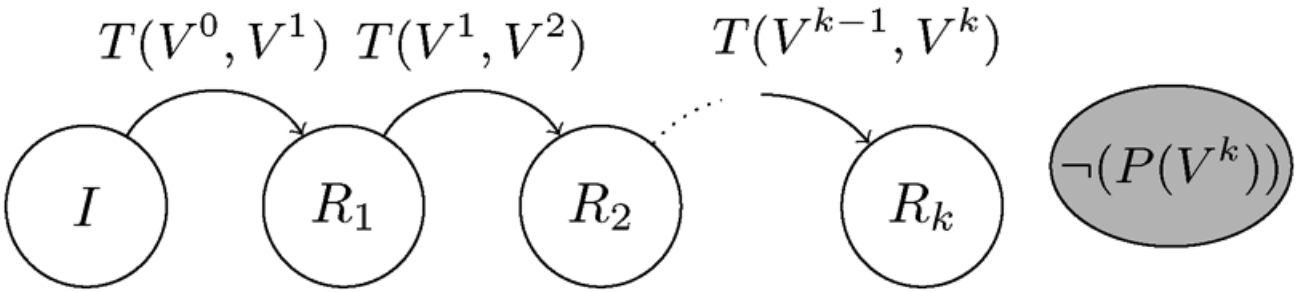
A SAT solver either finds a satisfying assignment for a propositional formula or proves its absence. Using this terminology, BMC determines whether a transition system has a counterexample of a given length k or proves its absence.

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]

Representation of how algorithm works



BMC checks whether a property P can be violated in k steps by encoding reachable sets of states $(R_1, ..., R_k)$ as a SAT instance. BMC does not identify repeatedly visited states and can therefore not determine whether the property holds for arbitrarily many steps.

BMC, Bounded model checking [3]

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

□ Easy formula

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

□ Hard one

Let $\mathrm{path}^{\mathrm{i,j}} \stackrel{\text{def}}{=} T(V^i, V^{\mathrm{i+1}}) \wedge ... \wedge T(V^{\mathrm{j\text{-}1}}, V^j)$, where $0 \leq i < j$ and $j - i = k$.

An initial path of length k is defined using the formula $I(V^0) \wedge \mathrm{path}^{0,\mathrm{k}}$, then:

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

BMC, Bounded model checking [4]

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

1. $A = I(s_0) \wedge T(s_0, s_1)$
2. $T(s_1, s_2) \wedge ... \wedge T(s_{\text{k-1}}, s_k) \wedge (\psi(s_0) \vee \psi(s_1) \vee ... \vee \psi(s_k))$

If $A \wedge B$ is **unsatisfiable** by SAT, 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)

This interpolant A' overapproximates reachable states in k transitions.

BMC, Bounded model checking [5]

```
def bounded_model_checking(M, P, max_depth):
    """
    M: Transition system (V, I, T)
    P: Safety property to verify
    max_depth: Maximum unrolling depth to check
    Returns: Verification result and counterexample if found
    """
    for k in range(0, max_depth + 1):
        # I ∧ T₀ ∧ T₁ ∧ ... ∧ T_{k-1} ∧ ¬P_k
        formula = []

        formula.append(M.I)  # I at time 0

        for i in range(k):
            # Instantiate T at each time step
            Ti = substitute(M.T, {'current': i, 'next': i+1})
            formula.append(Ti)
```

BMC, Bounded model checking [6]

```
        notPk = substitute(¬P, {'current': k})
        formula.append(notPk)

        cnf = convert_to_cnf(conjunction(formula))
        result = sat_solver(cnf)

        if result == SAT:
            trace = []
            for t in range(k+1):
                state = {}
                for var in M.V:
                    state[var] = get_assignment(var, t)
                trace.append(state)
            return ("Violation found at depth", k, trace)
    return ("Property holds up to depth", max_depth)
# Example Usage:
# M = TransitionSystem(V={'x'}, I='x==0', T='x_next == x + 1')
# result = bounded_model_checking(M, P='x <= 3', max_depth=5)
```

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.

Nondetermenistic means that the states of the variables **not from V_a** are not determined and they may have any value. Other variables in V_a are still bounded by the rules of model M.

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

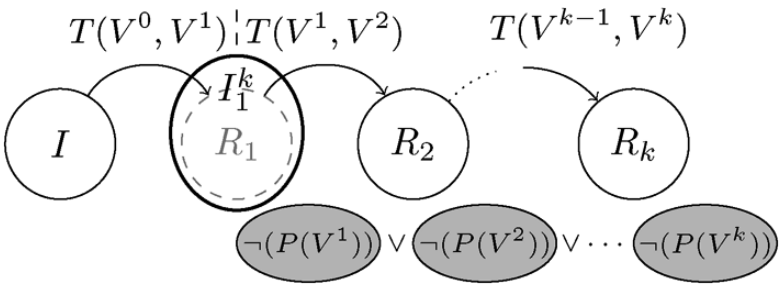
Complete SAT-based model checking algorithms are predominantly based on a search for an inductive invariant. A popular approach is **k-induction**, which aims to find a bound k such that all states reachable via an initial path $I(V^0) \wedge \mathrm{path}_{0,\mathrm{k}}$ are safe, and that whenever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

Representation of how algorithm works



Interpolation-based model checking partitions an unsatisfiable BMC instance into a formula A representing initial states and the first step, and a formula B representing the states from which a property P can be violated within $k-1$ steps. The interpolant I_1^k safely overapproximates all states reachable in a single step and is used to extend the FRS.

ITP, Interpolation-Based Model Checking [2]

□ 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:

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

By Definition 1 an 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 $I_1^k \rightarrow P$.

Inner loop of ITP [2]

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

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

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

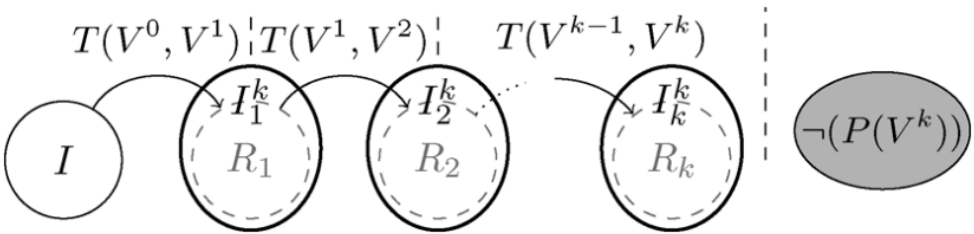
After the first iteration of the inner loop, overapproximated sets of reachable states are used as the initial condition of the checked BMC formulas. Even if formula is satisfiable, it is not clear if it is due to the existence of a counterexample or due to the overapproximation. To make calculation more precise, outer 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.

Note that B represents all bad states and all states that can reach a bad state in $k - 1$ transitions or less. Therefore, when k is increased, the precision of the computed interpolant is also increased. For a sufficiently large k, the approximation obtained through interpolation becomes precise enough such that the inner loop is guaranteed to find an inductive invariant if the system is safe, leading to the termination of the algorithm.

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.

Representation of how algorithm works



Interpolation sequence-based model checking partitions an unsatisfiable BMC instance into $k+1$ partitions (with the last one representing the "bad" states), resulting in the interpolants $I_1^k, \ldots, I_k^k$. Each I_i^k overapproximates the states reachable in i steps, and states in I_{i+1}^k are reachable from I_i^k in a single step.

Interpolation Sequence-Based Model Checking [2]

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

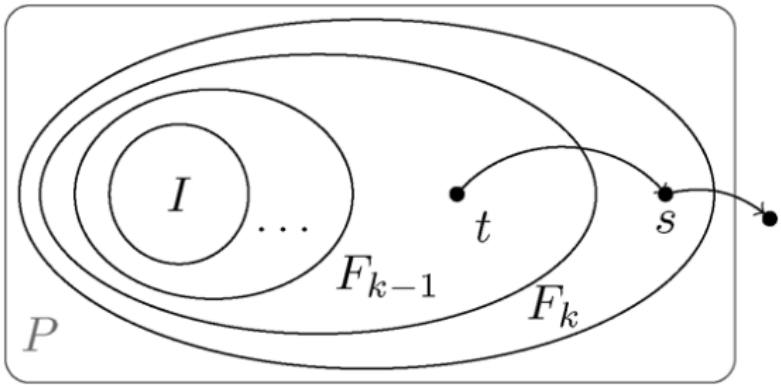
- Starting point is $\langle F_0 = I \rangle$ as an FRS. At first operation it solves φ^1.
- At kth iteration, the FRS is $\langle F_0, ..., F_{\text{k-1}} \rangle$ and φ^k is checked. The goal of this step is to extend FRS with new element F_k. If φ^k is satisfiable, a counterexample is found and the algorithm terminates
- If it is not, an interpolation sequence $\langle I_0^k, ..., I_k^k \rangle$ 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 $I_k^k(F_k = I_k^k[V^k \leftarrow V])$.
- 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

IC3 (Incremental Construction of Inductive Clauses) is a SAT-based algorithm that incrementally refines inductive invariants to prove safety properties. It operates on a Forward Reachability Sequence (FRS), blocking counterexamples via interpolation, and can integrate abstractions (e.g., FBA) for efficiency. Unlike BMC, IC3 avoids full unrolling, making it scalable for large systems.

Unlike interpolation-based techniques, IC3 does not depend on the unpredictable result of an interpolation engine.

IC3 [2]



IC3 maintains a monotonic sequence of frames $F_1, \ldots, F_k$ that overapproximate the states reachable in up to k steps. The approximation is iteratively refined by eliminating from frame F_i states t that are predecessors of a state in $\neg P$, but have themselves no predecessor in F_{i-1} (i.e., $\neg t$ is inductive relative to F_{i-1} and $F_{i-1} \wedge \neg t \wedge T \rightarrow \neg t'$ holds).

IC3 [3]

IC3′s search strategy is based on a backward search that starts from the unsafe (or “bad”) states in $\neg P$. The algorithm maintains a **monotonic safe** FRS $F_0, \ldots, F_k$, where each frame F_i overapproximates the set of states reachable from I in up to i steps of T. In addition, IC3 maintains a queue of states s occurring in the FRS from which a bad state is reachable (via a sequence of steps from s to a state in $\neg P$, which we call a bad suffix).

At each iteration, IC3 picks a state s from the queue, prioritizing states in frames with lower indices. Assume that s occurs in F_k. Then, IC3 tries to find a one-step predecessor to s in $F_{\mathrm{k}-1}$ in an attempt to extend the bad suffix until an initial state is reached. If the bad suffix is found to be unreachable (i.e., no predecessor t exists), then IC3 blocks the suffix using a process called inductive generalization.

The generalization technique yields a clause that is inductive relative to $F_{\mathrm{k}-1}$ and blocks s, which is then used to strengthen the frames $F_0, \ldots, F_k$ of the FRS. The algorithm terminates if either a counterexample is found or a frame is determined to be an inductive invariant that proves the property.

If that is not enough

- **Avy** - An interpolating IC3 (Y. Vizel, A. Gurfinkel: Interpolating Property Directed Reachability. CAV 2014)
- IC3 full algorithm (a lot of different literature, for example:
  - Somenzi_and_Bradley_2011_IC3_Where_monolithic_and_incremental_meet
  - Bradley (2012) - Understanding IC3)
- Hardware model checking competition(HWMCC)

Sources

- https://ieeexplore.ieee.org/document/7225110/figures#figures
- https://lara.epfl.ch/w/_media/sav08/mcmillaninterpolation.pdf