PDR

Property-Directed Reachability

Landon Taylor

College of Engineering
UtahStateUniversity

(or IC3)

1. **I**ncremental **C**onstruction of
2. **I**nductive **C**lauses for
3. **I**ndubitable **C**orrectness

Symbolic Transition System

Described by a set of logic formulas:

1. Initial condition (usually a single state) I
2. State variables x_n
3. Transitions $T(x, x')$

A Simple Symbolic Transition System

- State variables x_0, x_1
- State (conjunction of state variables) $x_0 = A, x_1 = B$
- Initial state $x_0 = 10, x_1 = 12$
- Transitions $T_0 : x_0' := x_0 + 1$ and $T_1 : x_1' := x_1 \times 3$

Literals

Literals describe a boolean variable or its negation.

- $x_0 = 4$
- $x_1 < 3$
- $\neg(x_1 = 100)$

Formulae

A formula $F(s)$ is a conjunction of literals.

For example: $F(s) = (x_0 = 4) \wedge (x_1 < 3) \wedge \neg(x_1 = 100)$

An assignment s to at least all variables in $F(s)$ either:

- satisfies the formula (causes $F = \texttt{true}$), notated as $s \models F(s)$
- does not satisfy the formula, notated as $s \not\models F$.

Reachability

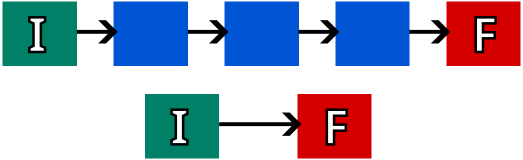
We often want to find out if a formula F (i.e. a *target state*)

- can be satisfied by any state in the model (satisfiability)
- can be reached from any state not satisfying F (inductive invariance)
- can be reached from a given state, usually I (reachability)

The focus of PDR is reachability analysis.

Inductive Reachability

If from I we can reach a state reaching a state reaching a state reaching a state reaching $S \vDash F$, then we can reach $S \vDash F$ from I.



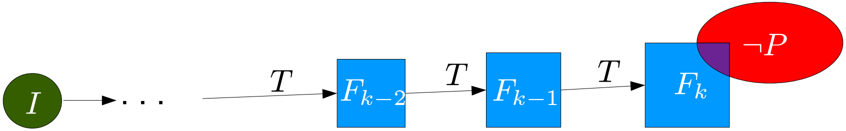
Goal of PDR

Assuming $F(s)$ is not already an inductive invariant,
Find an inductive invariant $\mathbf{P}(s)$ stronger than $F(s)$, such that

- $I \models \mathbf{G}(s)$
- $\mathbf{G}(s) \wedge T(s, s') \models \mathbf{G}(s')$
- $\mathbf{G}(s) \models F(s)$

The following slides are borrowed from “Interpolation in SMT and in Verification”, presented by Alberto Griggio at VTSA Summer School in 2015.

A (very) high level view of IC3



- Given a symbolic transition system and invariant property P, build an inductive invariant F s.t. $F \models P$
- Trace of formulae $F_0(X) \equiv I, \dots, F_k(X)$ s.t:
  - for $i > 0$, F_i is a set of clauses

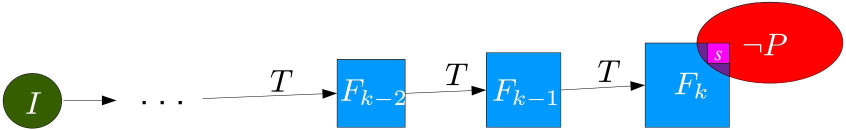
    overapproximation of states reachable in up to i steps

    $F_{i+1} \subseteq F_i$ (so $F_i \models F_{i+1}$)

    $F_i \wedge T \models F'_{i+1}$

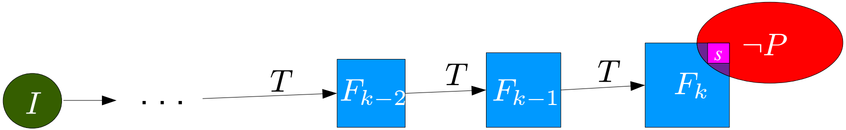
    for all $i < k$, $F_i \models P$

A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \wedge \neg s \wedge T \wedge s'$
    (i.e., check if $F_{k-1} \wedge \neg s \wedge T \models \neg s'$)

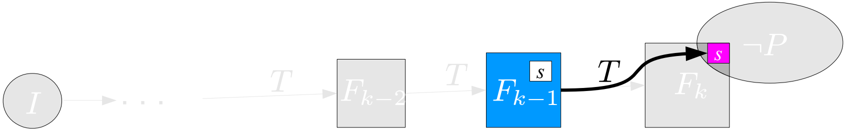
A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \wedge \neg s \wedge T \wedge s'$
    (i.e., check if $F_{k-1} \wedge \neg s \wedge T \models \neg s'$)

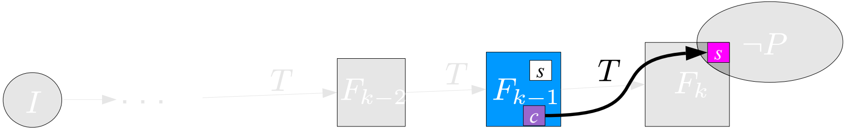
Check if s is inductive relative to F_{k-1}

A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \land \neg s \land T \land s'$
    (i.e., check if $F_{k-1} \land \neg s \land T \models \neg s'$)

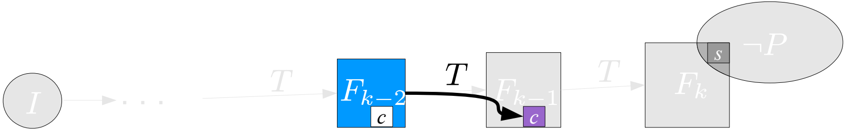
A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-1} \wedge \neg s \wedge T \wedge s'$
    - SAT: s is reachable from $F_{k-1} \wedge \neg s$ in 1 step
    - Get a cube c in the preimage of s and try (recursively) to prove it unreachable from F_{k-2}, …
      - c is a counterexample to induction (CTI)

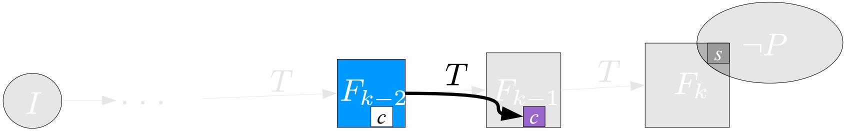
If I is reached, counterexample found

A (very) high level view of IC3



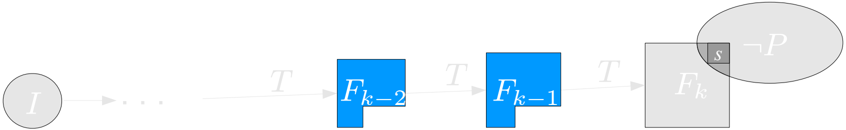
- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-2} \wedge \neg s \wedge T \wedge s'$

A (very) high level view of IC3



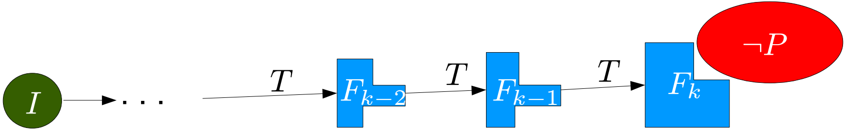
- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-2} \wedge \neg s \wedge T \wedge s'$
    - UNSAT: $\neg c$ is inductive relative to F_{k-2} $\quad F_{k-2} \wedge \neg c \wedge T \models \neg c'$
    - Generalize c to g and block by adding $\neg g$ to $F_{k-1}, F_{k-2}, \ldots, F_1$

A (very) high level view of IC3



- Blocking phase: incrementally strengthen trace until $F_k \models P$
  - Get bad cube s
  - Call SAT solver on $F_{k-2} \wedge \neg s \wedge T \wedge s'$
    - UNSAT: $\neg c$ is inductive relative to F_{k-2} $\quad F_{k-2} \wedge \neg c \wedge T \models \neg c'$
    - Generalize c to g and block by adding $\neg g$ to $F_{k-1}, F_{k-2}, \ldots, F_1$

A (very) high level view of IC3



Propagation: extend trace to F_{k+1} and push forward clauses

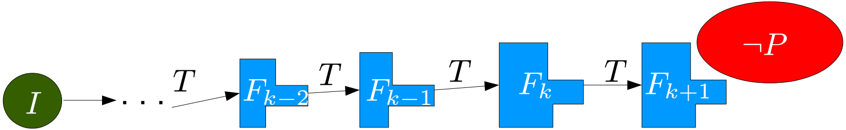
For each i and each clause $c \in F_i$:

Call SAT solver on $F_i \wedge T \wedge \neg c'$

If UNSAT, add c to F_{i+1}

$$F_i \wedge T \models c'$$

A (very) high level view of IC3



Propagation: extend trace to F_{k+1} and push forward clauses

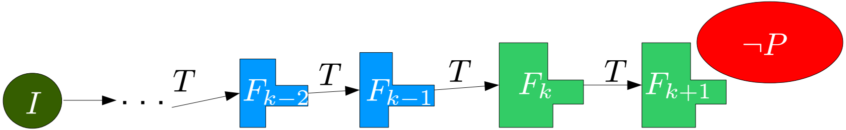
For each i and each clause $c \in F_i$:

Call SAT solver on $F_i \wedge T \wedge \neg c'$

If UNSAT, add c to F_{i+1}

$$F_i \wedge T \models c'$$

A (very) high level view of IC3



Propagation: extend trace to F_{k+1} and push forward clauses

For each i and each clause $c \in F_i$:

Call SAT solver on $F_i \wedge T \wedge \neg c'$

If UNSAT, add c to F_{i+1}

$$F_i \wedge T \models c'$$

If $F_i \equiv F_{i+1}$, P is proved,
otherwise start another round of blocking and propagation

IC3 pseudo-code

```
bool IC3(I, T, P):
    trace = [I]   # first elem of trace is init formula
    trace.push()  # add a new frame
    while True:
        # blocking phase
        while is_sat(trace.last() & ~P):
            c = extract_cube() # c |= trace.last() & ~P
            if not rec_block(c, trace.size()-1):
                return False # counterexample found

        # propagation phase
        trace.push()
        for i=1 to trace.size()-1:
            for each cube c in trace[i]:
                if not is_sat(trace[i] & ~c & T & c'):
                    trace[i+1].append(c)
            if trace[i] == trace[i+1]:
                return True # property proved
```

IC3 pseudo-code

```
bool rec_block(s, i):
    if i == 0:
        return False  # reached initial states
    while is_sat(trace[i-1] & ~s & T & s'):
        c = get_predecessor(i-1, T, s')
        if not rec_block(c, i-1):
            return False
    g = generalize(~s, i)
    trace[i].append(g)
    return True
```

Correctness (sketch)

- Consider the formula $F_{k-1} \wedge T \wedge s'$ where *s* is a bad cube
  - If UNSAT, then F_{k-1} is strong enough to block *s*
  - Since $F_i \wedge T \models F'_{i+1}$, then *s* is unreachable in *k* steps or less
  - Since $F_i \models F_{i+1}$, then we can add *s* to all $F_j, j \leq k$

Correctness (sketch)

- Consider the formula $F_{k-1} \wedge T \wedge s'$ where *s* is a bad cube
  - If UNSAT, then F_{k-1} is strong enough to block *s*
  - Since $F_i \wedge T \models F'_{i+1}$, then *s* is unreachable in *k* steps or less
  - Since $F_i \models F_{i+1}$, then we can add *s* to all $F_j, j \leq k$
- Consider now the relative induction check $F_{k-1} \wedge \neg s \wedge T \wedge s'$
  - We know that $I \equiv F_0 \not\models s$ because $I \models P$ (base case)
  - Since $F_i \models F_{i+1}$, then we know that $\neg s$ holds up to *k*

Correctness (sketch)

- Consider the formula $F_{k-1} \wedge T \wedge s'$ where *s* is a bad cube
  - If UNSAT, then F_{k-1} is strong enough to block *s*
  - Since $F_i \wedge T \models F'_{i+1}$, then *s* is unreachable in *k* steps or less
  - Since $F_i \models F_{i+1}$, then we can add *s* to all $F_j, j \leq k$
- Consider now the relative induction check $F_{k-1} \wedge \neg s \wedge T \wedge s'$
  - We know that $I \equiv F_0 \not\models s$ because $I \models P$ (base case)
  - Since $F_i \models F_{i+1}$, then we know that $\neg s$ holds up to *k*
- Propagation: for each $c \in F_i$, check $F_i \wedge T \wedge \neg c'$
  - we know that *c* holds up to *i*, if UNSAT then it holds up to *i+1*
  - since $F_i \models F_{i+1}$, $F_i \wedge T \models F'_{i+1}$ and $F_i \models P$,
    if $F_i \equiv F_{i+1}$ then the fixpoint is an inductive invariant

Inductive Clause Generalization

- Crucial step of IC3
- Given a relatively inductive clause $c \stackrel{\text{def}}{=} \{l_1, \ldots, l_n\}$ compute a generalization $g \subseteq c$ that is still inductive

$$F_{i-1} \wedge T \wedge g \models g' \qquad (1)$$

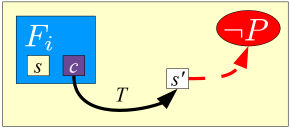
- Drop literals from c and check that (1) still holds
  - Accelerate with unsat cores returned by the SAT solver
    - Using SAT under assumptions
- However, make sure the base case still holds
  - If $I \not\models c \setminus \{l_j\}$, then l_j cannot be dropped

Simple iterative generalization

```
void indgen(c, i):
    done = False
    for iter = 1 to max_iters:
        if done:
            break
        done = True
        for each l in c:
            cand = c \ {l}
            if not is_sat(I & cand) and
               not is_sat(trace[i] & ~cand & T & cand'):
                c = get_unsat_core(cand)
                rest = cand \ c
                while is_sat(I & c):
                   l1 = rest.pop()
                   c.add(l1)
                done = False
                break
```

CTI computation

- When $F_i \land \neg s \land T \land s'$ is satisfiable:
  - s reaches $\neg P$ in k-i steps
  - s can be reached from F_i in 1 step
    - strengthen F_i by blocking cubes c in the preimage of s



- Extract CTI c from the SAT assignment
  - And generalize to represent multiple bad predecessors
  - Use unsat cores, exploiting a functional encoding of the transition relation
    - If T is functional, then $c \land \text{inputs} \land T \models s'$
    - check $\text{inputs} \land T \land \neg s'$ under assumptions c

SAT-based CTI generalization

```
void generalize_cti(cti, inputs, next):
    for i = 1 to max_iters:
        b = is_sat(cti & inputs & T & ~next')
        assert not b # assume T to be functional
        c = get_unsat_core(cti)
        if should_stop(c, cti):
            break
        cti = c
```

Access this presentation and a bibliography at:

`landonjtaylor.net/pdr`