IC3

Tuukka Korhonen

University of Helsinki, Seminar on Model Checking

November 19, 2019

Context

- Model checking *safety properties* on *bit-level* models
- Safety properties
  - Check that property holds in all reachable states
- Bit-level models
  - State is a bit-vector (Boolean variables)
  - Exponential number of states
  - Kripke structure and property given as CNF-SAT-formulas

Why IC3/PDR

- Published in 2010
- 3rd place in Hardware Model Checking Competition 2010
- Used by state-of-the-art model checkers today
- IC3 was a new idea compared to BMC and k-induction
  - No need to ”unroll” transitions
- IC3 is complete

Technical definitions

- Kripke structure (S, I, T), property P
- Input is 3 CNF-formulas:
  - $I(x)$ - true iff x is initial state
  - $T(x, x')$ - true iff there is transition from x to x'
  - $P(x)$ - true iff x satisfies the property
- A counterexample is $x_0, x_1, \ldots, x_n$ with
  - $I(x_0)$ is true
  - $T(x_i, x'_{i+1})$ is true for all i
  - $P(x_n)$ is false
- The task is to find a counterexample or prove that no counterexample exists

Idea of IC3

- To find an *inductive invariant* F that implies P
- Inductive invariant
  - includes initial states - $I \subset F$
  - closed under transition - $s \in F \wedge (s, s') \in T \implies s' \in F$
- Implies P
  - $s \in F \implies P(s)$
- F is *over-approximation* of states reachable from I

Frames

- IC3 maintain frames $F_0, F_1, \ldots, F_k$
- First frame is initial states $F_0 = I$
- Frame contains states reachable from previous frame
  - $F_i \subset F_{i+1}$
  - $s \in F_i \wedge (s, s') \in T \implies s' \in F_{i+1}$
  - i-th frame is over-approximation of frames reachable in $\leq i$ steps
- All frames except the last imply P
  - $s \in F_i \implies P(s)$ for all $i < k$
- Goal: construct frames such that $F_{k-1} = F_k$
  - $\Rightarrow F_k$ is inductive invariant that implies P

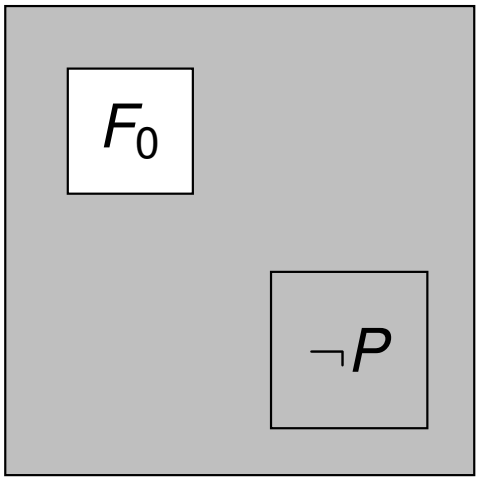
Frames

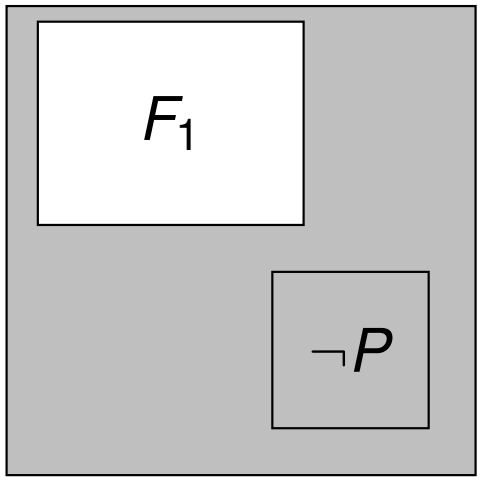
- Each frame represented as CNF formula $F_i(x)$
- $F_i \subset F_{i+1}$ holds by CNF syntax: $F_{i+1}(x) \subset F_i(x)$

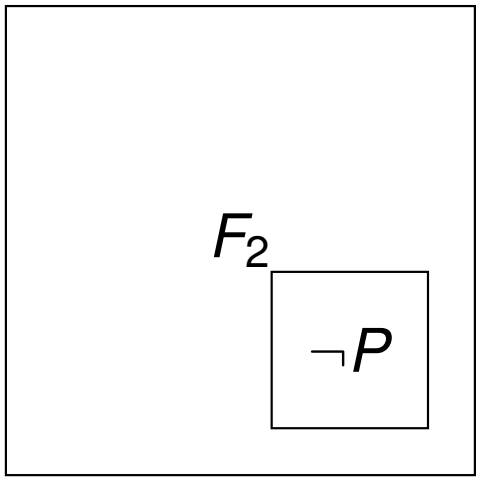
Constructing frames

- Add new frame $F_k = S$
- Repeat:
  - Find a state $s \in F_k$ that violates P
    - ★ SAT?$[F_k(x) \wedge \neg P(x)]$
  - Block states leading to s from earlier frames
    - ★ SAT?$[F_{k-1}(x) \wedge T(x, x') \wedge x']$
    - ★ Recursion
  - Block s from frame F_k

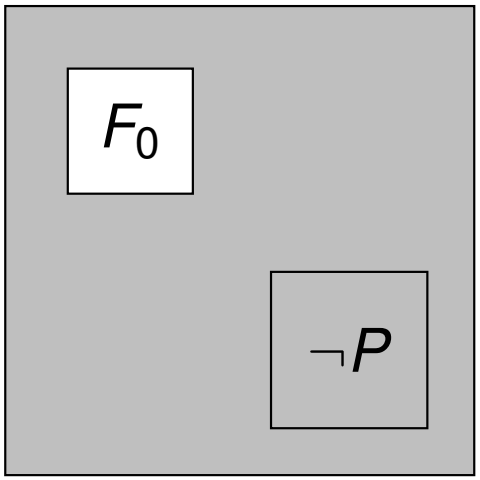
Example

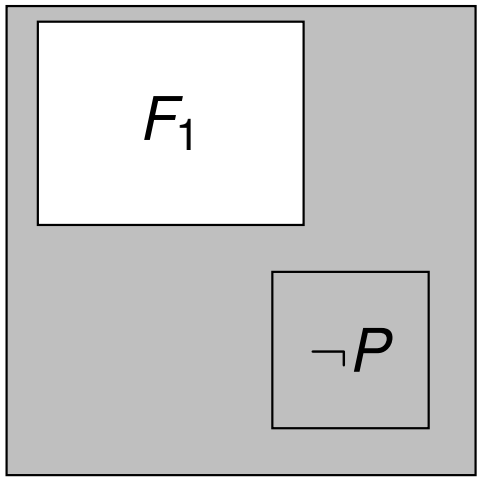


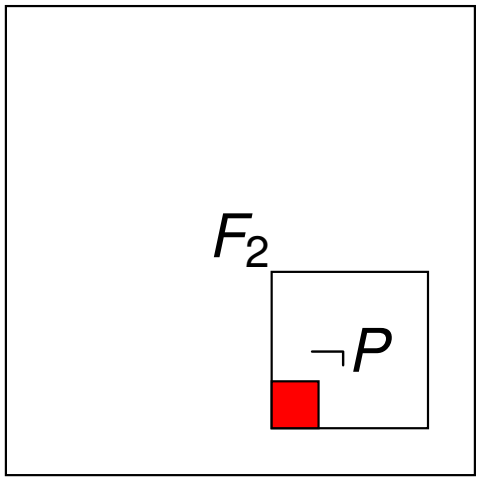




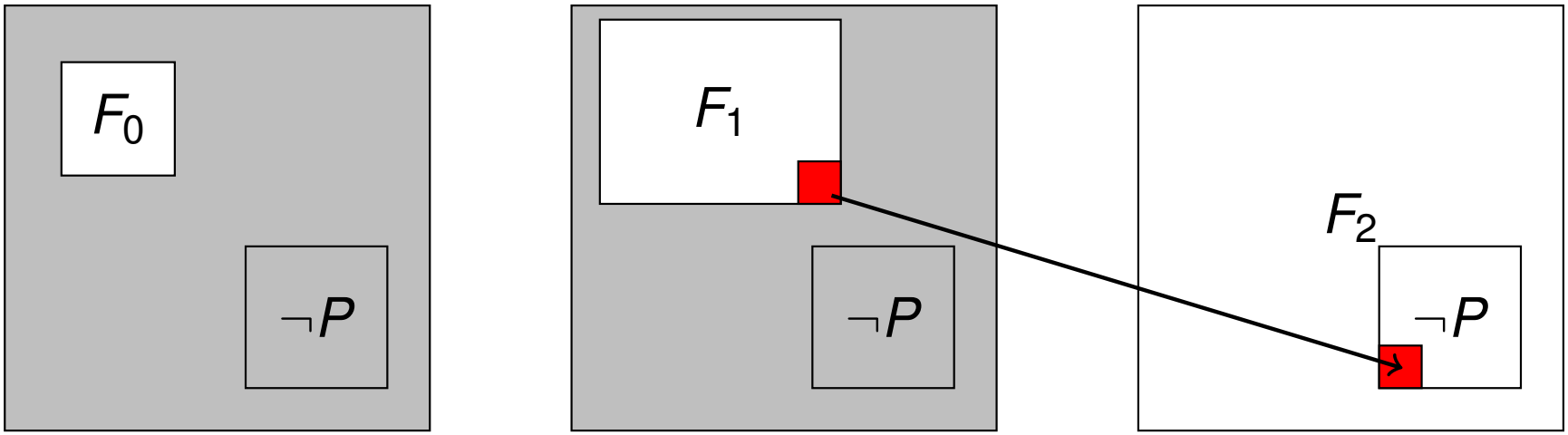
Example



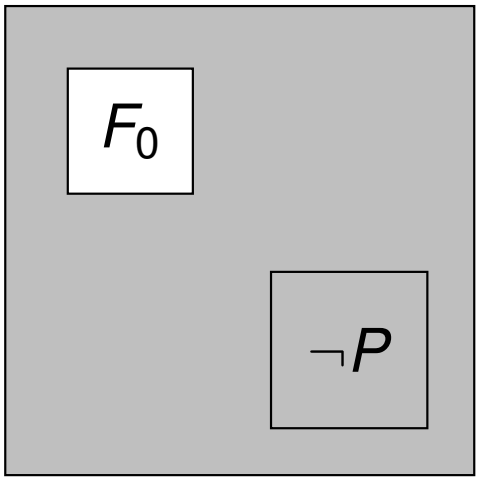


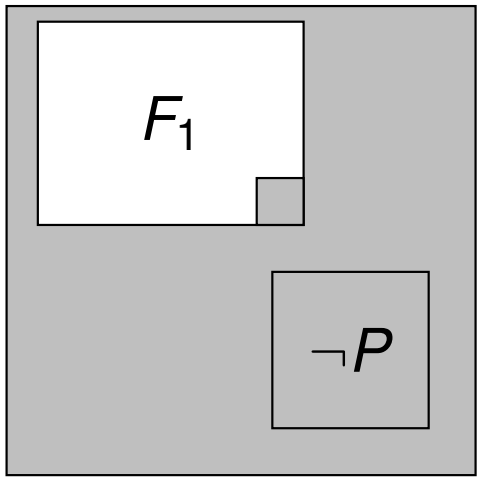


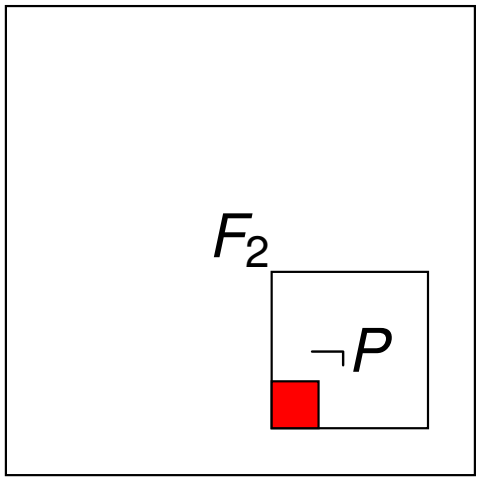
Example



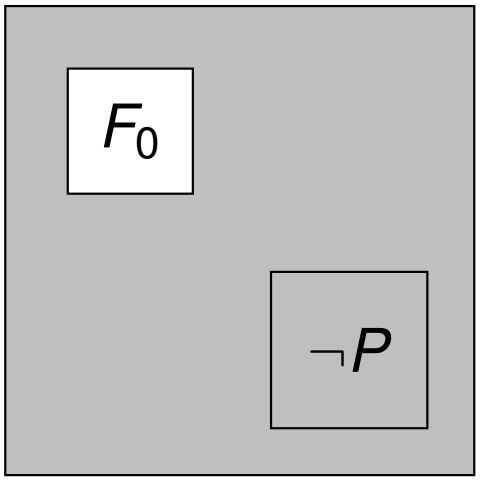
Example

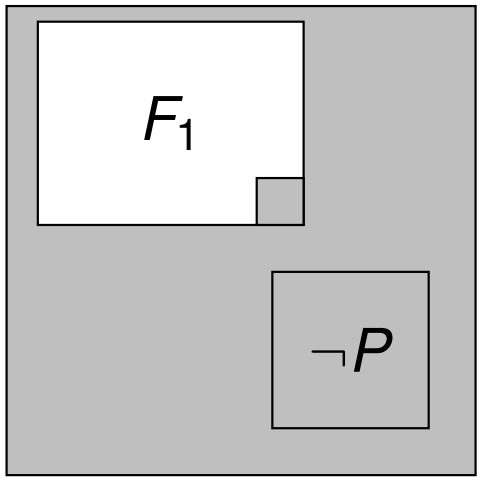


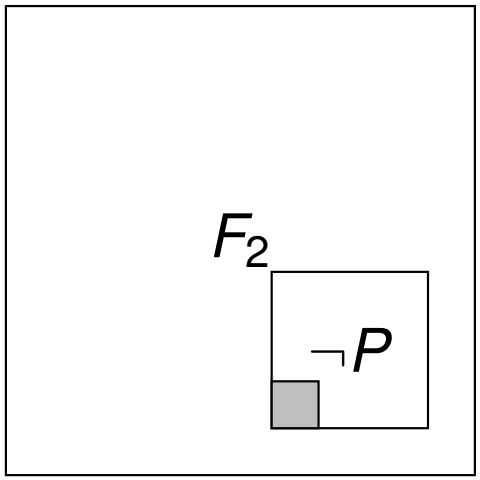




Example







Blocking states in CNF

- $m \leftarrow \text{SAT?}[F_k(x) \wedge \neg P(x)]$
- m is a *minterm*: assignment to variables x that represents a state
- Negation of m, $\neg m$, is a clause
- Blocking m in CNF:
  - $F_k(x) \leftarrow F_k(x) \cup \{\neg m\}$
- Example:
  - $m = (x \wedge \neg y \wedge z \wedge w)$
  - $\neg m = (\neg x \vee y \vee \neg z \vee \neg w)$

Generalizing states

- A *cube* is a subset of literals of *minterm*
- Cube represents many minterms
- Blocking a cube blocks many states
- Example:
  - $m_1 = (x \wedge \neg y \wedge z \wedge w)$
  - $m_2 = (x \wedge \neg y \wedge z \wedge \neg w)$
  - $m_3 = (x \wedge \neg y \wedge \neg z \wedge w)$
  - $m_4 = (x \wedge \neg y \wedge \neg z \wedge \neg w)$
  - $c = (x \wedge \neg y)$

Generalization in both SAT and UNSAT

- Generalization in both SAT and UNSAT
- SAT: Found a bad state that must be blocked
- UNSAT: Bad state can be safely blocked

SAT Generalization

$$(x \vee \neg z \vee w) \wedge (\neg y \vee \neg z) \wedge (\neg x \vee \neg y) \wedge (x \vee z \vee w)$$

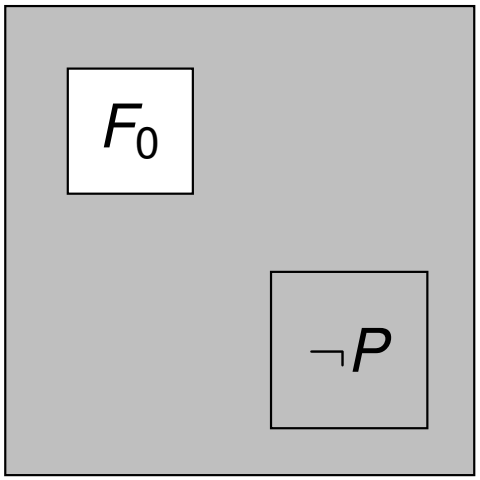
- Satisfied by minterm $m = (x \wedge \neg y \wedge z \wedge w)$
- m can be generalized to cube $c = (x \wedge \neg y)$

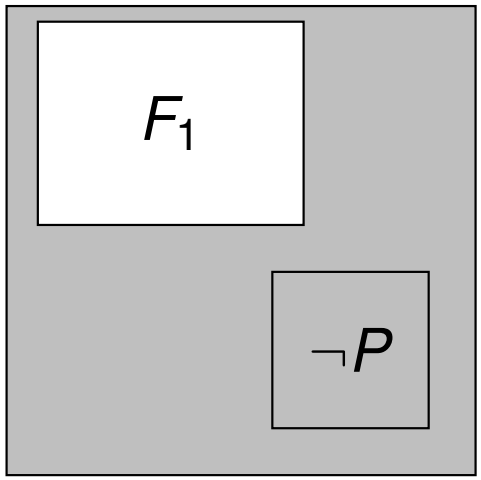
UNSAT Generalization

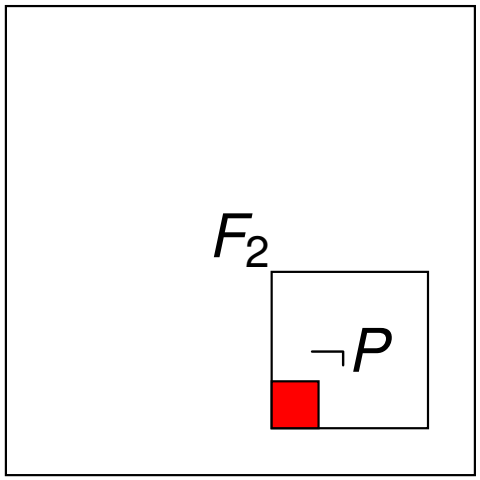
$$(x \vee \neg y) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y)$$

- UNSAT with assumption $(\neg x \wedge y \wedge z)$
- UNSAT also with assumption $(\neg x \wedge y)$

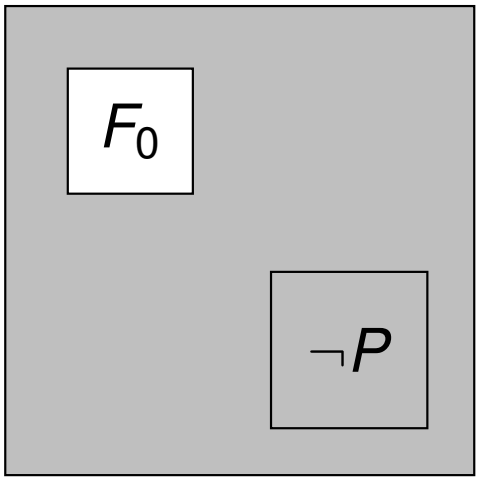
Generalization Example

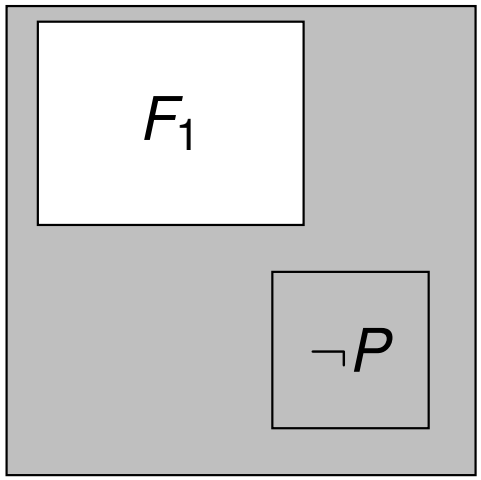


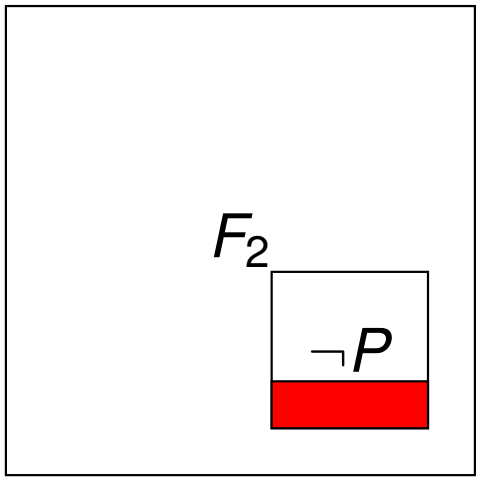




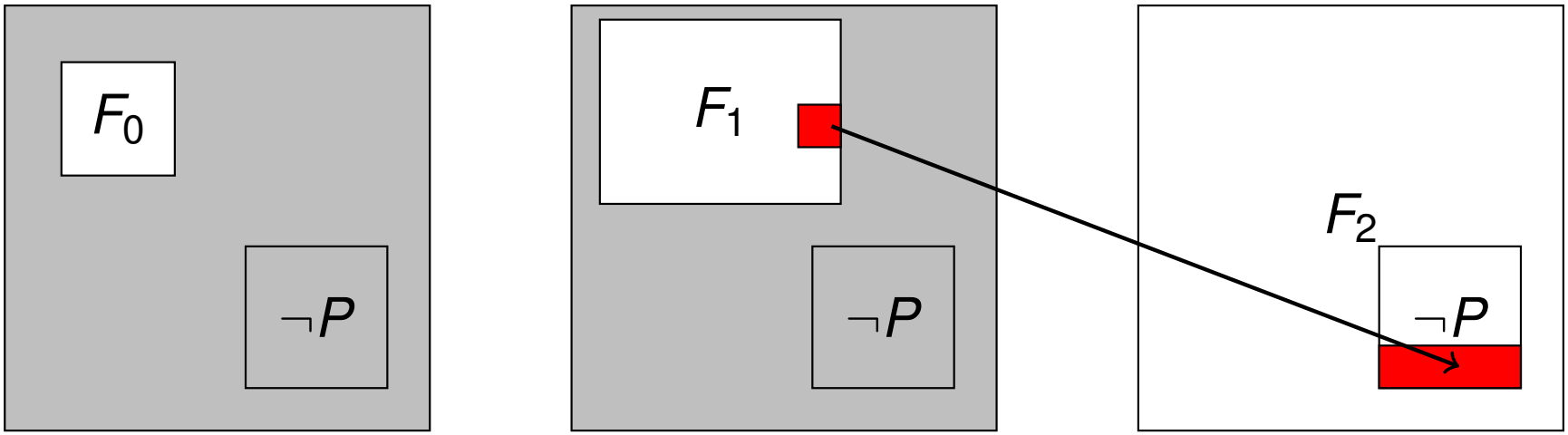
Generalization Example



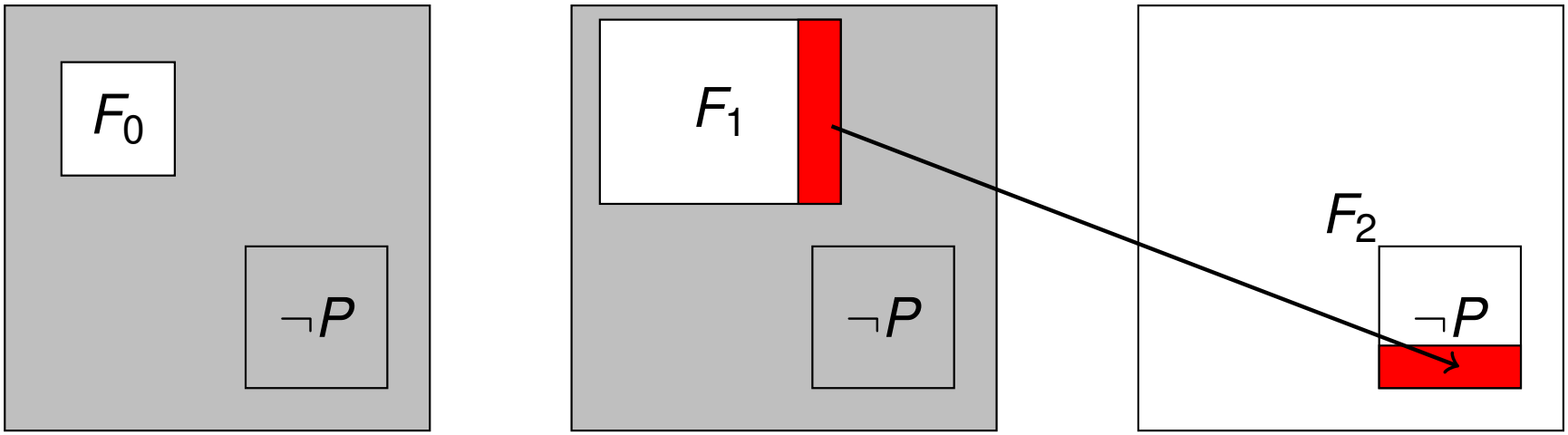




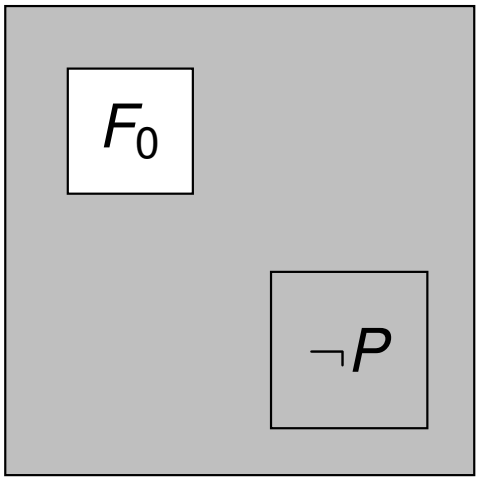
Generalization Example

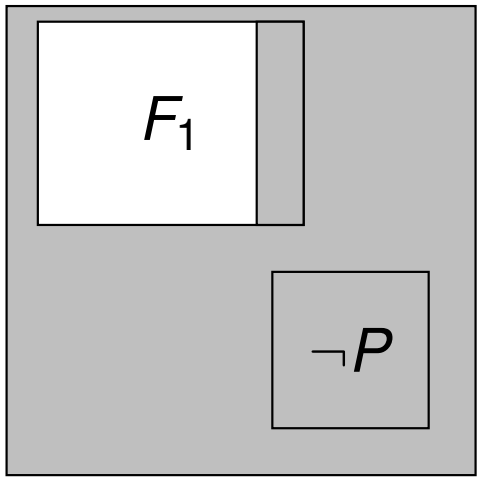


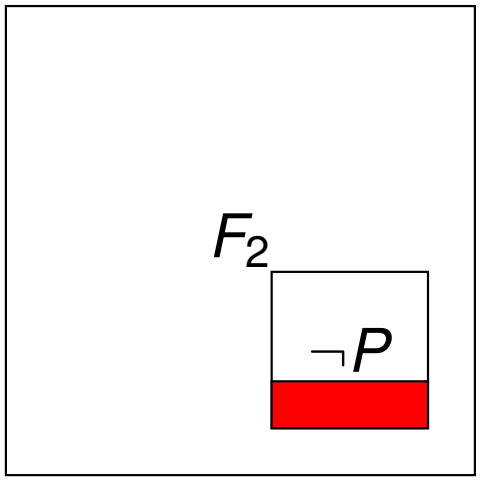
Generalization Example



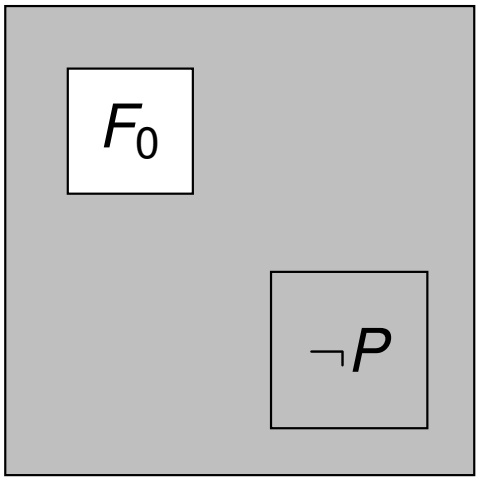
Generalization Example

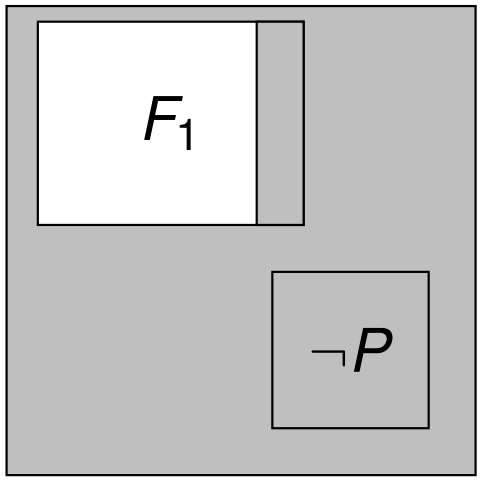


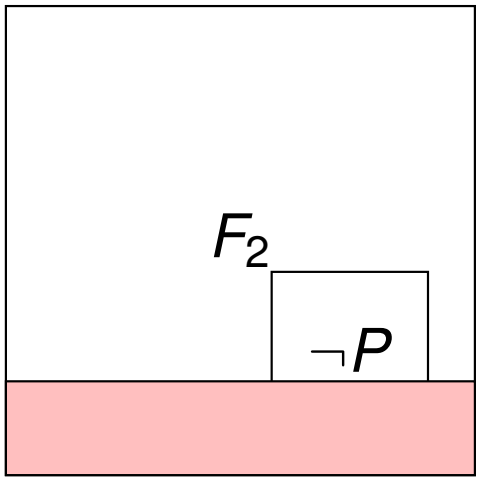




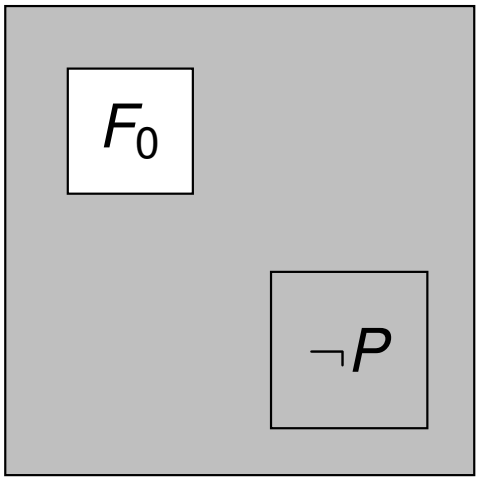
Generalization Example

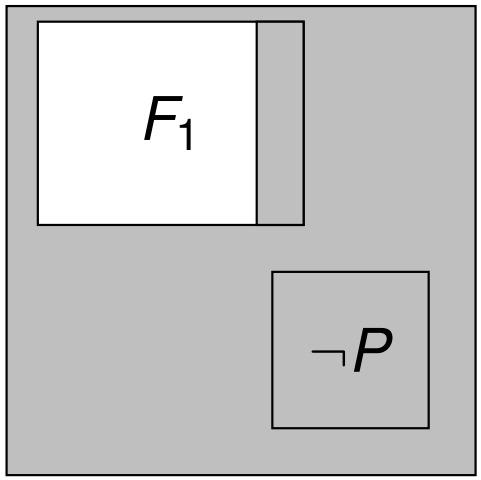


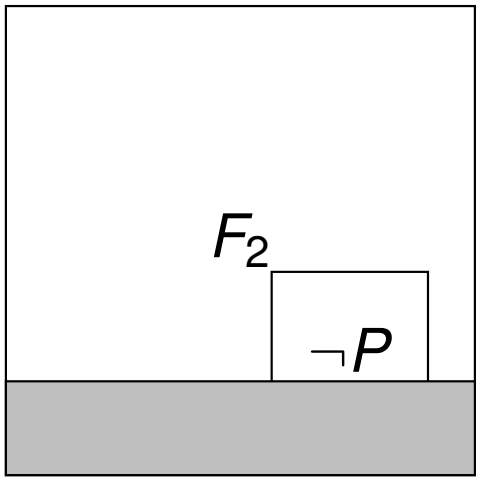




Generalization Example







Recursion

- Bad cube c in F_k
- Need to block all states that lead to c from F_{k-1}
- Repeat:
  - Find a state $s \in F_{k-1}$ that has transition to c
    - $m \leftarrow \text{SAT?}[F_{k-1}(x) \wedge T(x, x') \wedge c']$
  - Recurse
  - Block m from F_{k-1}
- Finally SAT?$[F_{k-1}(x) \wedge T(x, x') \wedge c']$=UNSAT
  - Generalize c' into a smaller cube c'' that is also UNSAT
  - Block c'' from F_k
- Need to block $s \in F_0$
  - Counterexample

Pseudocode

repeat
  while $m \leftarrow \text{SAT}[F_k(s) \wedge \neg P(s)]$ **do**
    counterexample $\leftarrow$ `rec-block`$(m, k, \neg P(s))$;
    if counterexample $\neq$ *null* **then**
      return False;
  if $F_{k-1}(s) = F_k(s)$ **then**
    return True;
  else
    $k \leftarrow k + 1$;
    $F_k(s) \leftarrow \{\}$;

Pseudocode

Procedure `rec-block(`m, i, $\mathcal{F}$`)`

- **if** $i = 0$ **then return** m as a counterexample;
- $c \leftarrow$ GeneralizeSAT(m, $\mathcal{F}$);
- **while** $m \leftarrow \text{SAT}[F_{i-1}(s) \wedge T(s, s') \wedge c']$ **do**
  - `rec-block(`m, $i - 1$, $T(s, s') \wedge c'$`)`;
- $c' \leftarrow$ GeneralizeUNSAT($F_{i-1}(s) \wedge T(s, s') \wedge c'$);
- $F_i(s) \leftarrow F_i(s) \cup \neg c'$;

Completeness

- We block only states from F_i that lead to $\neg P(s)$ or are not reachable in $\leq i$ steps
- Each call to `rec-block` blocks states from some frame
- So eventually $F_{k-1} = F_k$ must hold

Implementation details

- Frame CNF:s are represented as $F_i(x) = \bigwedge_{j \geq i} F'_i(x)$
  - Each clause stored only in the highest frame where it holds
- Incremental SAT-solver interface
  - Keeps learned clauses
  - Can control which parts of formula are ”active”
- Multiple ways to generalize minterms to cubes
  - Ternary simulation in Een et al.
  - Other tricks in original IC3

Optimizations

- Propagate all learned clauses from $F_i(x)$ to $F_{i+1}(x)$ if possible
  - Subsumption elimination
- Extra frame F_∞ that has blocked states that are not reachable from any F_i in any number of steps

IC3 in HWMCC

- IC3 3rd place in 2010
- Included in all 1st–3rd place solvers in HWMCC 2011-2017
  - superprove(ABC), PdTrav, iimc, v3, AVY, nuXmv
- IC3 also extended to liveness properties
- Notable solvers that seem to not include IC3:
  - AIGBMC, BLIMC, CoSA

Conclusion

- IC3 is arguably the most efficient bit-level safety property model checking algorithm
- IC3 utilizes repeated calls to SAT solver to construct an inductive invariant to prove a property