SAT-Based Model Checking

Fabio Somenzi

Department of Electrical, Computer, and Energy Engineering
University of Colorado at Boulder

SAT-SMT Summer School, 10 July 2014

Outline

1. A Short Intro to Model Checking
   - Structures
   - Properties
2. SAT Solver Interface
   - To The Solver
   - From The Solver
3. Checking Invariants
   - Bounded Model Checking
   - Interpolation
   - Proving Invariants by Induction
   - IC3: Incremental Inductive Verification
4. Progress Properties and Branching Time
   - Bounded Model Checking
   - Incremental Inductive Verification (FAIR and k-Liveness)
   - Model Checking CTL

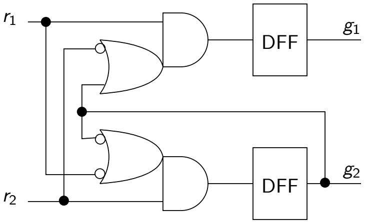
Outline

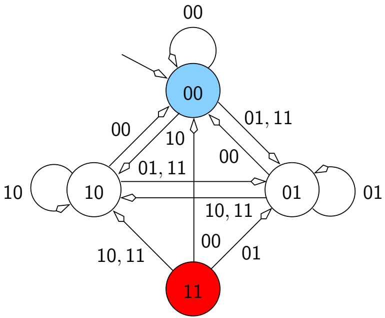
1. A Short Intro to Model Checking
   - Structures
   - Properties
2. SAT Solver Interface
   - To The Solver
   - From The Solver
3. Checking Invariants
   - Bounded Model Checking
   - Interpolation
   - Proving Invariants by Induction
   - IC3: Incremental Inductive Verification
4. Progress Properties and Branching Time
   - Bounded Model Checking
   - Incremental Inductive Verification (FAIR and k-Liveness)
   - Model Checking CTL

Simple Synchronous Arbiter (Verilog)

```
module arbsim (input clock, input [1:2] r, output reg [1:2] g);
    initial g <= 0;
    always @ (posedge clock) begin
        g[1] <= r[1] & (~r[2] | g[2]);
        g[2] <= r[2] & (~r[1] | ~g[2]);
    end
endmodule // arbsim
```

Mutual Exclusion for the Simple Arbiter





$$
\begin{aligned}
I(\overline{g}) &= \neg g_1 \wedge \neg g_2 \\
\exists r_1, r_2 \,.\, T(\overline{r}, \overline{g}, \overline{g}') &= \neg g_1' \vee \neg g_2' \\
P(\overline{g}) &= \neg g_1 \vee \neg g_2
\end{aligned}
$$

The Model Checking Question

- Given a structure S and a property φ, is S a model of φ?
- Written $S \models \varphi$
- More in detail: does φ hold for all computations of S?
    - From all initial states

Finite-State Transition Systems

Symbolic representation of a system:

$$S : (\bar{i},\ \bar{x},\ I(\bar{x}),\ T(\bar{i},\bar{x},\bar{x}'))$$

- $\bar{i}$: primary inputs
- $\bar{x}$: state variables
- $\bar{x}'$: next state variables
- $I(\bar{x})$: initial states
- $T(\bar{i},\bar{x},\bar{x}')$: transition relation

I and T define a finite transition structure (Kripke structure)

- Every valuation of $\bar{x}$ is a state
- $\exists \bar{i} \,.\, T(\bar{i},\bar{x},\bar{x}') = T(\bar{x},\bar{x}')$ defines the transitions

Composition

- Complex systems are composed of several modules
- Each module is described as a finite state structure S_i
- The overall Kripke structure is obtained as the product of the structures
    - State explosion!
- The product can be either synchronous or asynchronous (interleaving)

Examples of Temporal Logic Properties

- $\mathrm{G}\,p$: p is invariably true (always along all paths)
  - p is an atomic proposition
  - G is a temporal operator
- $\mathrm{F}\,p$: p is inevitably true (sometimes true along all paths)
- $p\,\mathrm{U}\,q$: q eventually holds and p holds until then
- $\mathrm{G}(p \rightarrow \mathrm{X}\,q)$: every p is immediately followed by a q
  - Only allowed if time is discrete
- $\mathrm{G}\,\mathrm{F}(p \rightarrow q)$: if p is persistent, then q is inevitable

Examples of Temporal Logic Properties

- $\mathrm{G}\,p$: p is invariably true (always along all paths)
  - p is an atomic proposition
  - G is a temporal operator
- $\mathrm{F}\,p$: p is inevitably true (sometimes true along all paths)
- $p\,\mathrm{U}\,q$: q eventually holds and p holds until then
- $\mathrm{G}(p \rightarrow \mathrm{X}\,q)$: every p is immediately followed by a q
  - Only allowed if time is discrete
- $\mathrm{G}\,\mathrm{F}(p \rightarrow q)$: if p is persistent, then q is inevitable

Examples of Temporal Logic Properties

- $\mathrm{G}\,p$: p is invariably true (always along all paths)
  - p is an atomic proposition
  - G is a temporal operator
- $\mathrm{F}\,p$: p is inevitably true (sometimes true along all paths)
- $p\,\mathrm{U}\,q$: q eventually holds and p holds until then
- $\mathrm{G}(p \to \mathrm{X}\,q)$: every p is immediately followed by a q
  - Only allowed if time is discrete
- $\mathrm{G}\,\mathrm{F}(p \to q)$: if p is persistent, then q is inevitable

Examples of Temporal Logic Properties

- $\mathrm{G}\, p$: p is invariably true (always along all paths)
  - p is an atomic proposition
  - G is a temporal operator
- $\mathrm{F}\, p$: p is inevitably true (sometimes true along all paths)
- $p \,\mathrm{U}\, q$: q eventually holds and p holds until then
- $\mathrm{G}(p \rightarrow \mathrm{X}\, q)$: every p is immediately followed by a q
  - Only allowed if time is discrete
- $\mathrm{G}\,\mathrm{F}(p \rightarrow q)$: if p is persistent, then q is inevitable

Examples of Temporal Logic Properties

- $\mathrm{G}\, p$: p is invariably true (always along all paths)
  - p is an atomic proposition
  - G is a temporal operator
- $\mathrm{F}\, p$: p is inevitably true (sometimes true along all paths)
- $p \,\mathrm{U}\, q$: q eventually holds and p holds until then
- $\mathrm{G}(p \rightarrow \mathrm{X}\, q)$: every p is immediately followed by a q
  - Only allowed if time is discrete
- $\mathrm{G}\,\mathrm{F}(p \rightarrow q)$: if p is persistent, then q is inevitable

Properties

- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic LTL (Linear-Time Logic)
- Syntactic sugar often useful (e.g., PSL, Property Specification Language)

Properties

- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic LTL (Linear-Time Logic)
- Syntactic sugar often useful (e.g., PSL, Property Specification Language)

Properties

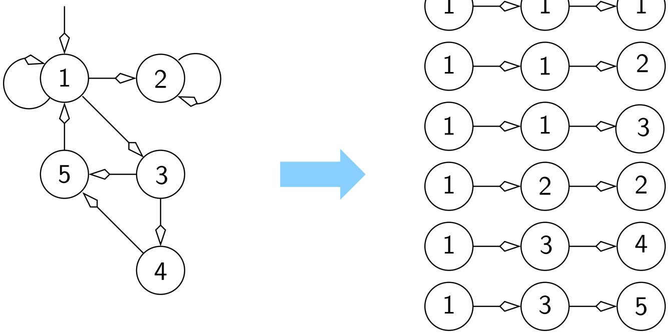
- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic LTL (Linear-Time Logic)
- Syntactic sugar often useful (e.g., PSL, Property Specification Language)

Properties

- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic LTL (Linear-Time Logic)
- Syntactic sugar often useful (e.g., PSL, Property Specification Language)

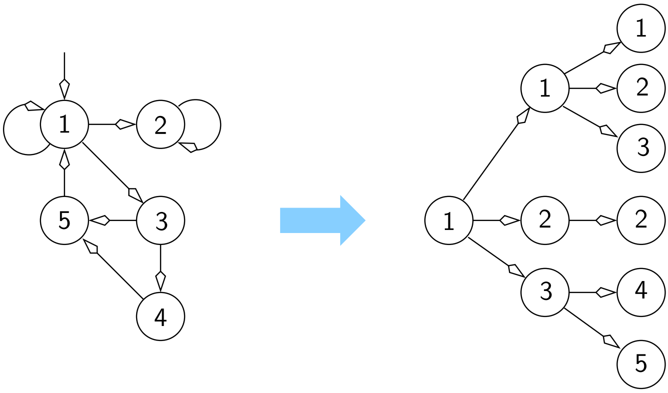
Linear Time

Linear time logics reason about sets of computation paths



Branching Time

Branching time logics reason about computation trees



Invariance, Safety, and Progress

- Invariance properties say that certain states are unreachable
  - Reachability analysis
- Safety properties say that certain events never happen
  - Generalize invariants and can be reduced to them
- Progress properties are the non-safety properties
  - Cycle detection (for finite state systems)
- Liveness (Alpern and Schneider [1985]) is related to progress, but not the same
- This can be made (a lot) more formal
  - Why is $\mathrm{G}(p \to \mathrm{X}\, q)$ a safety property, but $\mathrm{G}(p \to \mathrm{F}\, q)$ is not?
  - Borel hierarchy, (Landweber [1969])

Invariance, Safety, and Progress

- Invariance properties say that certain states are unreachable
  - Reachability analysis
- Safety properties say that certain events never happen
  - Generalize invariants and can be reduced to them
- Progress properties are the non-safety properties
  - Cycle detection (for finite state systems)
- Liveness (Alpern and Schneider [1985]) is related to progress, but not the same
- This can be made (a lot) more formal
  - Why is $\mathrm{G}(p \rightarrow \mathrm{X}\, q)$ a safety property, but $\mathrm{G}(p \rightarrow \mathrm{F}\, q)$ is not?
  - Borel hierarchy, (Landweber [1969])

Invariance, Safety, and Progress

- Invariance properties say that certain states are unreachable
  - Reachability analysis
- Safety properties say that certain events never happen
  - Generalize invariants and can be reduced to them
- Progress properties are the non-safety properties
  - Cycle detection (for finite state systems)
- Liveness (Alpern and Schneider [1985]) is related to progress, but not the same
- This can be made (a lot) more formal
  - Why is $\mathsf{G}(p \to \mathsf{X}\, q)$ a safety property, but $\mathsf{G}(p \to \mathsf{F}\, q)$ is not?
  - Borel hierarchy, (Landweber [1969])

Invariance, Safety, and Progress

- Invariance properties say that certain states are unreachable
  - Reachability analysis
- Safety properties say that certain events never happen
  - Generalize invariants and can be reduced to them
- Progress properties are the non-safety properties
  - Cycle detection (for finite state systems)
- Liveness (Alpern and Schneider [1985]) is related to progress, but not the same
- This can be made (a lot) more formal
  - Why is $\mathrm{G}(p \to \mathrm{X}\, q)$ a safety property, but $\mathrm{G}(p \to \mathrm{F}\, q)$ is not?
  - Borel hierarchy, (Landweber [1969])

Invariance, Safety, and Progress

- Invariance properties say that certain states are unreachable
  - Reachability analysis
- Safety properties say that certain events never happen
  - Generalize invariants and can be reduced to them
- Progress properties are the non-safety properties
  - Cycle detection (for finite state systems)
- Liveness (Alpern and Schneider [1985]) is related to progress, but not the same
- This can be made (a lot) more formal
  - Why is $\mathrm{G}(p \to \mathrm{X}\, q)$ a safety property, but $\mathrm{G}(p \to \mathrm{F}\, q)$ is not?
  - Borel hierarchy, (Landweber [1969])

Automata

- Properties may be described by automata that take the computation of the system as input and either accept it or reject it
- For non-terminating computations and linear-time properties we need ω-automata, which accept ω-regular languages
- For linear-time model checking we need the automaton for the negation of the property of interest
  - Model checking reduced to checking language emptiness of an ω-automaton

Automata

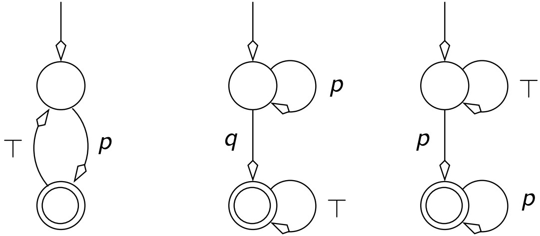
- Properties may be described by automata that take the computation of the system as input and either accept it or reject it
- For non-terminating computations and linear-time properties we need ω-automata, which accept ω-regular languages
- For linear-time model checking we need the automaton for the negation of the property of interest
  - Model checking reduced to checking language emptiness of an ω-automaton

Automata

- Properties may be described by automata that take the computation of the system as input and either accept it or reject it
- For non-terminating computations and linear-time properties we need ω-automata, which accept ω-regular languages
- For linear-time model checking we need the automaton for the negation of the property of interest
  - Model checking reduced to checking language emptiness of an ω-automaton

Omega-Automata

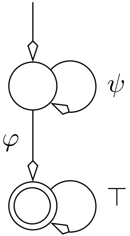
- ω-automata describe linear-time properties
  - Nondeterministic Büchi automata recognize all ω-regular properties
- Examples of Büchi automata (an accepting run visits some accepting state infinitely often)



- They are more expressive than LTL

From Formula to Büchi Automaton

$$\psi \,\mathsf{U}\, \varphi = \varphi \vee [\psi \wedge \mathsf{X}(\psi \,\mathsf{U}\, \varphi)]$$



- Expansion produces a DNF whose every term is the conjunction of:
  1. a propositional formula that must hold now and
  2. a temporal formula that must hold from the next step

Branching Time Temporal Logic

- Add path quantifiers to LTL to obtain CTL*
  - A: for all paths
  - E: for at least one path
- AG EF p: resetability
- LTL is embedded in CTL* by prepending A to all formulae
  - $\mathrm{AG}(p \rightarrow \mathrm{F}\, q)$
- $\mathrm{AG}(p \rightarrow \mathrm{F}\, q)$ is equivalent to $\mathrm{AG}(p \rightarrow \mathrm{AF}\, q)$, but...
- $\mathrm{AF}\,\mathrm{AG}\, p$ is not equivalent to $\mathrm{A}\,\mathrm{F}\,\mathrm{G}\, p$
- Maidl [2000] for more info
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathrm{AG}\,\varphi$, $\mathrm{A}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{AF}\,\varphi$, $\mathrm{AX}\,\varphi$, $\mathrm{EG}\,\varphi$, $\mathrm{E}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{EF}\,\varphi$, $\mathrm{EX}\,\varphi$

Branching Time Temporal Logic

- Add path quantifiers to LTL to obtain CTL*
  - A: for all paths
  - E: for at least one path
- $\mathrm{AG}\,\mathrm{EF}\,p$: resetability
- LTL is embedded in CTL* by prepending A to all formulae
  - $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$
- $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$ is equivalent to $\mathrm{AG}(p \rightarrow \mathrm{AF}\,q)$, but. . .
- $\mathrm{AF}\,\mathrm{AG}\,p$ is not equivalent to $\mathrm{A}\,\mathrm{F}\,\mathrm{G}\,p$
- Maidl [2000] for more info
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathrm{AG}\,\varphi$, $\mathrm{A}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{AF}\,\varphi$, $\mathrm{AX}\,\varphi$, $\mathrm{EG}\,\varphi$, $\mathrm{E}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{EF}\,\varphi$, $\mathrm{EX}\,\varphi$

Branching Time Temporal Logic

- Add path quantifiers to LTL to obtain CTL*
  - A: for all paths
  - E: for at least one path
- $\mathrm{AG}\,\mathrm{EF}\,p$: resetability
- LTL is embedded in CTL* by prepending A to all formulae
  - $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$
- $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$ is equivalent to $\mathrm{AG}(p \rightarrow \mathrm{AF}\,q)$, but...
- $\mathrm{AF}\,\mathrm{AG}\,p$ is not equivalent to $\mathrm{A}\,\mathrm{F}\,\mathrm{G}\,p$
- Maidl [2000] for more info
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathrm{AG}\,\varphi$, $\mathrm{A}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{AF}\,\varphi$, $\mathrm{AX}\,\varphi$, $\mathrm{EG}\,\varphi$, $\mathrm{E}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{EF}\,\varphi$, $\mathrm{EX}\,\varphi$

Branching Time Temporal Logic

- Add path quantifiers to LTL to obtain CTL*
  - A: for all paths
  - E: for at least one path
- $\mathrm{AG}\,\mathrm{EF}\,p$: resetability
- LTL is embedded in CTL* by prepending A to all formulae
  - $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$
- $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$ is equivalent to $\mathrm{AG}(p \rightarrow \mathrm{AF}\,q)$, but...
- $\mathrm{AF}\,\mathrm{AG}\,p$ is not equivalent to $\mathrm{A}\,\mathrm{F}\,\mathrm{G}\,p$
- Maidl [2000] for more info
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathrm{AG}\,\varphi$, $\mathrm{A}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{AF}\,\varphi$, $\mathrm{AX}\,\varphi$, $\mathrm{EG}\,\varphi$, $\mathrm{E}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{EF}\,\varphi$, $\mathrm{EX}\,\varphi$

Branching Time Temporal Logic

- Add path quantifiers to LTL to obtain CTL*
  - A: for all paths
  - E: for at least one path
- $\mathrm{AG}\,\mathrm{EF}\,p$: resetability
- LTL is embedded in CTL* by prepending A to all formulae
  - $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$
- $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$ is equivalent to $\mathrm{AG}(p \rightarrow \mathrm{AF}\,q)$, but...
- $\mathrm{AF}\,\mathrm{AG}\,p$ is not equivalent to $\mathrm{A}\,\mathrm{F}\,\mathrm{G}\,p$
- Maidl [2000] for more info
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathrm{AG}\,\varphi$, $\mathrm{A}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{AF}\,\varphi$, $\mathrm{AX}\,\varphi$, $\mathrm{EG}\,\varphi$, $\mathrm{E}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{EF}\,\varphi$, $\mathrm{EX}\,\varphi$

Branching Time Temporal Logic

- Add path quantifiers to LTL to obtain CTL*
  - A: for all paths
  - E: for at least one path
- $\mathrm{AG}\,\mathrm{EF}\,p$: resetability
- LTL is embedded in CTL* by prepending A to all formulae
  - $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$
- $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$ is equivalent to $\mathrm{AG}(p \rightarrow \mathrm{AF}\,q)$, but...
- $\mathrm{AF}\,\mathrm{AG}\,p$ is not equivalent to $\mathrm{A}\,\mathrm{F}\,\mathrm{G}\,p$
- Maidl [2000] for more info
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathrm{AG}\,\varphi$, $\mathrm{A}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{AF}\,\varphi$, $\mathrm{AX}\,\varphi$, $\mathrm{EG}\,\varphi$, $\mathrm{E}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{EF}\,\varphi$, $\mathrm{EX}\,\varphi$

Branching Time Temporal Logic

- Add path quantifiers to LTL to obtain CTL*
    - A: for all paths
    - E: for at least one path
- $\mathrm{AG}\,\mathrm{EF}\,p$: resetability
- LTL is embedded in CTL* by prepending A to all formulae
    - $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$
- $\mathrm{AG}(p \rightarrow \mathrm{F}\,q)$ is equivalent to $\mathrm{AG}(p \rightarrow \mathrm{AF}\,q)$, but...
- $\mathrm{AF}\,\mathrm{AG}\,p$ is not equivalent to $\mathrm{A}\,\mathrm{F}\,\mathrm{G}\,p$
- Maidl [2000] for more info
- In CTL every temporal operator must be immediately preceded by a path quantifier
    - $\mathrm{AG}\,\varphi$, $\mathrm{A}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{AF}\,\varphi$, $\mathrm{AX}\,\varphi$, $\mathrm{EG}\,\varphi$, $\mathrm{E}\,\psi\,\mathrm{U}\,\varphi$, $\mathrm{EF}\,\varphi$, $\mathrm{EX}\,\varphi$

Linear vs. Branching Time

- Branching time is more powerful, but less intuitive
  - Resetability
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ vs. $\mathsf{AF}\,\mathsf{AG}\,\varphi$
- Structure equivalence is finer-grained for branching time:
  - Linear time $\leftrightarrow$ language (trace) equivalence
  - Branching time $\leftrightarrow$ simulation relations
- Linear time is more suitable for compositional verification and Bounded Model Checking
- Counterexample generation simpler for linear time

Linear vs. Branching Time

- Branching time is more powerful, but less intuitive
  - Resetability
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ vs. $\mathsf{AF}\,\mathsf{AG}\,\varphi$
- Structure equivalence is finer-grained for branching time:
  - Linear time $\leftrightarrow$ language (trace) equivalence
  - Branching time $\leftrightarrow$ simulation relations
- Linear time is more suitable for compositional verification and Bounded Model Checking
- Counterexample generation simpler for linear time

Linear vs. Branching Time

- Branching time is more powerful, but less intuitive
  - Resetability
  - $\mathsf{A\,F\,G}\,\varphi$ vs. $\mathsf{AF\,AG}\,\varphi$
- Structure equivalence is finer-grained for branching time:
  - Linear time $\leftrightarrow$ language (trace) equivalence
  - Branching time $\leftrightarrow$ simulation relations
- Linear time is more suitable for compositional verification and Bounded Model Checking
- Counterexample generation simpler for linear time

Linear vs. Branching Time

- Branching time is more powerful, but less intuitive
  - Resetability
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ vs. $\mathsf{AF}\,\mathsf{AG}\,\varphi$
- Structure equivalence is finer-grained for branching time:
  - Linear time $\leftrightarrow$ language (trace) equivalence
  - Branching time $\leftrightarrow$ simulation relations
- Linear time is more suitable for compositional verification and Bounded Model Checking
- Counterexample generation simpler for linear time

Outline

1. A Short Intro to Model Checking
   - Structures
   - Properties
2. SAT Solver Interface
   - To The Solver
   - From The Solver
3. Checking Invariants
   - Bounded Model Checking
   - Interpolation
   - Proving Invariants by Induction
   - IC3: Incremental Inductive Verification
4. Progress Properties and Branching Time
   - Bounded Model Checking
   - Incremental Inductive Verification (FAIR and k-Liveness)
   - Model Checking CTL

From Hardware Description Language to CNF

- From source code to CDFG
- From CDFG to formulae over bit vectors and finite-domain variables
  - May involve abstraction
- Bit-blasting (binary encoding) to Boolean circuit plus memory elements
- Optimization of Boolean circuit
  - Often uses And-Inverter Graphs (AIGs) or similar data structures
- Conversion of circuit to CNF

From Hardware Description Language to CNF

- From source code to CDFG
- From CDFG to formulae over bit vectors and finite-domain variables
  - May involve abstraction
- Bit-blasting (binary encoding) to Boolean circuit plus memory elements
- Optimization of Boolean circuit
  - Often uses And-Inverter Graphs (AIGs) or similar data structures
- Conversion of circuit to CNF

From Hardware Description Language to CNF

- From source code to CDFG
- From CDFG to formulae over bit vectors and finite-domain variables
  - May involve abstraction
- Bit-blasting (binary encoding) to Boolean circuit plus memory elements
- Optimization of Boolean circuit
  - Often uses And-Inverter Graphs (AIGs) or similar data structures
- Conversion of circuit to CNF

From Hardware Description Language to CNF

- From source code to CDFG
- From CDFG to formulae over bit vectors and finite-domain variables
  - May involve abstraction
- Bit-blasting (binary encoding) to Boolean circuit plus memory elements
- Optimization of Boolean circuit
  - Often uses And-Inverter Graphs (AIGs) or similar data structures
- Conversion of circuit to CNF

From Hardware Description Language to CNF

- From source code to CDFG
- From CDFG to formulae over bit vectors and finite-domain variables
  - May involve abstraction
- Bit-blasting (binary encoding) to Boolean circuit plus memory elements
- Optimization of Boolean circuit
  - Often uses And-Inverter Graphs (AIGs) or similar data structures
- Conversion of circuit to CNF

Distributivity

- Apply $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$ systematically along with simplifications
- Preserves equivalence and does not introduce new variables
- Size may blow up
  - $(a \wedge b) \vee (c \wedge d) = (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$
  - $(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6) \vee \cdots$
- Seldom applied in its pure form

Distributivity

- Apply $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$ systematically along with simplifications
- Preserves equivalence and does not introduce new variables
- Size may blow up
  - $(a \wedge b) \vee (c \wedge d) = (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$
  - $(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6) \vee \cdots$
- Seldom applied in its pure form

Distributivity

- Apply $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$ systematically along with simplifications
- Preserves equivalence and does not introduce new variables
- Size may blow up
  - $(a \wedge b) \vee (c \wedge d) = (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$
  - $(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6) \vee \cdots$
- Seldom applied in its pure form

Distributivity

- Apply $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$ systematically along with simplifications
- Preserves equivalence and does not introduce new variables
- Size may blow up
  - $(a \wedge b) \vee (c \wedge d) = (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$
  - $(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6) \vee \cdots$
- Seldom applied in its pure form

Equisatisfiability

Two formulae F and G are *equisatisfiable* if

1. F is satisfiable iff G is satisfiable.
2. If η_F (η_G) is a satisfying assignment for F (G), there exists a satisfying assignment η_G (η_F) for G (F) that agrees with η_F (η_G) on all the variables that F and G have in common.

A common case occurs when one of the two formulae, say G, contains all the variables in the other formula. Then a satisfying assignment for F can be easily derived from one for G by dropping the extra variables.

Tseitin

Use definitions for subformulae

$$
\begin{aligned}
f &\leftrightarrow g \vee h \\
g &\leftrightarrow a \wedge b \\
h &\leftrightarrow c \wedge d
\end{aligned}
$$

Then, from $(a \wedge b) \vee (c \wedge d)$, we get

$$
\begin{aligned}
&(a \vee \neg g) \wedge (b \vee \neg g) \wedge (\neg a \vee \neg b \vee g) \\
&\qquad \wedge (c \vee \neg h) \wedge (d \vee \neg h) \wedge (\neg c \vee \neg d \vee h) \\
&\qquad\qquad \wedge (\neg g \vee f) \wedge (\neg h \vee f) \wedge (g \vee h \vee \neg f) \wedge f
\end{aligned}
$$

Simpler Equisatisfiable CNF Formulae

If the formula is in negation normal form, Tseitin's translation can be simplified (Plaisted and Greenbaum [1986])

$$
\begin{aligned}
f &\rightarrow g \vee h \\
g &\rightarrow a \wedge b \\
h &\rightarrow c \wedge d
\end{aligned}
$$

Then, from $(a \wedge b) \vee (c \wedge d)$, we get

$$
(a \vee \neg g) \wedge (b \vee \neg g) \wedge (c \vee \neg h) \wedge (d \vee \neg h) \wedge (g \vee h \vee \neg f) \wedge f
$$

More Conversions to CNF

- Wilson, Sheridan
- Nice DAGs
- Cut-based
- BDD-based
- SAT preprocessor

More Conversions to CNF

- Wilson, Sheridan
- Nice DAGs
- Cut-based
- BDD-based
- SAT preprocessor

Proofs of Unsatisfiability

Different verification techniques require

- Resolution proofs
- UNSAT cores
- Assumptions (unit clauses) in UNSAT cores
  - Can be extracted with minimal overhead (Eén and Sörensson [2003])

Incremental Solving

- Solve sequences of related SAT instances
- Ability to push and pop clauses (efficiently)
- Keep learned clauses that are still valid
    - All learned clauses remain valid if no clause is popped
- Keep variable scores
- Multiple solver objects

Outline

1. A Short Intro to Model Checking
   - Structures
   - Properties
2. SAT Solver Interface
   - To The Solver
   - From The Solver
3. Checking Invariants
   - Bounded Model Checking
   - Interpolation
   - Proving Invariants by Induction
   - IC3: Incremental Inductive Verification
4. Progress Properties and Branching Time
   - Bounded Model Checking
   - Incremental Inductive Verification (FAIR and k-Liveness)
   - Model Checking CTL

Bounded Model Checking

- A technique to falsify invariants ("bug finding")
- Based on unrolling the transition relation
- Looks for counterexamples of certain lengths
- May be extended to a complete method

Bounded Model Checking

- A technique to falsify invariants ("bug finding")
- Based on unrolling the transition relation
- Looks for counterexamples of certain lengths
- May be extended to a complete method

Bounded Model Checking

- A technique to falsify invariants ("bug finding")
- Based on unrolling the transition relation
- Looks for counterexamples of certain lengths
- May be extended to a complete method

Bounded Model Checking

- A technique to falsify invariants ("bug finding")
- Based on unrolling the transition relation
- Looks for counterexamples of certain lengths
- May be extended to a complete method

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(\overline{x}_0) \wedge \bigwedge_{0 \leq i < k} T(\overline{i}_i, \overline{x}_i, \overline{x}_{i+1}) \wedge \neg P(\overline{x}_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(\overline{x}_0) \wedge \bigwedge_{0 \leq i < k} T(\overline{i}_i, \overline{x}_i, \overline{x}_{i+1}) \wedge \neg P(\overline{x}_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(\overline{x}_0) \wedge \bigwedge_{0 \leq i < k} T(\bar{i}_i, \overline{x}_i, \overline{x}_{i+1}) \wedge \neg P(\overline{x}_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(\overline{x}_0) \wedge \bigwedge_{0 \leq i < k} T(\overline{i}_i, \overline{x}_i, \overline{x}_{i+1}) \wedge \neg P(\overline{x}_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete for finite state, is limited in practice to falsification
- BMC can prove that an invariant ψ holds on a model S only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $S \models \psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $S \models \psi$ (Clarke et al. [2004])

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete for finite state, is limited in practice to falsification
- BMC can prove that an invariant ψ holds on a model S only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $S \models \psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $S \models \psi$ (Clarke et al. [2004])

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete for finite state, is limited in practice to falsification
- BMC can prove that an invariant ψ holds on a model S only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $S \models \psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $S \models \psi$ (Clarke et al. [2004])

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete for finite state, is limited in practice to falsification
- BMC can prove that an invariant ψ holds on a model S only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $S \models \psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $S \models \psi$ (Clarke et al. [2004])

Finding The Bound κ

- Compute diameter of graph
  - Minimum d such that, if there is a path of length $d+1$ between two states, then there is a path of length at most d between the same states
  - $\forall \overline{x}_0, \ldots, \overline{x}_{d+1} . \bigwedge_{0 \leq i \leq d} T(\overline{x}_i, \overline{x}_{i+1}) \rightarrow \exists \overline{x}'_0, \ldots, \overline{x}'_d . (\bigwedge_{0 \leq i < d} T(\overline{x}'_i, \overline{x}'_{i+1}) \wedge \overline{x}'_0 = \overline{x}_0 \wedge \bigvee_{0 \leq i \leq d} \overline{x}'_i = \overline{x}_{d+1})$
- If one end of the path is constrained to an initial (target) state, one obtains the forward (backward) recursive radius of the graph
- Restrict search to simple paths (next slide)

Finding The Bound κ

- Compute diameter of graph
  - Minimum d such that, if there is a path of length $d+1$ between two states, then there is a path of length at most d between the same states
  - $\forall \overline{x}_0, \ldots, \overline{x}_{d+1} \,.\, \bigwedge_{0 \le i \le d} T(\overline{x}_i, \overline{x}_{i+1}) \rightarrow \exists \overline{x}'_0, \ldots, \overline{x}'_d \,.(\bigwedge_{0 \le i < d} T(\overline{x}'_i, \overline{x}'_{i+1}) \wedge \overline{x}'_0 = \overline{x}_0 \wedge \bigvee_{0 \le i \le d} \overline{x}'_i = \overline{x}_{d+1})$
- If one end of the path is constrained to an initial (target) state, one obtains the forward (backward) recursive radius of the graph
- Restrict search to simple paths (next slide)

Finding The Bound κ

- Compute diameter of graph
  - Minimum d such that, if there is a path of length $d+1$ between two states, then there is a path of length at most d between the same states
  - $\forall \overline{x}_0, \ldots, \overline{x}_{d+1} \,.\, \bigwedge_{0 \leq i \leq d} T(\overline{x}_i, \overline{x}_{i+1}) \rightarrow$ $\exists \overline{x}'_0, \ldots, \overline{x}'_d \,.(\bigwedge_{0 \leq i < d} T(\overline{x}'_i, \overline{x}'_{i+1}) \wedge \overline{x}'_0 = \overline{x}_0 \wedge \bigvee_{0 \leq i \leq d} \overline{x}'_i = \overline{x}_{d+1})$
- If one end of the path is constrained to an initial (target) state, one obtains the forward (backward) recursive radius of the graph
- Restrict search to simple paths (next slide)

Simple Paths

- A counterexample to an invariant is a finite prefix path to a state that satisfies $\neg P$ (bad state)
- If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state
- An invariant holds (Sheeran et al. [2000]) if:
  - there is no counterexample of length k to $\neg P$, and
  - no simple path of length $k+1$ to $\neg P$ that does not go through any other states satisfying $\neg P$, or
  - no simple path of length $k+1$ from an initial state that does not go through any other initial states

Simple Paths

- A counterexample to an invariant is a finite prefix path to a state that satisfies $\neg P$ (bad state)
- If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state
- An invariant holds (Sheeran et al. [2000]) if:
  - there is no counterexample of length k to $\neg P$, and
  - no simple path of length $k+1$ to $\neg P$ that does not go through any other states satisfying $\neg P$, or
  - no simple path of length $k+1$ from an initial state that does not go through any other initial states

Simple Paths

- A counterexample to an invariant is a finite prefix path to a state that satisfies $\neg P$ (bad state)
- If a counterexample exists, then there is a simple path from an initial state to a bad state that goes through no other initial or bad state
- An invariant holds (Sheeran et al. [2000]) if:
  - there is no counterexample of length k to $\neg P$, and
  - no simple path of length $k+1$ to $\neg P$ that does not go through any other states satisfying $\neg P$, or
  - no simple path of length $k+1$ from an initial state that does not go through any other initial states

Checking for Simple Paths

- Simple-minded check produces quadratic formula

$$\bigwedge_{0<i\leq k} \bigwedge_{0\leq j<i} (\overline{x}_i \neq \overline{x}_j)$$

- Using a bitonic sorting network (Kröning and Strichman [2003]) reduces the complexity to $O(k \log^2 k)$
- Lazy checking is more effective in practice (Sörensson's thesis)

Checking for Simple Paths

- Simple-minded check produces quadratic formula

$$\bigwedge_{0<i\leq k} \bigwedge_{0\leq j<i} (\overline{x}_i \neq \overline{x}_j)$$

- Using a bitonic sorting network (Kröning and Strichman [2003]) reduces the complexity to $O(k \log^2 k)$
- Lazy checking is more effective in practice (Sörensson's thesis)

Checking for Simple Paths

- Simple-minded check produces quadratic formula

$$\bigwedge_{0<i\leq k} \bigwedge_{0\leq j<i} (\overline{x}_i \neq \overline{x}_j)$$

- Using a bitonic sorting network (Kröning and Strichman [2003]) reduces the complexity to $O(k \log^2 k)$
- Lazy checking is more effective in practice (Sörensson's thesis)

k-Induction

- Sheeran *et al.* call their method k-induction
- If all states on length-k paths from the initial states satisfy p, and
- k consecutive states satisfying p are always followed by a state satisfying p, then
- all states reachable from the initial states satisfy p
- The second premise is verified when there are no simple paths of length $k + 1$

Abstraction Refinement

- Assume abstract model S_a and abstraction of property φ_a such that $S_a \models \varphi_a$ implies $S \models \varphi$
- Use complete method on abstract model S_a, but use BMC on the concrete model S when a counterexample is found in S_a
  - Use the counterexample(s) found in S_a to constrain search in S
  - If concretization fails, use UNSAT core to refine abstraction
  - One-to-one and one-to-many concretization possible
- It is possible to reverse the order: proof-based abstraction (Amla and McMillan [2004])
  - Use BMC and periodically extract abstract model from UNSAT core and check it with complete method

Abstraction Refinement

- Assume abstract model S_a and abstraction of property φ_a such that $S_a \models \varphi_a$ implies $S \models \varphi$
- Use complete method on abstract model S_a, but use BMC on the concrete model S when a counterexample is found in S_a
  - Use the counterexample(s) found in S_a to constrain search in S
  - If concretization fails, use UNSAT core to refine abstraction
  - One-to-one and one-to-many concretization possible
- It is possible to reverse the order: proof-based abstraction (Amla and McMillan [2004])
  - Use BMC and periodically extract abstract model from UNSAT core and check it with complete method

Abstraction Refinement

- Assume abstract model S_a and abstraction of property φ_a such that $S_a \models \varphi_a$ implies $S \models \varphi$
- Use complete method on abstract model S_a, but use BMC on the concrete model S when a counterexample is found in S_a
  - Use the counterexample(s) found in S_a to constrain search in S
  - If concretization fails, use UNSAT core to refine abstraction
  - One-to-one and one-to-many concretization possible
- It is possible to reverse the order: proof-based abstraction (Amla and McMillan [2004])
  - Use BMC and periodically extract abstract model from UNSAT core and check it with complete method

Interpolation (McMillan [2003])

- Suppose $I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1) \wedge T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$ is unsatisfiable
- Let $F_1 = I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ and $F_2 = T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$
- Then $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(\overline{x}_1)$ is such that
  - $F_1(\overline{x}_0, \overline{x}_1) \rightarrow \mathcal{I}_1(\overline{x}_1)$
  - $\mathcal{I}_1(\overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\mathcal{I}_1(\overline{x}_1)$ can be computed in linear time from a resolution proof that $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\exists \overline{x}_0 \,.\, I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ is the strongest interpolant
  - set of states reachable from $I(\overline{x}_0)$ in one step

Interpolation (McMillan [2003])

- Suppose $I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1) \wedge T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$ is unsatisfiable
- Let $F_1 = I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ and $F_2 = T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$
- Then $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(\overline{x}_1)$ is such that
  - $F_1(\overline{x}_0, \overline{x}_1) \rightarrow \mathcal{I}_1(\overline{x}_1)$
  - $\mathcal{I}_1(\overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\mathcal{I}_1(\overline{x}_1)$ can be computed in linear time from a resolution proof that $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\exists \overline{x}_0 \,.\, I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ is the strongest interpolant
  - set of states reachable from $I(\overline{x}_0)$ in one step

Interpolation (McMillan [2003])

- Suppose $I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1) \wedge T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$ is unsatisfiable
- Let $F_1 = I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ and $F_2 = T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$
- Then $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(\overline{x}_1)$ is such that
  - $F_1(\overline{x}_0, \overline{x}_1) \rightarrow \mathcal{I}_1(\overline{x}_1)$
  - $\mathcal{I}_1(\overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\mathcal{I}_1(\overline{x}_1)$ can be computed in linear time from a resolution proof that $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\exists \overline{x}_0 . I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ is the strongest interpolant
  - set of states reachable from $I(\overline{x}_0)$ in one step

Interpolation (McMillan [2003])

- Suppose $I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1) \wedge T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$ is unsatisfiable
- Let $F_1 = I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ and $F_2 = T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$
- Then $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(\overline{x}_1)$ is such that
  - $F_1(\overline{x}_0, \overline{x}_1) \rightarrow \mathcal{I}_1(\overline{x}_1)$
  - $\mathcal{I}_1(\overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\mathcal{I}_1(\overline{x}_1)$ can be computed in linear time from a resolution proof that $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\exists \overline{x}_0 \,.\, I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ is the strongest interpolant
  - set of states reachable from $I(\overline{x}_0)$ in one step

Interpolation (McMillan [2003])

- Suppose $I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1) \wedge T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$ is unsatisfiable
- Let $F_1 = I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ and $F_2 = T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$
- Then $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(\overline{x}_1)$ is such that
  - $F_1(\overline{x}_0, \overline{x}_1) \rightarrow \mathcal{I}_1(\overline{x}_1)$
  - $\mathcal{I}_1(\overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\mathcal{I}_1(\overline{x}_1)$ can be computed in linear time from a resolution proof that $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\exists \overline{x}_0 \,.\, I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ is the strongest interpolant
  - set of states reachable from $I(\overline{x}_0)$ in one step

Interpolation (McMillan [2003])

- Suppose $I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1) \wedge T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$ is unsatisfiable
- Let $F_1 = I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ and $F_2 = T(\overline{x}_1, \overline{x}_2) \wedge \cdots \wedge T(\overline{x}_{k-1}, \overline{x}_k) \wedge \neg P(\overline{x}_k)$
- Then $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(\overline{x}_1)$ is such that
  - $F_1(\overline{x}_0, \overline{x}_1) \rightarrow \mathcal{I}_1(\overline{x}_1)$
  - $\mathcal{I}_1(\overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\mathcal{I}_1(\overline{x}_1)$ can be computed in linear time from a resolution proof that $F_1(\overline{x}_0, \overline{x}_1) \wedge F_2(\overline{x}_1, \ldots, \overline{x}_k)$ is unsatisfiable
- $\exists \overline{x}_0 \,.\, I(\overline{x}_0) \wedge T(\overline{x}_0, \overline{x}_1)$ is the strongest interpolant
  - set of states reachable from $I(\overline{x}_0)$ in one step

Interpolation-Based Termination Check

- $\mathcal{I}_1(\overline{x}_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(\overline{x}_0)$ with $I(\overline{x}_0) \vee \mathcal{I}_1(\overline{x}_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(\overline{x}_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable
- At convergence, an inductive invariant is obtained
- Convergence guaranteed when the backward recursive radius is reached

Interpolation-Based Termination Check

- $\mathcal{I}_1(\overline{x}_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(\overline{x}_0)$ with $I(\overline{x}_0) \vee \mathcal{I}_1(\overline{x}_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(\overline{x}_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable
- At convergence, an inductive invariant is obtained
- Convergence guaranteed when the backward recursive radius is reached

Interpolation-Based Termination Check

- $\mathcal{I}_1(\overline{x}_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(\overline{x}_0)$ with $I(\overline{x}_0) \vee \mathcal{I}_1(\overline{x}_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(\overline{x}_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable
- At convergence, an inductive invariant is obtained
- Convergence guaranteed when the backward recursive radius is reached

Interpolation-Based Termination Check

- $\mathcal{I}_1(\overline{x}_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(\overline{x}_0)$ with $I(\overline{x}_0) \vee \mathcal{I}_1(\overline{x}_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(\overline{x}_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable
- At convergence, an inductive invariant is obtained
- Convergence guaranteed when the backward recursive radius is reached

Interpolation-Based Termination Check

- $\mathcal{I}_1(\overline{x}_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(\overline{x}_0)$ with $I(\overline{x}_0) \vee \mathcal{I}_1(\overline{x}_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(\overline{x}_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable
- At convergence, an inductive invariant is obtained
- Convergence guaranteed when the backward recursive radius is reached

Preimage Computation by Solution Enumeration

- Let $\text{Pre}(Q(\overline{x}))$ be the predicate describing the states that are predecessors of the states described by Q
- Repeated application of Pre from $\neg P$ corresponds to backward breadth-first search from the error states
  - It computes an inductive strengthening of the property
- Common approach with BDDs
- Can be adapted to CNF (McMillan [2002])
  - Introduced the use of blocking clauses

Preimage Computation by Solution Enumeration

- Let $\mathrm{Pre}(Q(\overline{x}))$ be the predicate describing the states that are predecessors of the states described by Q
- Repeated application of Pre from $\neg P$ corresponds to backward breadth-first search from the error states
  - It computes an inductive strengthening of the property
- Common approach with BDDs
- Can be adapted to CNF (McMillan [2002])
  - Introduced the use of blocking clauses

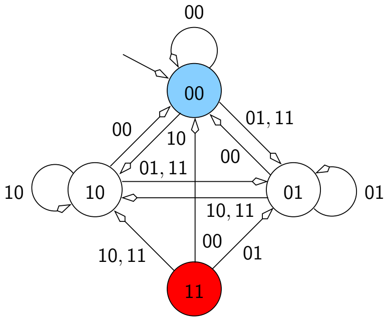
Preimage Computation by Solution Enumeration

- Let $\text{Pre}(Q(\overline{x}))$ be the predicate describing the states that are predecessors of the states described by Q
- Repeated application of Pre from $\neg P$ corresponds to backward breadth-first search from the error states
  - It computes an inductive strengthening of the property
- Common approach with BDDs
- Can be adapted to CNF (McMillan [2002])
  - Introduced the use of blocking clauses

Preimage Computation by Solution Enumeration

- Let $\mathrm{Pre}(Q(\overline{x}))$ be the predicate describing the states that are predecessors of the states described by Q
- Repeated application of Pre from $\neg P$ corresponds to backward breadth-first search from the error states
  - It computes an inductive strengthening of the property
- Common approach with BDDs
- Can be adapted to CNF (McMillan [2002])
  - Introduced the use of blocking clauses

Back to The Simple Arbiter

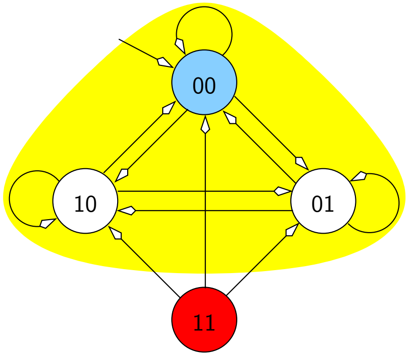


$$I(\overline{g}) = \neg g_1 \wedge \neg g_2$$
$$\exists r_1, r_2 \,.\, T(\overline{r}, \overline{g}, \overline{g}') = \neg g_1' \vee \neg g_2'$$
$$P(\overline{g}) = \neg g_1 \vee \neg g_2$$

Inductive Proofs for Transition Systems

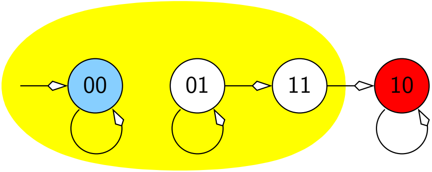
- Prove initiation (base case)
  - $I(\overline{x}) \Rightarrow P(\overline{x})$
  - All initial states satisfy P
  - $(\neg g_1 \wedge \neg g_2) \Rightarrow (\neg g_1 \vee \neg g_2)$
- Prove consecution (inductive step)
  - $P(\overline{x}) \wedge T(\bar{i}, \overline{x}, \overline{x}') \Rightarrow P(\overline{x}')$
  - All successors of states satisfying P satisfy P
  - $(\neg g_1 \vee \neg g_2) \wedge (\neg g_1' \vee \neg g_2') \Rightarrow (\neg g_1' \vee \neg g_2')$
- If both pass, all reachable states satisfy the property
  - $S \models P$

Visualizing Inductive Proofs

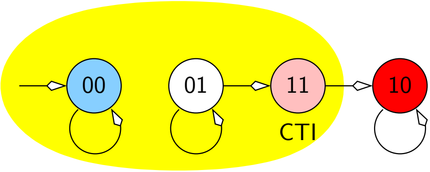


The inductive assertion (yellow) contains all initial (blue) states and no arrow leaves it (it is closed under the transition relation)

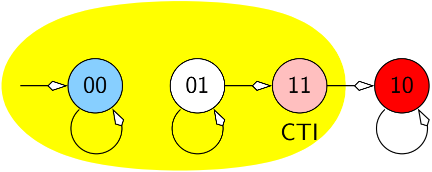
Counterexamples to Induction: The Troublemakers



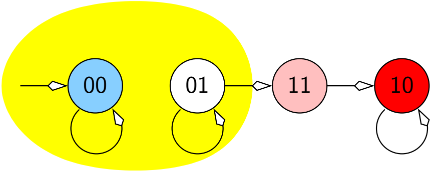
Counterexamples to Induction: The Troublemakers



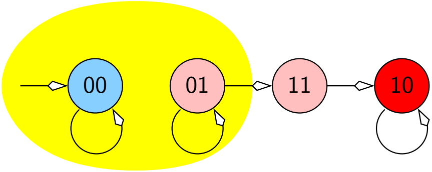
Invariant Strengthening



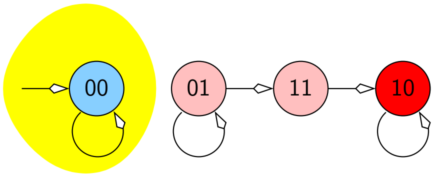
Invariant Strengthening



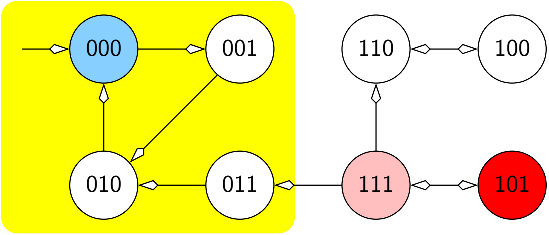
Invariant Strengthening



Invariant Strengthening



Strong and Weak Invariants



Induction is not restricted to:

- the strongest inductive invariant (forward-reachable states)
- . . . or the weakest inductive invariant (complement of the backward-reachable states)
- $\neg x_1$ is simpler than $\neg x_1 \wedge (\neg x_2 \vee \neg x_3)$ (strongest) and $(\neg x_1 \vee \neg x_3)$ (weakest)

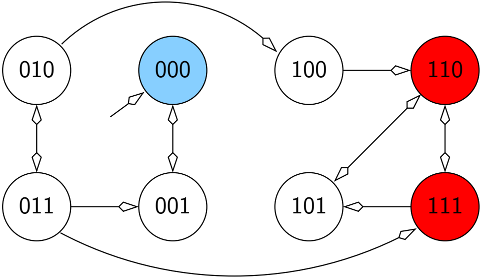
Completeness for Finite-State Systems

- CTIs are effectively bad states
  - If a CTI is reachable so is at least one bad state
- Remove CTI from P and try again
- Eventually either:
  - An inductive strengthening of P results
  - An initial state is removed from P
- In the latter case, a counterexample is obtained

Examples of Strengthening Strategies

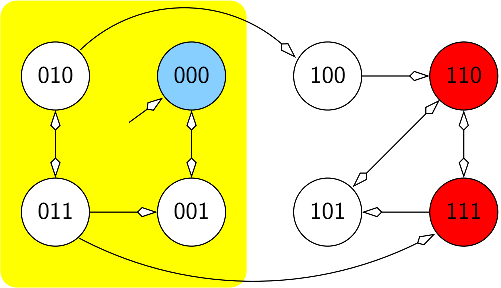
- Removing one CTI at a time is very inefficient!
  - Several strategies in use to avoid that
- Fixpoint-based invariant checking: if $\nu Z \, . \, p \wedge \mathrm{AX}\, Z$ converges in $n > 0$ iterations, then $\bigwedge_{0 \leq i < n} \mathrm{AX}^i\, p$ is an inductive invariant
  - In fact, the weakest inductive invariant
- k-induction: if all states on length-k paths from the initial states satisfy p, and k distinct consecutive states satisfying p are always followed by a state satisfying p, then all states reachable from the initial states satisfy p.
- Interpolation-based model checking: the converged interpolant is an inductive invariant
- fsis algorithm (Bradley and Manna [2007]): try to extract an inductive clause from CTI to exclude multiple CTIs

Relative Induction



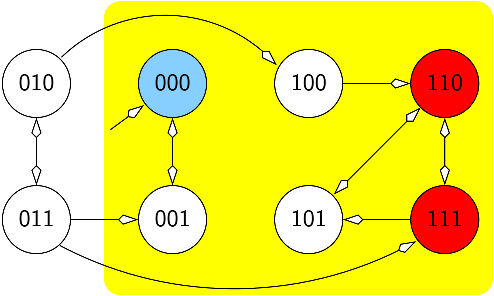
$$\varphi = \neg x_1 \wedge (x_1 \vee \neg x_2)$$

Relative Induction



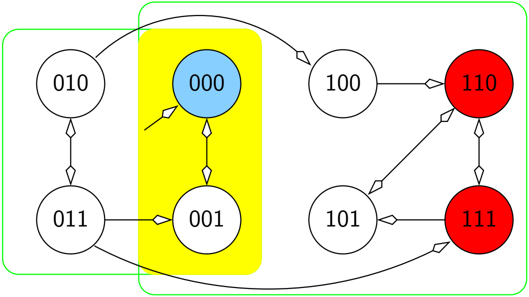
$\neg x_1$ is not inductive

Relative Induction



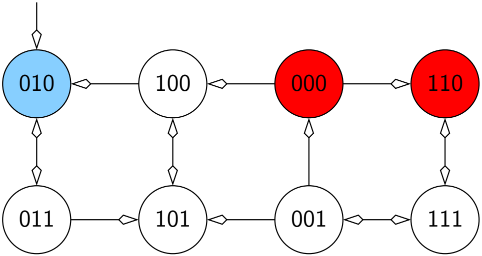
$x_1 \vee \neg x_2$ is inductive

Relative Induction



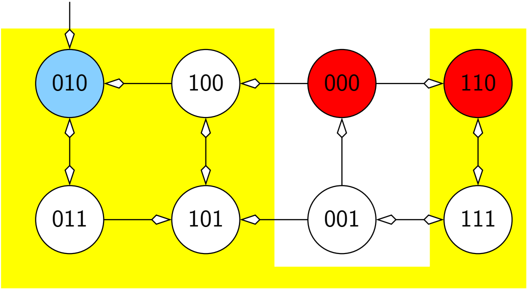
$\neg x_1$ is inductive relative to $x_1 \vee \neg x_2$

Shortcoming of Relative Induction



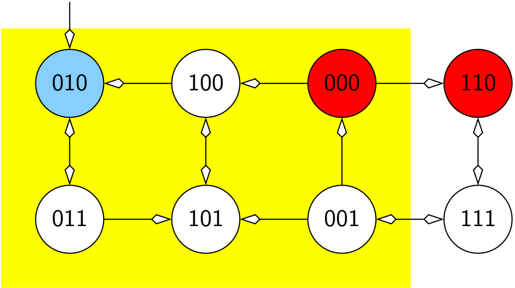
$$P = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$
$$\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$

Shortcoming of Relative Induction



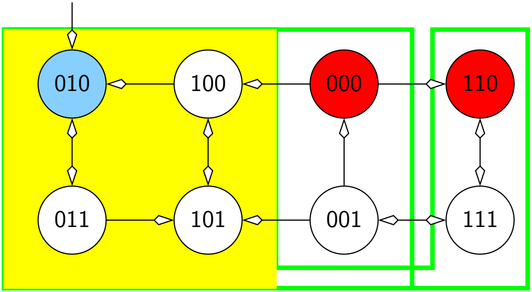
$$(x_1 \vee x_2) \wedge P \wedge T \not\Rightarrow (x_1' \vee x_2')$$

Shortcoming of Relative Induction



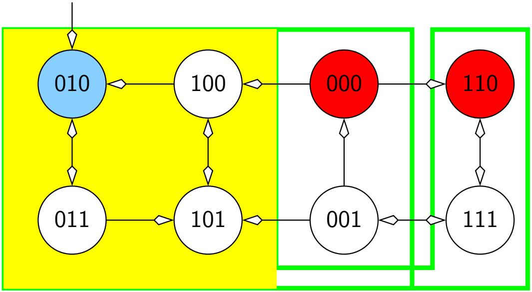
$$(\neg x_1 \vee \neg x_2) \wedge P \wedge T \not\Rightarrow (\neg x_1' \vee \neg x_2')$$

Shortcoming of Relative Induction



$$(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge P \wedge T \Rightarrow (x_1' \vee x_2') \wedge (\neg x_1' \vee \neg x_2')$$

Shortcoming of Relative Induction



$(x_1 \vee x_2)$ and $(\neg x_1 \vee \neg x_2)$ are mutually inductive

IC3: Basic Algorithm

IC3 (Bradley [2011]) stands for

1. Incremental Construction of
2. Inductive Clauses for
3. Indubitable Correctness

IC3 is an Incremental Inductive Verification (IIV) algorithm

Basic Tenets

- Approximate reachability assumptions
  - F_i: contains at least all the states reachable in i steps or less
  - If $S \models P$, F_i eventually becomes inductive for some i
  - Approximation is desirable: IC3 does not attempt to get the most precise F_i's
- Stepwise relative induction
  - Learn useful facts via induction relative to reachability assumptions
- Clausal representation
  - Learn clauses (lemmas) from CTIs
  - A form of abstract interpretation

IC3 Invariants

- The four main invariants of IC3:

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \le i < k \\
F_i &\Rightarrow P & & 0 \le i \le k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \le i < k
\end{aligned}
$$

- Established if there are no counterexamples of length 0 or 1
- The implicit invariant of the outer loop: no counterexamples of length k or less

Reasonable Invariants

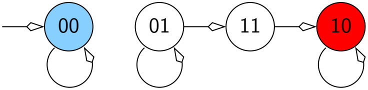
- $I \Rightarrow F_0$: F_0 overapproximates the initial condition. (In practice, $I = F_0$.)
- $F_i \Rightarrow F_{i+1}$: a state believed to be reachable in i steps or less is also believed to be reachable in $i+1$ steps or less
- $F_i \Rightarrow P$: no state believed to be reachable in i steps or less violates P
- $F_i \wedge T \Rightarrow F'_{i+1}$: all the immediate successors of a state believed to be reachable in i steps or less are believed to be reachable in $i+1$ steps or less

Pseudo-Pseudocode

```
bool IC3 {
    if (I ⇏ P or I ∧ T ⇏ P′)
        return ⊥
    F_0 = I; F_1 = P; k = 1
    repeat {
        while (there are CTIs in F_k) {
            either find a counterexample and return ⊥
            or refine F_1, . . . , F_k
        }
        k ++
        set F_k = P and propagate clauses
        if (F_i = F_{i+1} for some 0 < i < k)
            return ⊤
    }
}
```

Example: Passing Property

No counterexamples of length 0 or 1

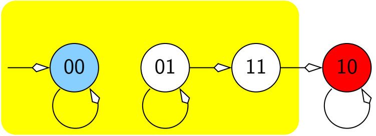


$$I = \neg x_1 \wedge \neg x_2$$
$$P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \le i < k \\ F_i &\Rightarrow P & & 0 \le i \le k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \le i < k \end{aligned}$$

Example: Passing Property

Does $F_1 \wedge T \Rightarrow P'$?

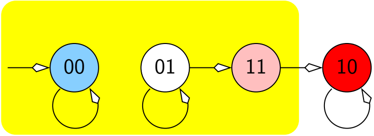


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = P = \neg x_1 \vee x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Passing Property

Found CTI $s = x_1 \wedge x_2$

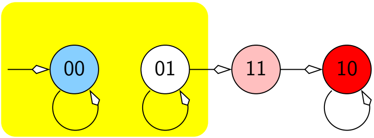


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = P = \neg x_1 \vee x_2$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\
F_i &\Rightarrow P & & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k
\end{aligned}
$$

Example: Passing Property

Is $\neg s = \neg x_1 \vee \neg x_2$ inductive relative to F_1?

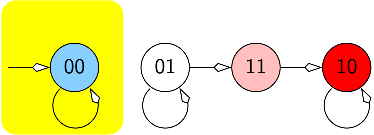


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 && \\ F_i &\Rightarrow F_{i+1} && 0 \leq i < k \\ F_i &\Rightarrow P && 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} && 0 \leq i < k \end{aligned}$$

Example: Passing Property

No. Is $\neg s = \neg x_1 \vee \neg x_2$ inductive relative to F_0?

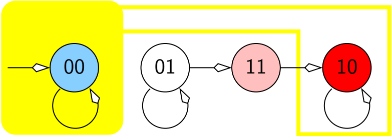


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

Yes. Generalize $\neg s$ at level 0 in one of the two possible ways: either $\neg x_1$ or $\neg x_2$

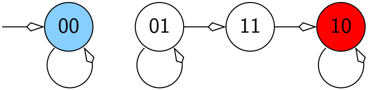


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & \\ F_i &\Rightarrow F_{i+1} & 0 \le i < k \\ F_i &\Rightarrow P & 0 \le i \le k \\ F_i \wedge T &\Rightarrow F'_{i+1} & 0 \le i < k \end{aligned}$$

Example: Passing Property

Update F_1

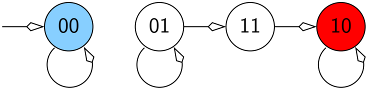


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

No more CTIs in F_1. No counterexamples of length 2. Instantiate F_2

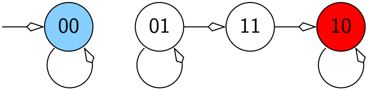


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_2$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

Propagate clauses from F_1 to F_2

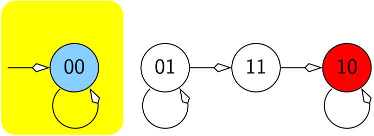


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_2$$
$$F_2 = (\neg x_1 \vee x_2) \wedge \neg x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

F_1 and F_2 are identical. Property proved

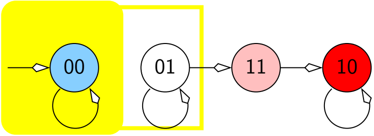


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_2$$
$$F_2 = (\neg x_1 \vee x_2) \wedge \neg x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

What happens if we generalize $\neg s = \neg x_1 \vee \neg x_2$ at level 0 in the other way $(\neg x_1)$?

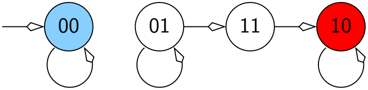


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1 \vee x_2$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\
F_i &\Rightarrow P & & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k
\end{aligned}
$$

Example: Passing Property

Update F_1

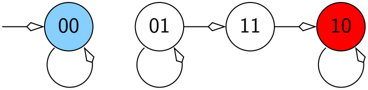


$$F_0 = I = \neg x_1 \land \neg x_2$$
$$F_1 = (\neg x_1 \lor x_2) \land \neg x_1$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \land T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

No more CTIs in F_1. No counterexamples of length 2. Instantiate F_2

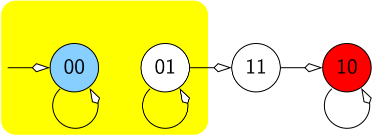


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Passing Property

No clauses propagate from F_1 to F_2

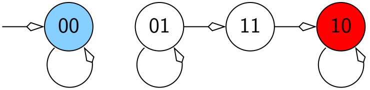


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \le i < k \\ F_i &\Rightarrow P & & 0 \le i \le k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \le i < k \end{aligned}$$

Example: Passing Property

Remove subsumed clauses

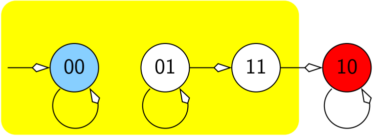


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Passing Property

Does $F_2 \wedge T \Rightarrow P'$?

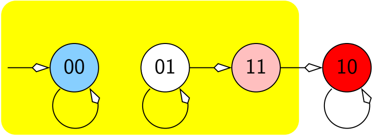


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$
\begin{array}{rcll}
I & \Rightarrow & F_0 & \\
F_i & \Rightarrow & F_{i+1} & 0 \leq i < k \\
F_i & \Rightarrow & P & 0 \leq i \leq k \\
F_i \wedge T & \Rightarrow & F'_{i+1} & 0 \leq i < k
\end{array}
$$

Example: Passing Property

Found CTI $s = x_1 \wedge x_2$ (same as before)

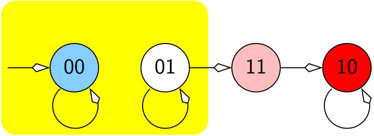


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\
F_i &\Rightarrow P & & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k
\end{aligned}
$$

Example: Passing Property

Is $\neg s = \neg x_1 \vee \neg x_2$ inductive relative to F_1?

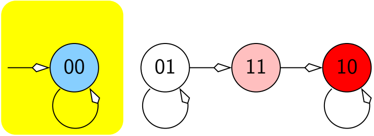


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & \\ F_i &\Rightarrow F_{i+1} & 0 \leq i < k \\ F_i &\Rightarrow P & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & 0 \leq i < k \end{aligned}$$

Example: Passing Property

No. We know it is inductive at level 0.

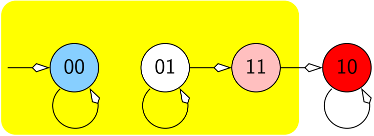


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Passing Property

If generalization produces $\neg x_1$ again, the CTI is not eliminated

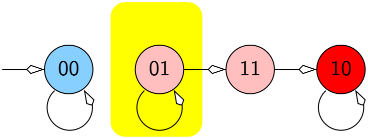


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\
F_i &\Rightarrow P & & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k
\end{aligned}
$$

Example: Passing Property

Find predecessor t of CTI $x_1 \wedge x_2$ in $F_1 \setminus F_0$

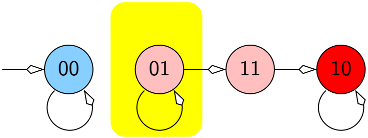


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Passing Property

Found $t = \neg x_1 \wedge x_2$

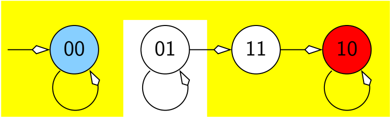


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

The clause $\neg t = x_1 \vee \neg x_2$ is inductive at all levels

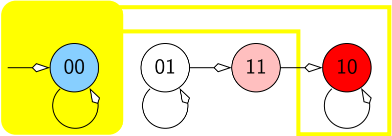


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Passing Property

Generalization of $\neg t = x_1 \vee \neg x_2$ produces $\neg x_2$

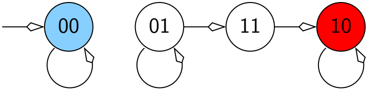


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1$$
$$F_2 = P = \neg x_1 \vee x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \le i < k$$
$$F_i \Rightarrow P \qquad 0 \le i \le k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \le i < k$$

Example: Passing Property

Update F_1 and F_2

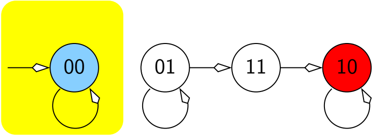


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1 \wedge \neg x_2$$
$$F_2 = (\neg x_1 \vee x_2) \wedge \neg x_2$$

$$\begin{aligned} I &\Rightarrow F_0 & & \\ F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\ F_i &\Rightarrow P & & 0 \leq i \leq k \\ F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k \end{aligned}$$

Example: Passing Property

F_1 and F_2 are equivalent. Property (almost) proved

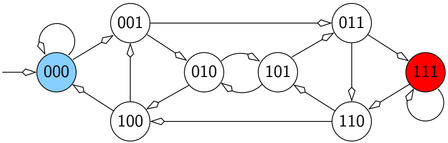


$$F_0 = I = \neg x_1 \wedge \neg x_2$$
$$F_1 = \neg x_1 \wedge \neg x_2$$
$$F_2 = (\neg x_1 \vee x_2) \wedge \neg x_2$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Failing Property

No counterexamples of length 0 or 1

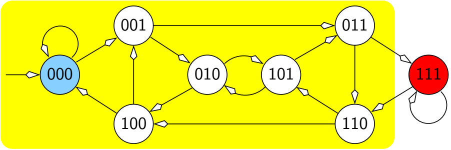


$$I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \le i < k \\
F_i &\Rightarrow P & & 0 \le i \le k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \le i < k
\end{aligned}
$$

Example: Failing Property

Does $F_1 \wedge T \Rightarrow P'$?

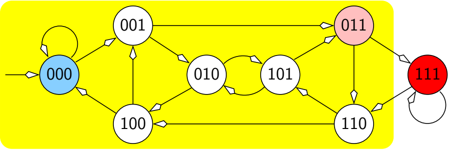


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\
F_i &\Rightarrow P & & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k
\end{aligned}
$$

Example: Failing Property

Found CTI $s = \neg x_1 \wedge x_2 \wedge x_3$

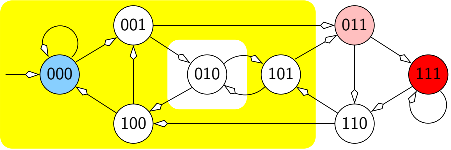


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\
F_i &\Rightarrow P & & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k
\end{aligned}
$$

Example: Failing Property

The clause $\neg s = x_1 \vee \neg x_2 \vee \neg x_3$ generalizes to $\neg x_2$ at level 0



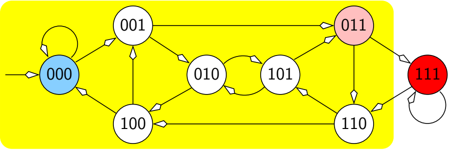
$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$

$F_1 = (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_2$

$$
\begin{array}{rcll}
I & \Rightarrow & F_0 & \\
F_i & \Rightarrow & F_{i+1} & 0 \leq i < k \\
F_i & \Rightarrow & P & 0 \leq i \leq k \\
F_i \wedge T & \Rightarrow & F'_{i+1} & 0 \leq i < k
\end{array}
$$

Example: Failing Property

No CTI left: no counterexample of length 2. F_2 instantiated, but no clause propagated

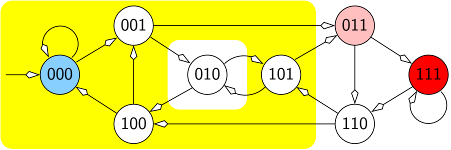


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = \neg x_2$$
$$F_2 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \le i < k \\
F_i &\Rightarrow P & & 0 \le i \le k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \le i < k
\end{aligned}
$$

Example: Failing Property

The clause $\neg s = x_1 \vee \neg x_2 \vee \neg x_3$ generalizes again to $\neg x_2$ at level 0

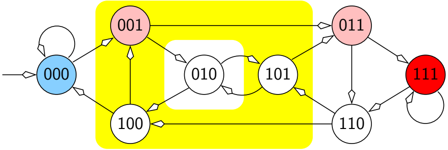


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = \neg x_2$$
$$F_2 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{array}{rcll}
I & \Rightarrow & F_0 & \\
F_i & \Rightarrow & F_{i+1} & 0 \leq i < k \\
F_i & \Rightarrow & P & 0 \leq i \leq k \\
F_i \wedge T & \Rightarrow & F'_{i+1} & 0 \leq i < k
\end{array}
$$

Example: Failing Property

Suppose IC3 recurs on $t = \neg x_1 \wedge \neg x_2 \wedge x_3$ in $F_1 \setminus F_0$

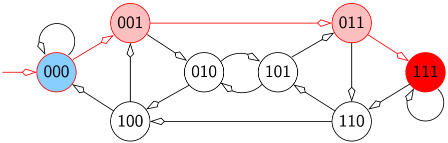


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = \neg x_2$$
$$F_2 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1} \qquad 0 \leq i < k$$
$$F_i \Rightarrow P \qquad 0 \leq i \leq k$$
$$F_i \wedge T \Rightarrow F'_{i+1} \qquad 0 \leq i < k$$

Example: Failing Property

Clause $\neg t = x_1 \vee x_2 \vee \neg x_3$ is not inductive at level 0: the property fails

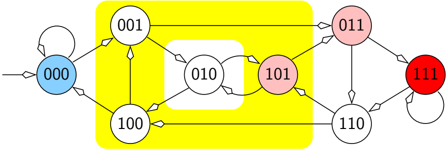


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = \neg x_2$$
$$F_2 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{aligned}
I &\Rightarrow F_0 && \\
F_i &\Rightarrow F_{i+1} && 0 \leq i < k \\
F_i &\Rightarrow P && 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} && 0 \leq i < k
\end{aligned}
$$

Example: Failing Property

Suppose now IC3 recurs on $t = x_1 \wedge \neg x_2 \wedge x_3$ in $F_1 \setminus F_0$

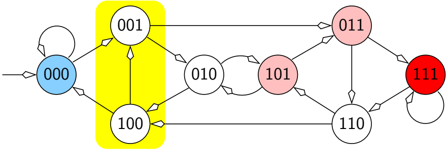


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = \neg x_2$$
$$F_2 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \le i < k \\
F_i &\Rightarrow P & & 0 \le i \le k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \le i < k
\end{aligned}
$$

Example: Failing Property

Clause $\neg t = \neg x_1 \vee x_2 \vee \neg x_3$ is inductive at level 1

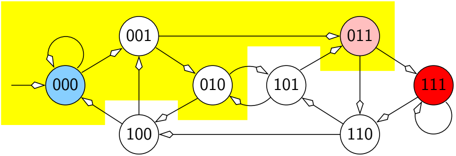


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = \neg x_2$$
$$F_2 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \leq i < k \\
F_i &\Rightarrow P & & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \leq i < k
\end{aligned}
$$

Example: Failing Property

Generalization of $\neg t$ adds $\neg x_1$ to F_1 and F_2

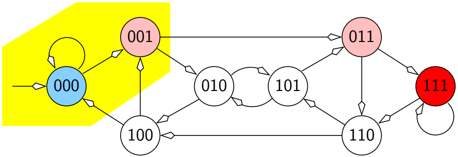


$$F_0 = I = \neg x_1 \wedge \neg x_3 \wedge \neg x_3$$
$$F_1 = \neg x_2 \wedge \neg x_1$$
$$F_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_1$$

$$
\begin{aligned}
I &\Rightarrow F_0 & \\
F_i &\Rightarrow F_{i+1} & 0 \le i < k \\
F_i &\Rightarrow P & 0 \le i \le k \\
F_i \wedge T &\Rightarrow F'_{i+1} & 0 \le i < k
\end{aligned}
$$

Example: Failing Property

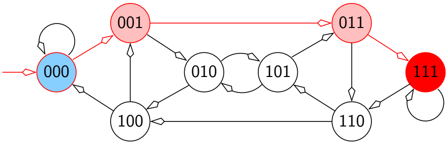
Only $t = \neg x_1 \wedge \neg x_2 \wedge x_3$ remains in $F_1 \setminus F_0$



$$
\begin{aligned}
I &\Rightarrow F_0 & \\
F_i &\Rightarrow F_{i+1} & 0 \leq i < k \\
F_i &\Rightarrow P & 0 \leq i \leq k \\
F_i \wedge T &\Rightarrow F'_{i+1} & 0 \leq i < k
\end{aligned}
$$

Example: Failing Property

The same counterexample as before is found



$$
\begin{aligned}
I &\Rightarrow F_0 & & \\
F_i &\Rightarrow F_{i+1} & & 0 \le i < k \\
F_i &\Rightarrow P & & 0 \le i \le k \\
F_i \wedge T &\Rightarrow F'_{i+1} & & 0 \le i < k
\end{aligned}
$$

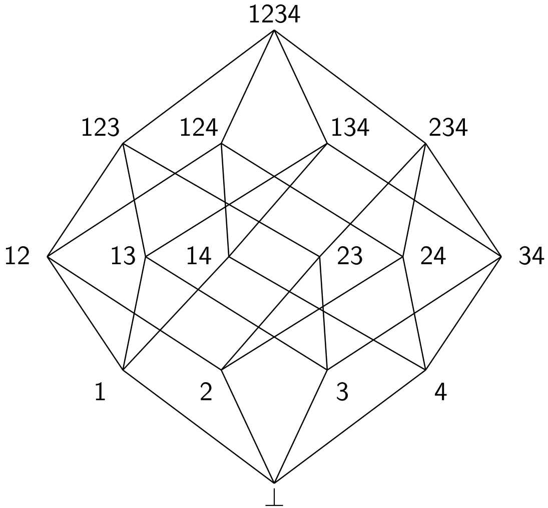
Clause Generalization

- A CTI is a cube (conjunction of literals)
  - e.g., $s = x_1 \wedge \neg x_2 \wedge x_3$
- The negation of a CTI is a clause
  - e.g., $\neg s = \neg x_1 \vee x_2 \vee \neg x_3$
- Conjoining $\neg s$ to a reachability assumption F_i excludes the CTI from it
- Generalization extracts a subclause from $\neg s$ that excludes more states that are "like the CTI"
  - e.g., $\neg x_3$ may be a subclause of $\neg s$ that excludes states that, like the CTI, are not reachable in i steps
  - Every literal dropped doubles the number of states excluded by a clause
  - Generalization is time-consuming, but critical to performance

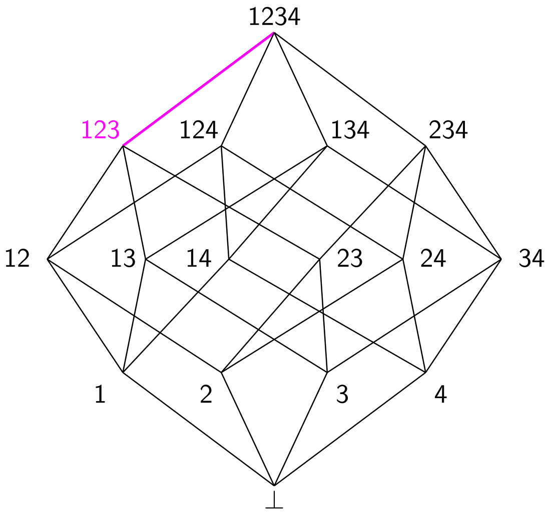
Generalization

- Crucial for efficiency
- Generalization in IC3 produces a minimal inductive clause (MIC)
- The MIC algorithm is based on DOWN and UP.
- DOWN extracts the (unique) maximal subclause
- UP finds a small, but not necessarily minimal subclause
- MIC recurs on subclauses of the result of UP

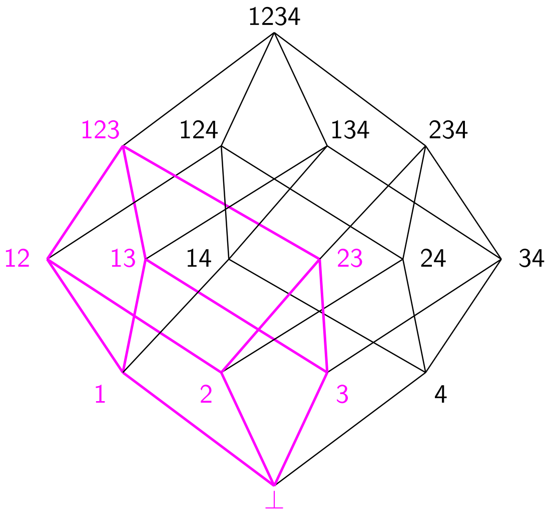
Minimal Inductive Clause



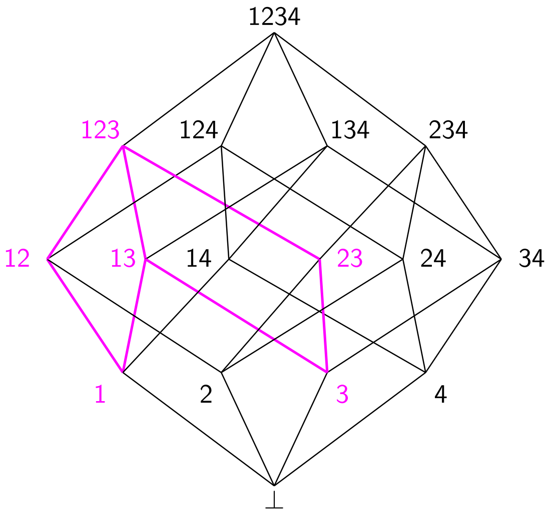
Minimal Inductive Clause



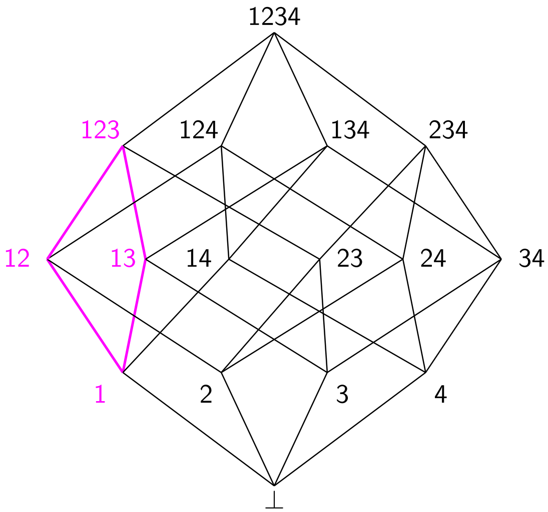
Minimal Inductive Clause



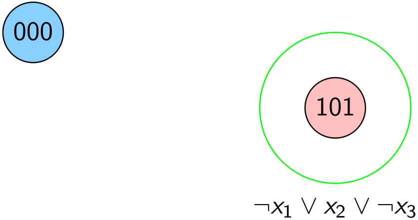
Minimal Inductive Clause



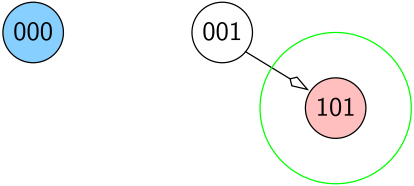
Minimal Inductive Clause



Maximal Inductive Subclause (DOWN)

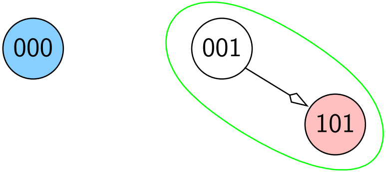


Maximal Inductive Subclause (DOWN)



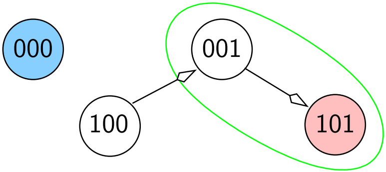
$\neg x_1 \vee x_2 \vee \neg x_3$

Maximal Inductive Subclause (DOWN)



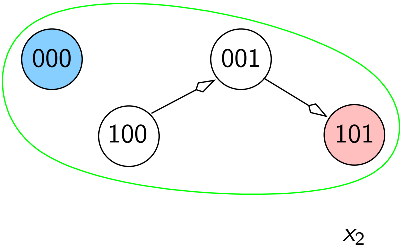
$x_2 \vee \neg x_3$

Maximal Inductive Subclause (DOWN)



$x_2 \vee \neg x_3$

Maximal Inductive Subclause (DOWN)



Use of UNSAT Cores

- $\neg s \wedge F_i \wedge T \Rightarrow \neg s'$ if and only if $\neg s \wedge F_i \wedge T \wedge s'$ is unsatisfiable
- The literals of s' are (unit) clauses in the SAT query
- If the implication holds, the SAT solver returns an unsatisfiable core
- Any literal of s' not in the core can be removed from s' because it does not contribute to the implication ...
- and from $\neg s$ because strengthening the antecedent preserves the implication

Use of UNSAT Core Example

- $\neg s \wedge F_0 \wedge T \Rightarrow \neg s'$ with

$$\begin{aligned}
\neg s &= \neg x_1 \vee \neg x_2 \\
F_0 &= \neg x_1 \wedge \neg x_2 \\
T &= (\neg x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge \neg x_2') \vee \cdots
\end{aligned}$$

- The SAT query, after some simplification, is

$$\neg x_1 \wedge \neg x_2 \wedge \neg x_1' \wedge \neg x_2' \wedge x_1' \wedge x_2'$$

- Two UNSAT cores are

$$\neg x_1' \wedge x_1'$$
$$\neg x_2' \wedge x_2'$$

from which the two generalizations we saw before follow

Clause Clean-Up

- As IC3 proceeds, clauses may be added to some F_i that subsume other clauses
- The weaker, subsumed clauses no longer contribute to the definition of F_i
- However, a weaker clause may propagate to F_{i+1} when the stronger clause does not
- Weak clauses are eliminated by subsumption only between major iterations and after propagation

More Efficiency-Related Issues

- State encoding determines what clauses are derived
- Incremental vs. monolithic
  - Reachability assumptions carry global information
  - . . . but are built incrementally
- Semantic vs. syntactic approach
  - Generalization "jumps over large distances"
- Long counterexamples at low k
  - Typically more efficient than increasing k
- Consequences of no unrolling
  - Many cheap (incremental) SAT calls
- Ability to parallelize
  - Clauses are easy to exchange

Outline

1. A Short Intro to Model Checking
   - Structures
   - Properties
2. SAT Solver Interface
   - To The Solver
   - From The Solver
3. Checking Invariants
   - Bounded Model Checking
   - Interpolation
   - Proving Invariants by Induction
   - IC3: Incremental Inductive Verification
4. Progress Properties and Branching Time
   - Bounded Model Checking
   - Incremental Inductive Verification (FAIR and *k*-Liveness)
   - Model Checking CTL

BMC: Translation from LTL

- Various techniques have been devised to translate an LTL formula φ into a propositional formula that expresses the constraints on a path that is a model of $\neg\varphi$. For instance:

$$[\![\neg\, \mathsf{F}\, \mathsf{G}\, \neg p]\!] = \bigvee_{0 \leq l \leq k} \left(T(\overline{x}_k, \overline{x}_l) \wedge \bigvee_{l \leq i \leq k} p(\overline{x}_i) \right)$$

- k-induction can be extended to provide a termination criterion

BMC: Translation from LTL

- Various techniques have been devised to translate an LTL formula φ into a propositional formula that expresses the constraints on a path that is a model of $\neg\varphi$. For instance:

$$[\![\neg \mathsf{F}\,\mathsf{G}\,\neg p]\!] = \bigvee_{0 \le l \le k} \left(T(\overline{x}_k, \overline{x}_l) \wedge \bigvee_{l \le i \le k} p(\overline{x}_i) \right)$$

- k-induction can be extended to provide a termination criterion

BMC: Liveness to Safety

- Checking progress properties requires cycle detection
- Augment model with shadow register
- The augmented model can nondeterministically save a snapshot of the current state in the shadow register
- If a state is subsequently reached that is identical to the one saved, a cycle has been detected
- Constraints can be added to make sure the cycle is an accepting one
- With this transformation an invariant checker suffices for all LTL properties

BMC: Liveness to Safety

- Checking progress properties requires cycle detection
- Augment model with shadow register
- The augmented model can nondeterministically save a snapshot of the current state in the shadow register
- If a state is subsequently reached that is identical to the one saved, a cycle has been detected
- Constraints can be added to make sure the cycle is an accepting one
- With this transformation an invariant checker suffices for all LTL properties

BMC: Liveness to Safety

- Checking progress properties requires cycle detection
- Augment model with shadow register
- The augmented model can nondeterministically save a snapshot of the current state in the shadow register
- If a state is subsequently reached that is identical to the one saved, a cycle has been detected
- Constraints can be added to make sure the cycle is an accepting one
- With this transformation an invariant checker suffices for all LTL properties

BMC: Liveness to Safety

- Checking progress properties requires cycle detection
- Augment model with shadow register
- The augmented model can nondeterministically save a snapshot of the current state in the shadow register
- If a state is subsequently reached that is identical to the one saved, a cycle has been detected
- Constraints can be added to make sure the cycle is an accepting one
- With this transformation an invariant checker suffices for all LTL properties

BMC: Liveness to Safety

- Checking progress properties requires cycle detection
- Augment model with shadow register
- The augmented model can nondeterministically save a snapshot of the current state in the shadow register
- If a state is subsequently reached that is identical to the one saved, a cycle has been detected
- Constraints can be added to make sure the cycle is an accepting one
- With this transformation an invariant checker suffices for all LTL properties

FAIR: Finding Rechable Fair Cycles

- Check language nonemptiness of the composition of structure S and generalized Büchi automaton for $\neg\varphi$
- Generalized means that multiple acceptance conditions (aka fairness constraints may be given: each must be satisfied
- FAIR (Bradley et al. [2011]) looks for a reachable fair cycle
- The search for a cycle is decomposed into several reachability queries
  - Each reachability query is a call to IC3

FAIR: Finding Rechable Fair Cycles

- Check language nonemptiness of the composition of structure S and generalized Büchi automaton for $\neg\varphi$
- Generalized means that multiple acceptance conditions (aka fairness constraints may be given: each must be satisfied
- FAIR (Bradley et al. [2011]) looks for a reachable fair cycle
- The search for a cycle is decomposed into several reachability queries
  - Each reachability query is a call to IC3

FAIR: Finding Rechable Fair Cycles

- Check language nonemptiness of the composition of structure S and generalized Büchi automaton for $\neg\varphi$
- Generalized means that multiple acceptance conditions (aka fairness constraints may be given: each must be satisfied
- FAIR (Bradley et al. [2011]) looks for a reachable fair cycle
- The search for a cycle is decomposed into several reachability queries
  - Each reachability query is a call to IC3

FAIR: Finding Rechable Fair Cycles

- Check language nonemptiness of the composition of structure S and generalized Büchi automaton for $\neg\varphi$
- Generalized means that multiple acceptance conditions (aka fairness constraints may be given: each must be satisfied
- FAIR (Bradley et al. [2011]) looks for a reachable fair cycle
- The search for a cycle is decomposed into several reachability queries
  - Each reachability query is a call to IC3

Strongly Connected Components

- A counterexample to a progress property is a lasso-shaped path that satisfies fairness constraints
- A lasso's cycle is contained in a strongly connected component (SCC) of the state graph
- A nonempty set of states is SCC-closed if every SCC is either contained in it or disjoint from it
- A partition of the states into SCC-closed sets is a coarser partition than the SCC partition; hence, . . .
- Every cycle of a graph is contained in some SCC-closed set
- Maintain a partition of the states into SCC-closed set
  - Refine it until a reachable fair cycle is found or none is proved to exist

Strongly Connected Components

- A counterexample to a progress property is a lasso-shaped path that satisfies fairness constraints
- A lasso's cycle is contained in a strongly connected component (SCC) of the state graph
- A nonempty set of states is SCC-closed if every SCC is either contained in it or disjoint from it
- A partition of the states into SCC-closed sets is a coarser partition than the SCC partition; hence, . . .
- Every cycle of a graph is contained in some SCC-closed set
- Maintain a partition of the states into SCC-closed set
  - Refine it until a reachable fair cycle is found or none is proved to exist

Strongly Connected Components

- A counterexample to a progress property is a lasso-shaped path that satisfies fairness constraints
- A lasso's cycle is contained in a strongly connected component (SCC) of the state graph
- A nonempty set of states is SCC-closed if every SCC is either contained in it or disjoint from it
- A partition of the states into SCC-closed sets is a coarser partition than the SCC partition; hence, . . .
- Every cycle of a graph is contained in some SCC-closed set
- Maintain a partition of the states into SCC-closed set
  - Refine it until a reachable fair cycle is found or none is proved to exist

Strongly Connected Components

- A counterexample to a progress property is a lasso-shaped path that satisfies fairness constraints
- A lasso's cycle is contained in a strongly connected component (SCC) of the state graph
- A nonempty set of states is SCC-closed if every SCC is either contained in it or disjoint from it
- A partition of the states into SCC-closed sets is a coarser partition than the SCC partition; hence, . . .
- Every cycle of a graph is contained in some SCC-closed set
- Maintain a partition of the states into SCC-closed set
  - Refine it until a reachable fair cycle is found or none is proved to exist

Strongly Connected Components

- A counterexample to a progress property is a lasso-shaped path that satisfies fairness constraints
- A lasso's cycle is contained in a strongly connected component (SCC) of the state graph
- A nonempty set of states is SCC-closed if every SCC is either contained in it or disjoint from it
- A partition of the states into SCC-closed sets is a coarser partition than the SCC partition; hence, . . .
- Every cycle of a graph is contained in some SCC-closed set
- Maintain a partition of the states into SCC-closed set
  - Refine it until a reachable fair cycle is found or none is proved to exist

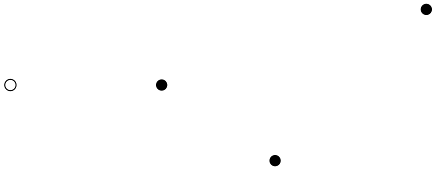
Strongly Connected Components

- A counterexample to a progress property is a lasso-shaped path that satisfies fairness constraints
- A lasso's cycle is contained in a strongly connected component (SCC) of the state graph
- A nonempty set of states is SCC-closed if every SCC is either contained in it or disjoint from it
- A partition of the states into SCC-closed sets is a coarser partition than the SCC partition; hence, . . .
- Every cycle of a graph is contained in some SCC-closed set
- Maintain a partition of the states into SCC-closed set
    - Refine it until a reachable fair cycle is found or none is proved to exist

FAIR: Finding Reachable Fair Cycles

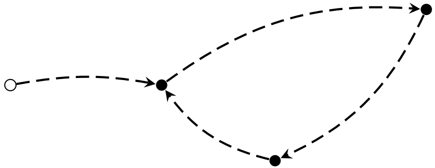
Reduce search for reachable fair cycle to a set of safety problems:

- Skeleton:



States of skeleton together satisfy all fairness constraints.

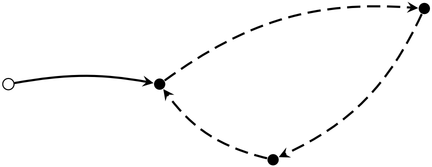
- Task: Connect states to form lasso.



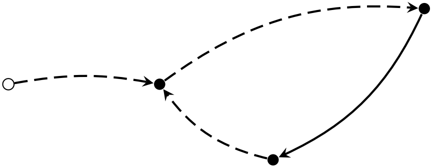
Reach Queries

Each connection task is a reach query.

- Stem query: Connect initial condition to a state:



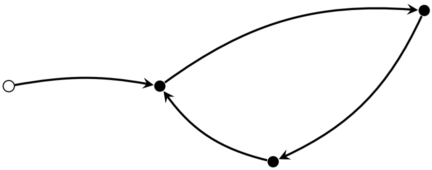
- Cycle query: Connect one state to another:



(To itself if skeleton has only one state.)

Witness to Nonemptiness

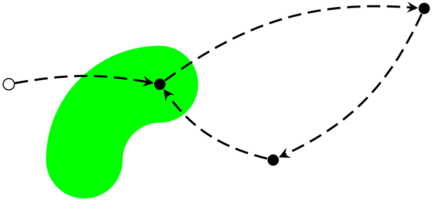
If all queries are answered positively:



Witness to nonemptiness of $\mathcal{C}$.

Global Reachability

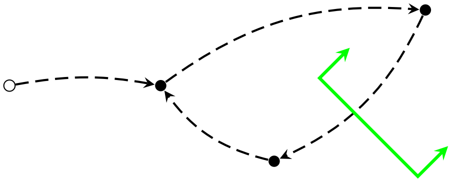
If a stem query is answered negatively: new inductive global reachability information.



- Constrains subsequent selection of skeletons.
- Constrains subsequent reach (stem and cycle) queries.
- Improve proof by strengthening (using ideas from IC3).

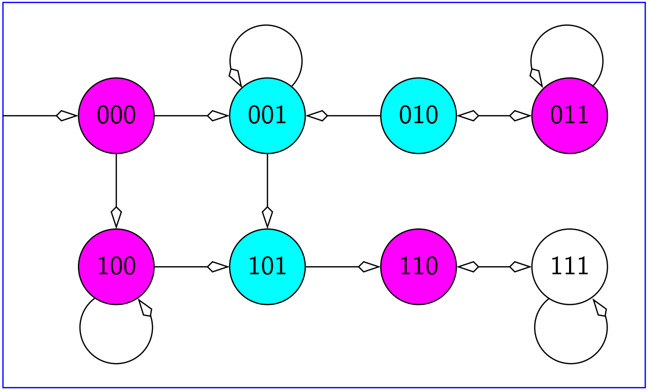
Barriers: Discovering SCC-Closed Sets

If a cycle query is answered negatively: new information about SCC structure of state graph.



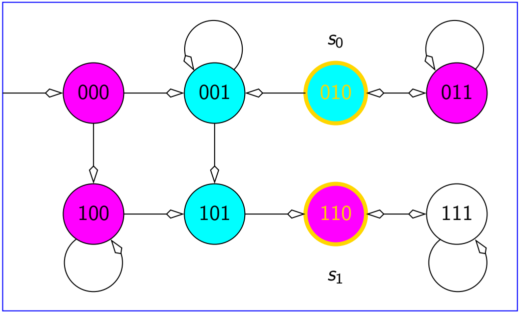
- Inductive proof: “one-way barrier”
- Each “side” of the proof is SCC-closed.
- Constrains subsequent selections of skeletons: all states on one side.

Example: Empty Language



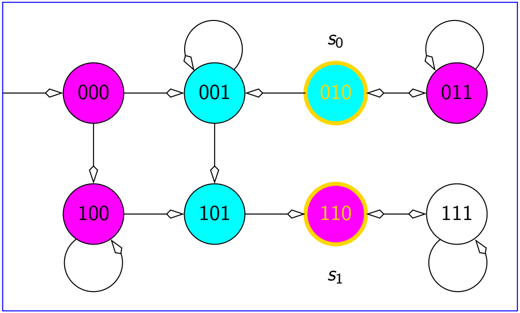
Example: Empty Language

	s_0	s_1
sk1	010	110



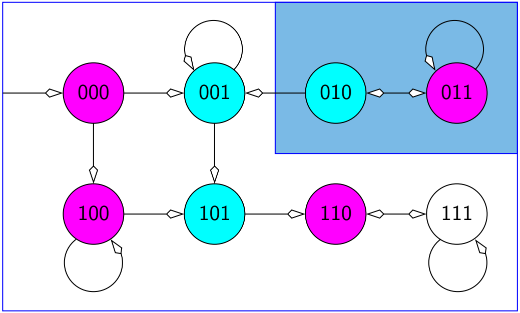
Example: Empty Language

	s_0	s_1
sk1	010	110



stem query produces $x_1 \vee \neg x_2$

Example: Empty Language

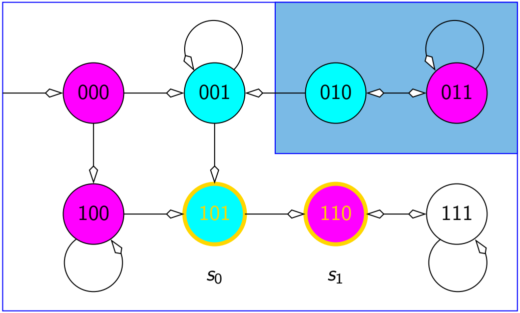


Example: Empty Language

	s_0	s_1
sk2	101	110

states satisfy

$x_1 \vee \neg x_2$

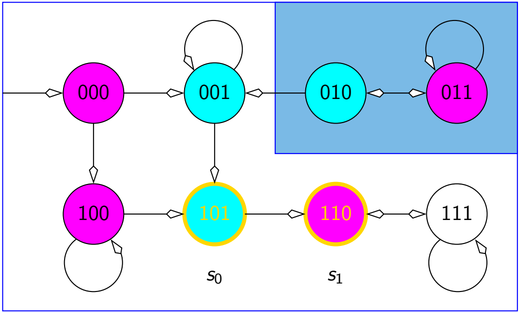


Example: Empty Language

	s_0	s_1
sk2	101	110

states satisfy

$x_1 \vee \neg x_2$



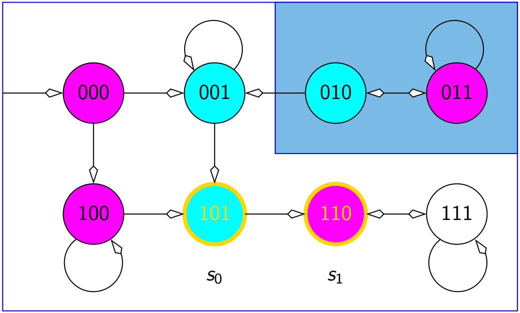
stem query passes

Example: Empty Language

	s_0	s_1
sk2	101	110

states satisfy

$x_1 \vee \neg x_2$



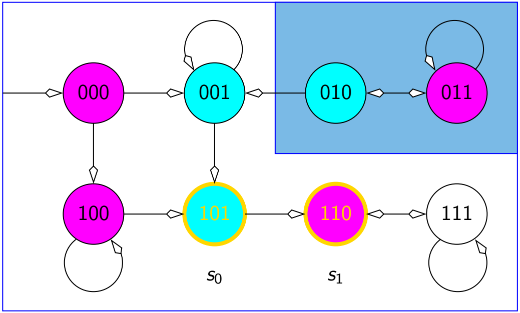
reach$(S, (x_1 \vee \neg x_2), s_0, s_1)$ passes

Example: Empty Language

	s_0	s_1
sk2	101	110

states satisfy

$x_1 \vee \neg x_2$



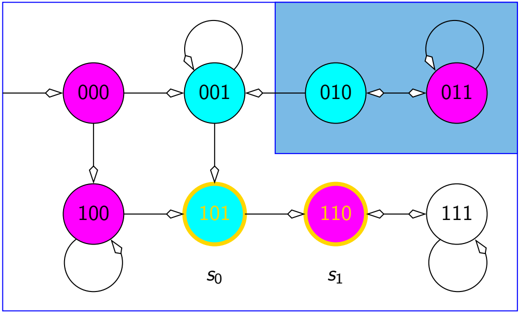
reach$(S, (x_1 \vee \neg x_2), s_1, s_0)$ produces x_2

Example: Empty Language

	s_0	s_1
sk2	101	110

states satisfy

$x_1 \vee \neg x_2$



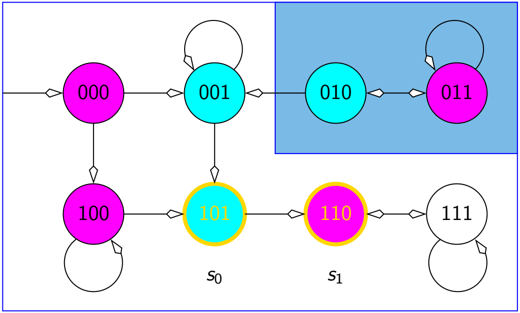
because $x_1 \wedge x_2 \wedge \neg x_3 \Rightarrow x_2 \;\ldots$

Example: Empty Language

	s_0	s_1
sk2	101	110

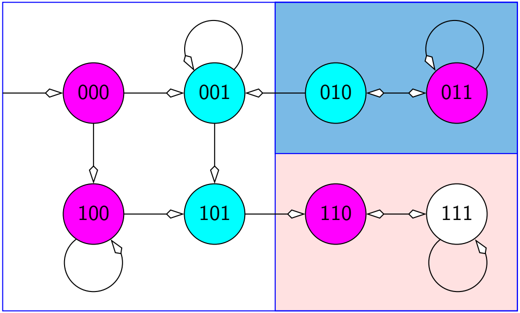
states satisfy

$x_1 \vee \neg x_2$



and $x_2 \wedge (x_1 \vee \neg x_2) \wedge T \Rightarrow x_2'$

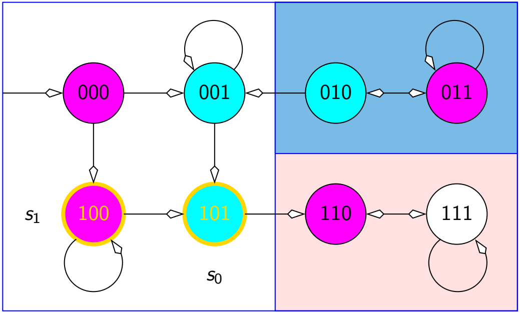
Example: Empty Language



Example: Empty Language

	s_0	s_1
sk3	101	100

states satisfy
$(x_1 \vee \neg x_2) \wedge \neg x_2$

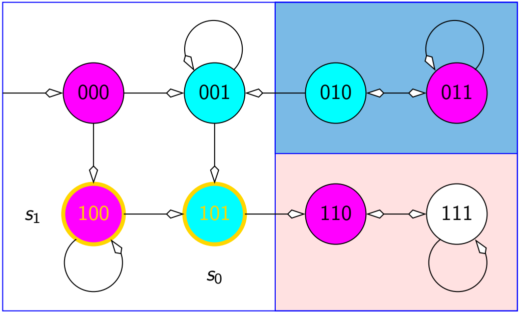


Example: Empty Language

	s_0	s_1
sk3	101	100

states satisfy

$(x_1 \vee \neg x_2) \wedge \neg x_2$

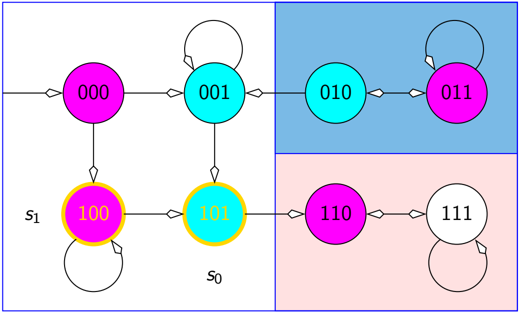


stem query passes

Example: Empty Language

	s_0	s_1
sk3	101	100

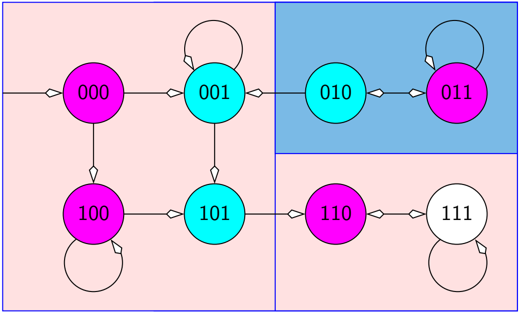
states satisfy
$(x_1 \vee \neg x_2) \wedge \neg x_2$



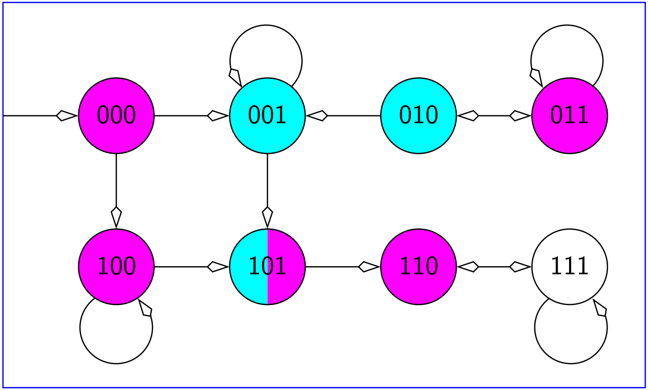
reach$(S, (x_1 \vee \neg x_2) \wedge \neg x_2, s_0, s_1)$ produces $x_2 \vee x_3$

Example: Empty Language

no skeletons left

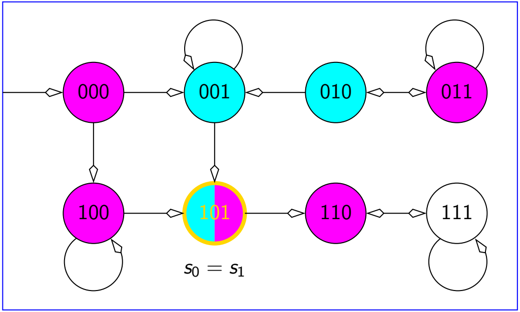


Example: Single-State Skeleton



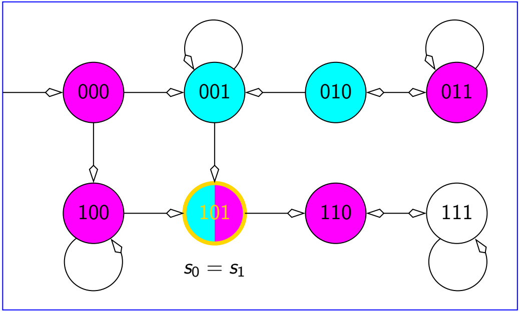
Example: Single-State Skeleton

	s_0	s_1
sk1	101	101



Example: Single-State Skeleton

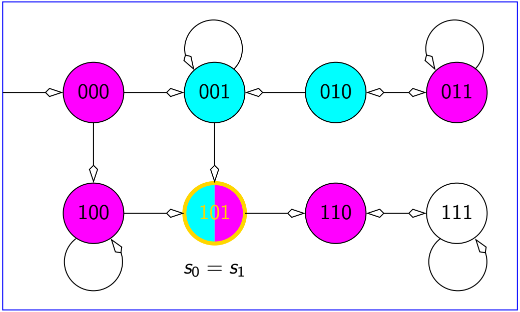
	s_0	s_1
sk1	101	101



stem query passes

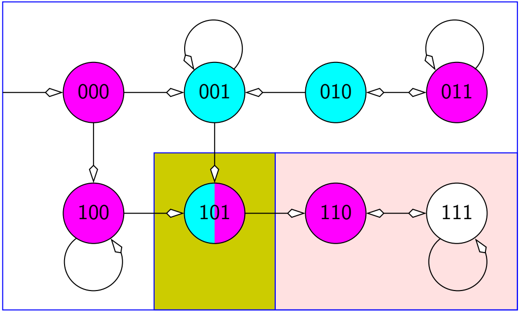
Example: Single-State Skeleton

	s_0	s_1
sk1	101	101



reach$(S, \top, \text{post}(S, s_0), s_0)$ produces $x_1 \wedge x_2$
and $(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3)$

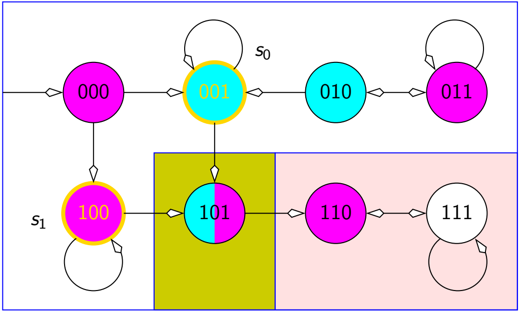
Example: Single-State Skeleton



Example: Single-State Skeleton

	s_0	s_1
sk2	001	100

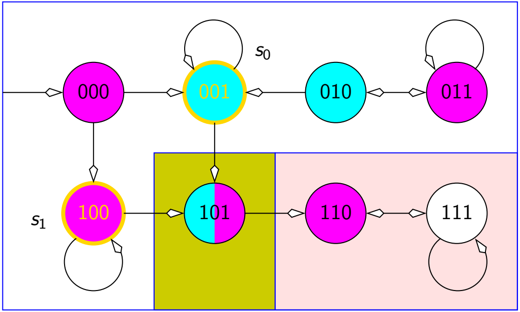
states satisfy
$(\neg x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee \neg x_3)$



Example: Single-State Skeleton

	s_0	s_1
sk2	001	100

states satisfy
$(\neg x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee \neg x_3)$

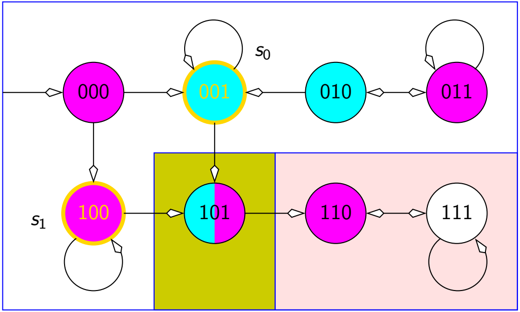


stem query passes

Example: Single-State Skeleton

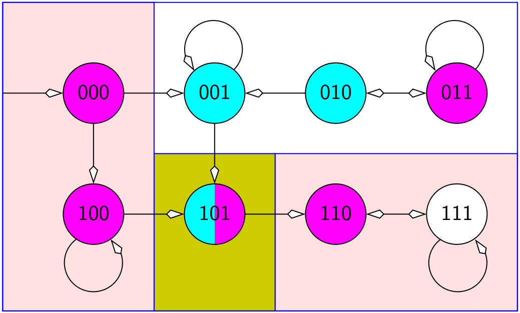
	s_0	s_1
sk2	001	100

states satisfy
$(\neg x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee \neg x_3)$



reach$(S, (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3), s_0, s_1)$ produces $x_2 \vee x_3$

Example: Single-State Skeleton

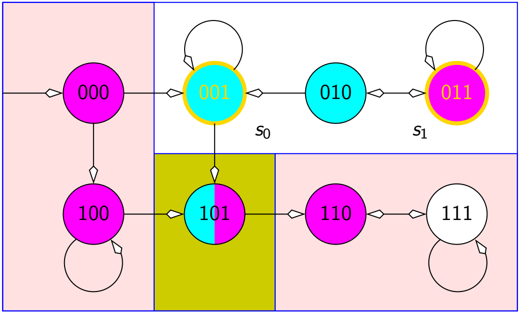


Example: Single-State Skeleton

	s_0	s_1
sk3	001	011

states satisfy

$$(\neg x_1 \vee \neg x_2) \wedge$$
$$(\neg x_1 \vee \neg x_3) \wedge$$
$$(x_2 \vee x_3)$$

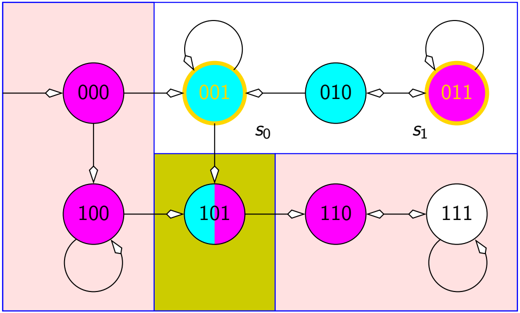


Example: Single-State Skeleton

	s_0	s_1
sk3	001	011

states satisfy

$$(\neg x_1 \vee \neg x_2) \wedge$$
$$(\neg x_1 \vee \neg x_3) \wedge$$
$$(x_2 \vee x_3)$$

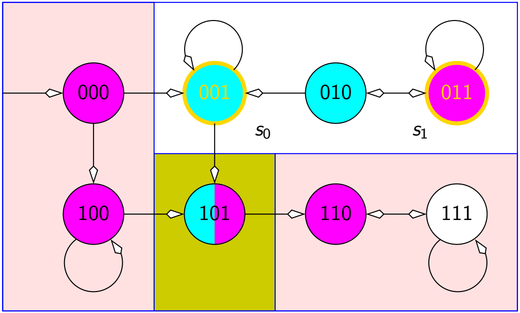


stem query passes

Example: Single-State Skeleton

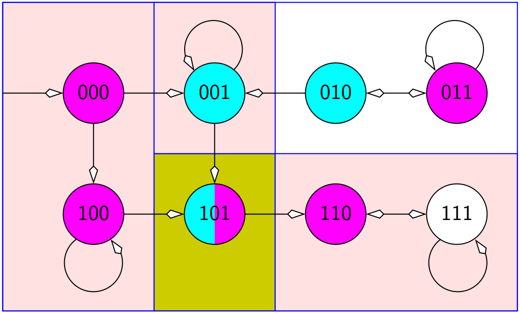
	s_0	s_1
sk3	001	011

states satisfy
$(\neg x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee \neg x_3) \wedge$
$(x_2 \vee x_3)$



reach$(S, (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee x_3), s_0, s_1)$ produces $\neg x_2$

Example: Single-State Skeleton

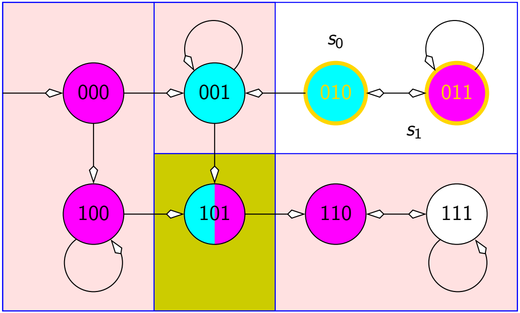


Example: Single-State Skeleton

	s_0	s_1
sk4	010	011

states satisfy

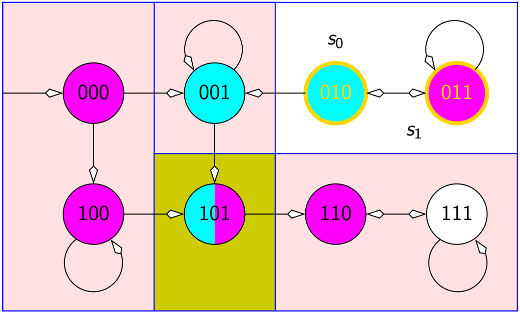
$(\neg x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee \neg x_3) \wedge$
$(x_2 \vee x_3) \wedge x_2$



Example: Single-State Skeleton

	s_0	s_1
sk4	010	011

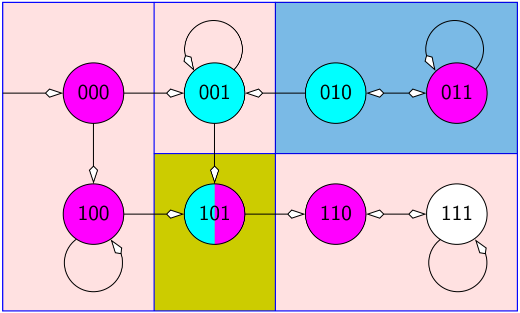
states satisfy
$(\neg x_1 \vee \neg x_2) \wedge$
$(\neg x_1 \vee \neg x_3) \wedge$
$(x_2 \vee x_3) \wedge x_2$



stem query produces $x_1 \vee \neg x_2$

Example: Single-State Skeleton

no skeletons left



Persistent Signals

- Signal p is persistent in structure S if

$$S \models \mathrm{G}(p \rightarrow \mathrm{X}\, p)$$

or

$$S \models \mathrm{G}(\neg p \rightarrow \mathrm{X}\, \neg p)$$

- Checking for persistence by a SAT check:

$$p \wedge T \Rightarrow p'$$

$$\neg p \wedge T \Rightarrow \neg p'$$

Persistent Signals

- Signal p is persistent in structure S if

$$S \models \mathsf{G}(p \rightarrow \mathsf{X}\, p)$$

or

$$S \models \mathsf{G}(\neg p \rightarrow \mathsf{X}\, \neg p)$$

- Checking for persistence by a SAT check:

$$p \wedge T \Rightarrow p'$$

$$\neg p \wedge T \Rightarrow \neg p'$$

Barriers from Persistent Signals

- Signals may be persistent under assumptions
  - Another signal is persistent
  - Another signal has a given value
- A persistent signal defines a barrier
- One side of the barrier may have no skeletons
- Then the persistent signal may be assumed to have a fixed value

Barriers from Persistent Signals

- Signals may be persistent under assumptions
  - Another signal is persistent
  - Another signal has a given value
- A persistent signal defines a barrier
- One side of the barrier may have no skeletons
- Then the persistent signal may be assumed to have a fixed value

Barriers from Persistent Signals

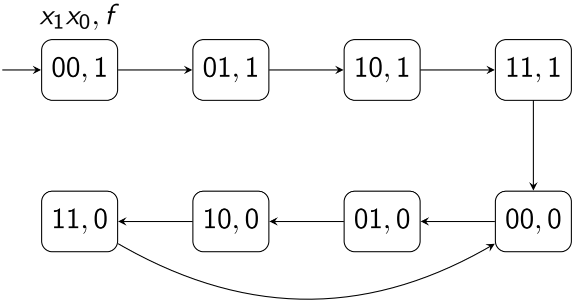
- Signals may be persistent under assumptions
  - Another signal is persistent
  - Another signal has a given value
- A persistent signal defines a barrier
- One side of the barrier may have no skeletons
- Then the persistent signal may be assumed to have a fixed value

Barriers from Persistent Signals

- Signals may be persistent under assumptions
  - Another signal is persistent
  - Another signal has a given value
- A persistent signal defines a barrier
- One side of the barrier may have no skeletons
- Then the persistent signal may be assumed to have a fixed value

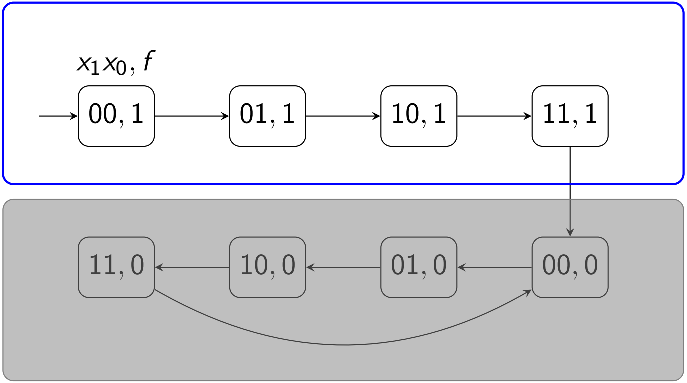
Slice'n'Dice

Property holds if $\mathsf{F}\,\mathsf{G}\,\neg f$.



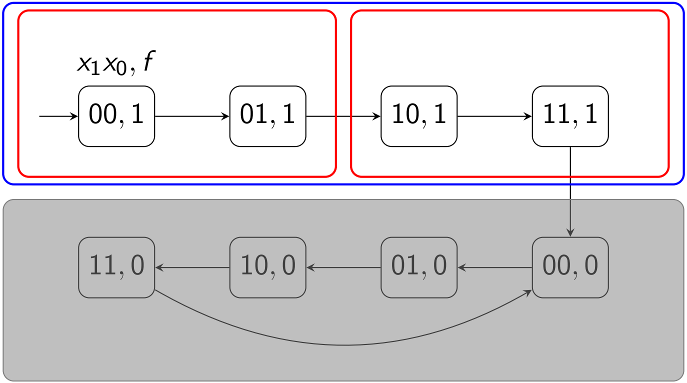
Slice'n'Dice

Property holds if $\mathsf{F}\,\mathsf{G}\,\neg f$.



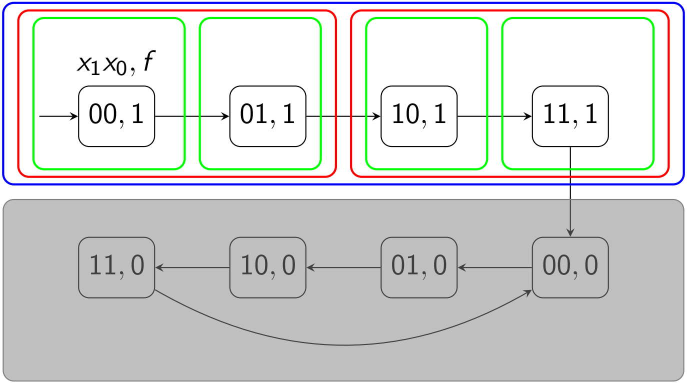
Slice'n'Dice

Property holds if $\mathsf{F}\,\mathsf{G}\,\neg f$.



Slice'n'Dice

Property holds if $\mathrm{F}\,\mathrm{G}\,\neg f$.



Key Insights

- Inductive assertions describe SCC-closed sets.
- Arena: Set of states all on the same side of each barrier.
- Unlike previous symbolic methods:

  Barrier constraints on the transition relation combined with the over-approximating nature of IC3 enable the simultaneous (symbolic) consideration of all arenas.

- A proof can provide information about many arenas even though the motivating skeleton comes from one arena.

Methodological Parallels with IC3

	IC3	**FAIR**
Seed:	CTI	Skeleton
Lemma:	Inductive clause	Global reachability proof One-way barrier
	Relative to previously discovered lemmas.	
CEX:	CTI sequence	Connected skeleton
	Discovery guided by lemmas. Not minimal.	
Proof:	Inductive strengthening	All arenas skeleton-free
	Sufficient set of lemmas.	

k-Liveness (Claessen and Sörensson [2012])

- If property holds in S, then $S \models \mathsf{F}\,\mathsf{G}\,\neg p$
  - p holds finitely many times
- Approximate with sequence of safety properties
  - p is never true
  - p holds at most once
  - p holds at most twice ...
  - p holds at most k times ...
- If any property in the sequence holds, so does $\mathsf{F}\,\mathsf{G}\,\neg p$
- If S is finite-state, then $S \models \mathsf{F}\,\mathsf{G}\,\neg p$ holds only if there is k such that p holds at most k times
- k-liveness is in practice a semi-decision procedure
  - Interleave BMC calls to check whether property fails
- For each value of k, IC3 decides whether safety property holds
  - In principle, any safety model checker would do

k-Liveness (Claessen and Sörensson [2012])

- If property holds in S, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$
  - p holds finitely many times
- Approximate with sequence of safety properties
  - p is never true
  - p holds at most once
  - p holds at most twice ...
  - p holds at most k times ...
- If any property in the sequence holds, so does $\mathrm{F}\,\mathrm{G}\,\neg p$
- If S is finite-state, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$ holds only if there is k such that p holds at most k times
- k-liveness is in practice a semi-decision procedure
  - Interleave BMC calls to check whether property fails
- For each value of k, IC3 decides whether safety property holds
  - In principle, any safety model checker would do

k-Liveness (Claessen and Sörensson [2012])

- If property holds in S, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$
  - p holds finitely many times
- Approximate with sequence of safety properties
  - p is never true
  - p holds at most once
  - p holds at most twice ...
  - p holds at most k times ...
- If any property in the sequence holds, so does $\mathrm{F}\,\mathrm{G}\,\neg p$
- If S is finite-state, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$ holds only if there is k such that p holds at most k times
- k-liveness is in practice a semi-decision procedure
  - Interleave BMC calls to check whether property fails
- For each value of k, IC3 decides whether safety property holds
  - In principle, any safety model checker would do

k-Liveness (Claessen and Sörensson [2012])

- If property holds in S, then $S \models \mathsf{F}\,\mathsf{G}\,\neg p$
  - p holds finitely many times
- Approximate with sequence of safety properties
  - p is never true
  - p holds at most once
  - p holds at most twice ...
  - p holds at most k times ...
- If any property in the sequence holds, so does $\mathsf{F}\,\mathsf{G}\,\neg p$
- If S is finite-state, then $S \models \mathsf{F}\,\mathsf{G}\,\neg p$ holds only if there is k such that p holds at most k times
- k-liveness is in practice a semi-decision procedure
  - Interleave BMC calls to check whether property fails
- For each value of k, IC3 decides whether safety property holds
  - In principle, any safety model checker would do

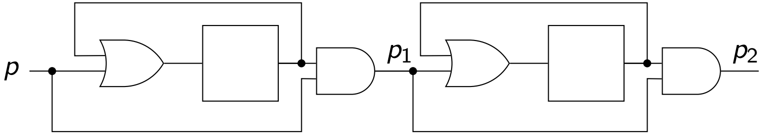
k-Liveness (Claessen and Sörensson [2012])

- If property holds in S, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$
  - p holds finitely many times
- Approximate with sequence of safety properties
  - p is never true
  - p holds at most once
  - p holds at most twice ...
  - p holds at most k times ...
- If any property in the sequence holds, so does $\mathrm{F}\,\mathrm{G}\,\neg p$
- If S is finite-state, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$ holds only if there is k such that p holds at most k times
- k-liveness is in practice a semi-decision procedure
  - Interleave BMC calls to check whether property fails
- For each value of k, IC3 decides whether safety property holds
  - In principle, any safety model checker would do

k-Liveness (Claessen and Sörensson [2012])

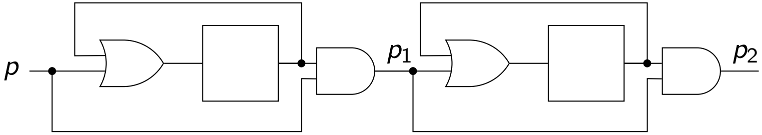
- If property holds in S, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$
  - p holds finitely many times
- Approximate with sequence of safety properties
  - p is never true
  - p holds at most once
  - p holds at most twice ...
  - p holds at most k times ...
- If any property in the sequence holds, so does $\mathrm{F}\,\mathrm{G}\,\neg p$
- If S is finite-state, then $S \models \mathrm{F}\,\mathrm{G}\,\neg p$ holds only if there is k such that p holds at most k times
- k-liveness is in practice a semi-decision procedure
  - Interleave BMC calls to check whether property fails
- For each value of k, IC3 decides whether safety property holds
  - In principle, any safety model checker would do

Iterative Counting Circuit



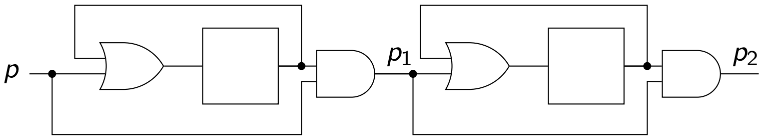
- Each subcircuit *absorbs* one occurrence of p
- Increasing k means adding another instance of the subcircuit
- This solution works well with an incremental safety solver
- General approach relies on representing property as a universal co-Büchi word automaton (Filiot et al. [2009], bounded synthesis)

Iterative Counting Circuit



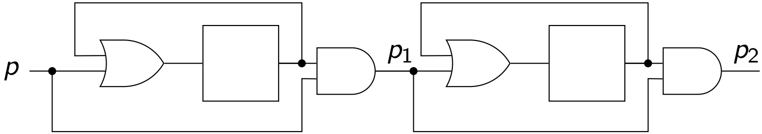
- Each subcircuit *absorbs* one occurrence of p
- Increasing k means adding another instance of the subcircuit
- This solution works well with an incremental safety solver
- General approach relies on representing property as a universal co-Büchi word automaton (Filiot et al. [2009], bounded synthesis)

Iterative Counting Circuit



- Each subcircuit *absorbs* one occurrence of p
- Increasing k means adding another instance of the subcircuit
- This solution works well with an incremental safety solver
- General approach relies on representing property as a universal co-Büchi word automaton (Filiot et al. [2009], bounded synthesis)

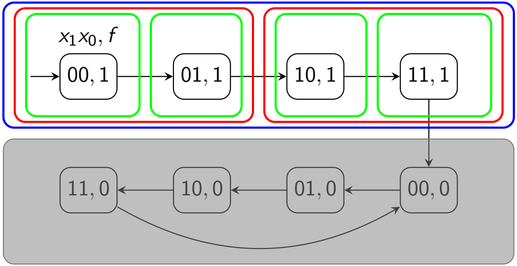
Iterative Counting Circuit



- Each subcircuit *absorbs* one occurrence of p
- Increasing k means adding another instance of the subcircuit
- This solution works well with an incremental safety solver
- General approach relies on representing property as a universal co-Büchi word automaton (Filiot et al. [2009], bounded synthesis)

Persistent Signals in k-Liveness

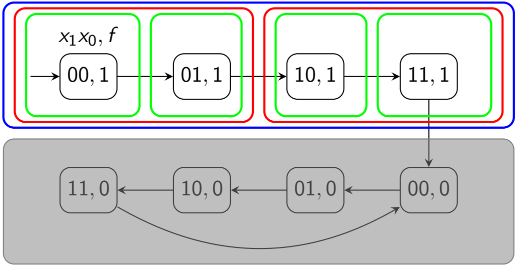
- Persistent signals are computed just as in FAIR
  - They are used to constrain the transition relation
  - More likely to prove the property at a lower k



- Without constraints:
  - $\mathsf{F}\,\mathsf{G}\,\neg f$ proved at $k = 5$
- With constraints $f \wedge (x_1 \leftrightarrow x_1') \wedge (x_0 \leftrightarrow x_0')$:
  - No infinite paths: $\mathsf{F}\,\mathsf{G}\,\neg f$ proved at $k = 0$

Persistent Signals in k-Liveness

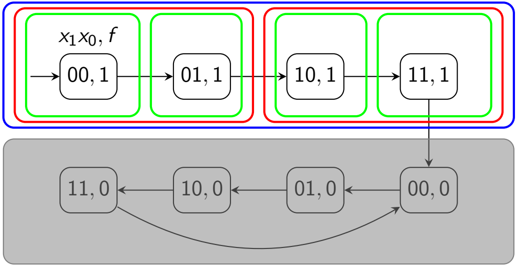
- Persistent signals are computed just as in FAIR
  - They are used to constrain the transition relation
  - More likely to prove the property at a lower k



- Without constraints:
  - $\mathsf{F}\,\mathsf{G}\,\neg f$ proved at $k = 5$
- With constraints $f \wedge (x_1 \leftrightarrow x_1') \wedge (x_0 \leftrightarrow x_0')$:
  - No infinite paths: $\mathsf{F}\,\mathsf{G}\,\neg f$ proved at $k = 0$

Persistent Signals in k-Liveness

- Persistent signals are computed just as in FAIR
  - They are used to constrain the transition relation
  - More likely to prove the property at a lower k



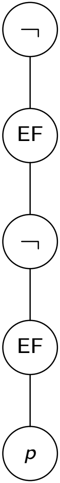
- Without constraints:
  - $\mathsf{F}\,\mathsf{G}\,\neg f$ proved at $k = 5$
- With constraints $f \wedge (x_1 \leftrightarrow x_1') \wedge (x_0 \leftrightarrow x_0')$:
  - No infinite paths: $\mathsf{F}\,\mathsf{G}\,\neg f$ proved at $k = 0$

IICTL: Incremental Inductive CTL Model Checking

- Task-directed strategy
- Maintains upper and lower bounds on states satisfying each subformula
- States in between the bounds are undecided
- Typically don't need to decide all states to decide the property (Traditional symbolic CTL algorithms do)
- Decide states by executing appropriate query:
  - EX: SAT query
  - EU: Safety model checker (e.g., IC3)
  - EG: Fair cycle finder (e.g., FAIR)
- Generalizing decisions (proofs or counterexamples) to other states and refining the bounds

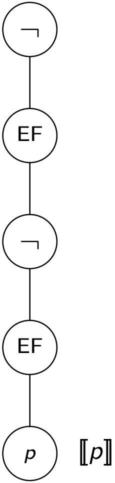
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



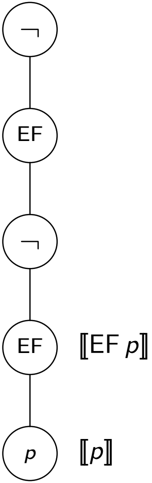
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



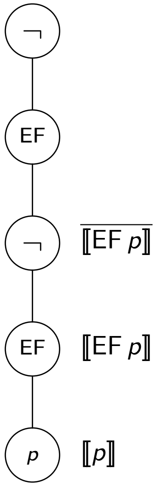
IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EFp$



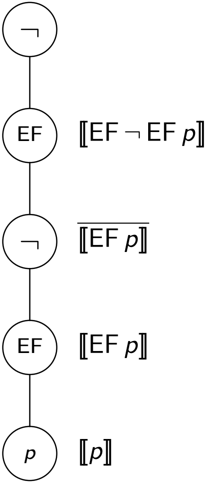
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



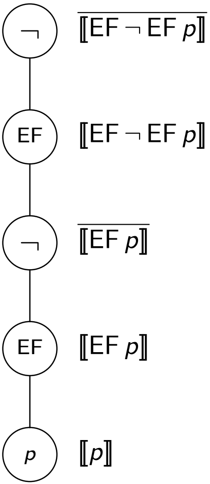
IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EFp$



IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$

$\neg$	$\overline{[\![\mathrm{EF} \neg \mathrm{EF}\, p]\!]} \supseteq$ initial states?
EF	$[\![\mathrm{EF} \neg \mathrm{EF}\, p]\!]$
$\neg$	$\overline{[\![\mathrm{EF}\, p]\!]}$
EF	$[\![\mathrm{EF}\, p]\!]$
p	$[\![p]\!]$

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EF p$

$\neg$ — $\overline{[\![\mathrm{EF}\neg\,\mathrm{EF}\,p]\!]} \supseteq$ initial states?
Yes: Property holds
No: Property fails

EF — $[\![\mathrm{EF}\neg\,\mathrm{EF}\,p]\!]$

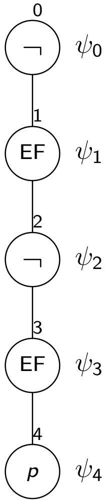
$\neg$ — $\overline{[\![\mathrm{EF}\,p]\!]}$

EF — $[\![\mathrm{EF}\,p]\!]$

p — $[\![p]\!]$

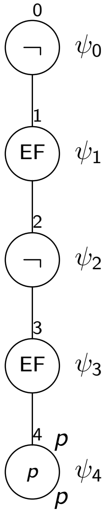
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



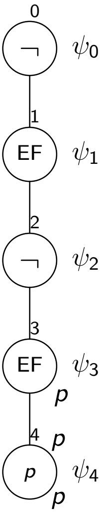
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



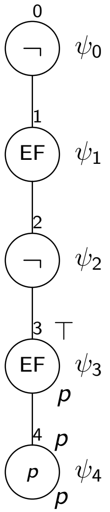
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



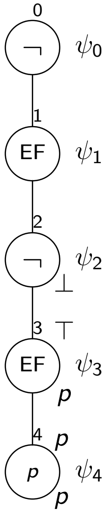
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EF p$



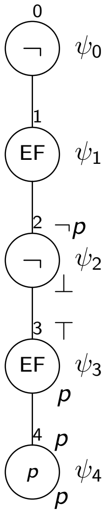
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EF p$



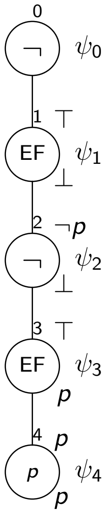
IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



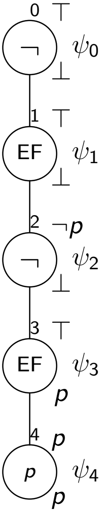
IICTL Example

Property: AG EF $p = \neg EF \neg EF p$



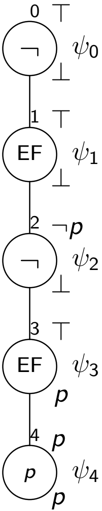
IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF\neg EFp$



IICTL Example

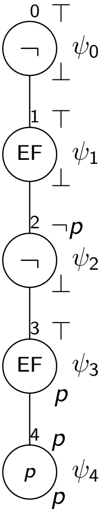
Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EF p$



$I \wedge \neg U_0?$

IICTL Example

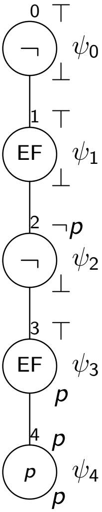
Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? No: Property fails

IICTL Example

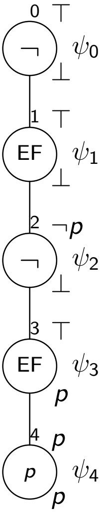
Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes

IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EFp$

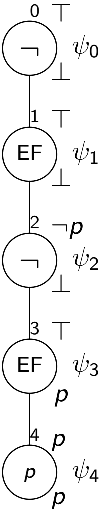


$I \wedge \neg U_0$? Yes

$I \wedge \neg L_0$?

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$

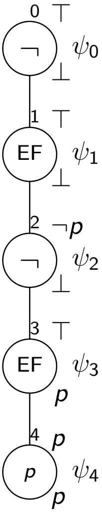


$I \wedge \neg U_0$? Yes

$I \wedge \neg L_0$? Yes: Property holds

IICTL Example

Property: $\mathrm{AG\,EF}\,p = \neg EF \neg EF p$

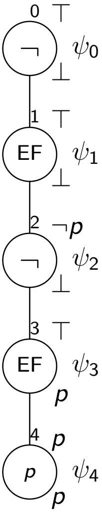


$I \wedge \neg U_0$? Yes

$I \wedge \neg L_0$? No

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$

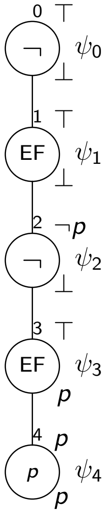


$I \wedge \neg U_0$? Yes

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$

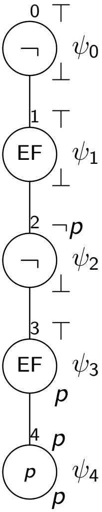


$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



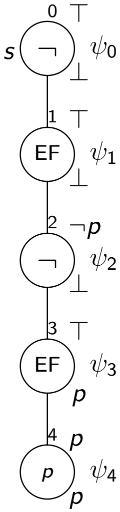
$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



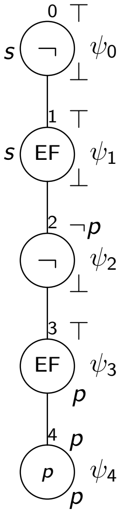
$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



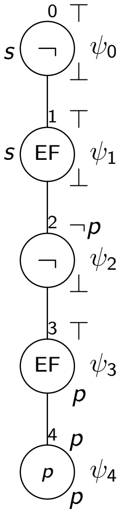
$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

IICTL Example

Property: $\text{AG}\,\text{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

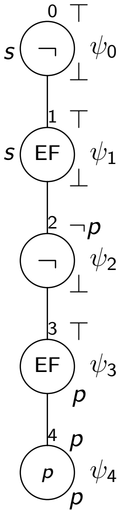
$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

$s \models \psi_1$?

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EF p$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

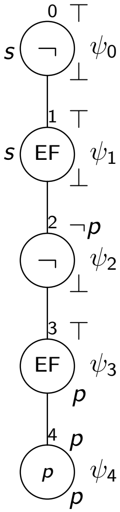
$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

$s \models \psi_1 ? \iff s \models \mathrm{EF}\,\psi_2 ?$

IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EF p$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

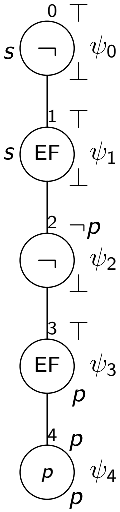
$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

$s \models \psi_1? \Longleftrightarrow \; s \models \mathsf{EF}\,\psi_2? \; \Longleftrightarrow \;$ can s reach ψ_2?

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

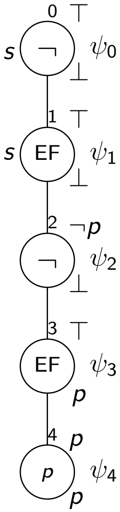
$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

$s \models \psi_1? \iff s \models \mathrm{EF}\,\psi_2? \iff$ can s reach ψ_2? can s reach L_2?

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

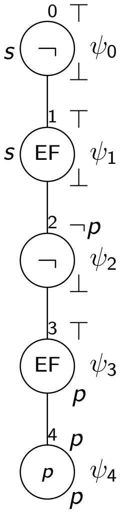
$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

$s \models \psi_1? \iff s \models \mathrm{EF}\,\psi_2? \iff$ can s reach ψ_2?
can s reach L_2? Yes: s can also reach ψ_2

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

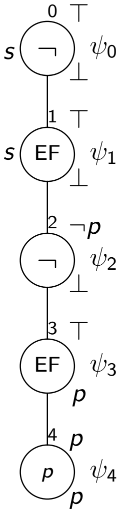
s is *undecided* for node 0

$s \models \psi_1? \iff s \models \mathrm{EF}\,\psi_2? \iff$ can s reach ψ_2?

can s reach L_2? No

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

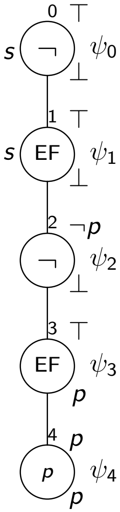
can s reach U_2?

$s \models \psi_1? \iff s \models \mathrm{EF}\,\psi_2? \iff$ can s reach ψ_2?

can s reach L_2? No

IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

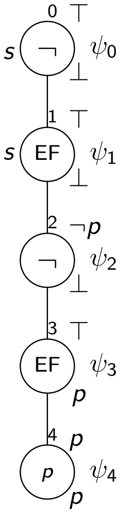
can s reach U_2? No: s cannot reach ψ_2

$s \models \psi_1? \iff s \models \mathsf{EF}\,\psi_2? \iff$ can s reach ψ_2?

can s reach L_2? No

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EF p$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

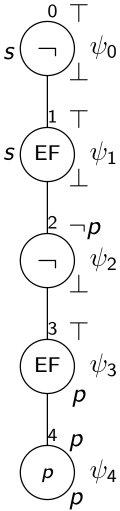
can s reach U_2? Yes

$s \models \psi_1? \iff s \models \mathrm{EF}\,\psi_2? \iff$ can s reach ψ_2?

can s reach L_2? No

IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

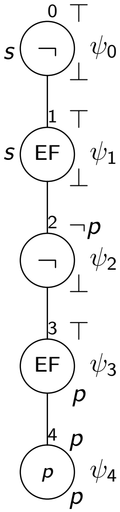
can s reach U_2? Yes $\quad t \models U_2$

$s \models \psi_1? \iff s \models \mathsf{EF}\,\psi_2? \iff$ can s reach ψ_2?

can s reach L_2? No

IICTL Example

Property: $\mathrm{AG\,EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

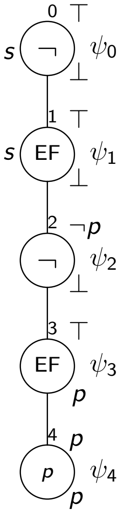
can s reach U_2? Yes $\quad t \models U_2$

$s \models \psi_1? \iff s \models \mathrm{EF}\,\psi_2? \iff$ can s reach ψ_2?

can s reach L_2? No $\quad t \not\models L_2$

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

s is *undecided* for node 0

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

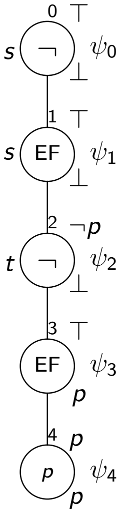
can s reach U_2? Yes $\quad t \models U_2$

t is undecided for node 2

can s reach L_2? No $\quad t \not\models L_2$

IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

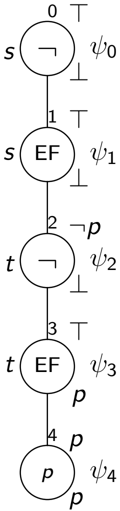
can s reach U_2? Yes $\quad t \models U_2$

can s reach L_2? No $\quad t \not\models L_2$

t is undecided for node 2

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

s is *undecided* for node 0

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

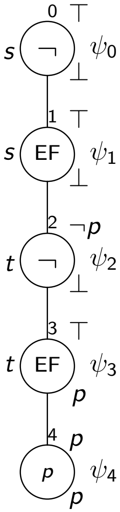
can s reach U_2? Yes $\quad t \models U_2$

t is undecided for node 2

can s reach L_2? No $\quad t \not\models L_2$

IICTL Example

Property: $\mathsf{AG}\,\mathsf{EF}\,p = \neg EF \neg EF p$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

s is *undecided* for node 0

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

can s reach U_2? Yes $\quad t \models U_2$

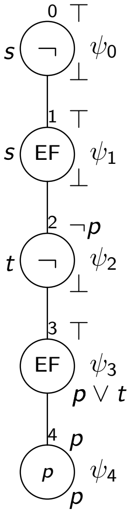
t is undecided for node 2

can s reach L_2? No $\quad t \not\models L_2$

can t reach L_4 (or U_4)?

IICTL Example

Property: $\mathrm{AG\,EF}\, p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

can s reach U_2? Yes $\quad t \models U_2$

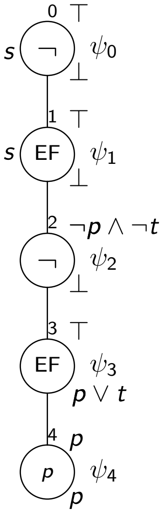
can s reach L_2? No $\quad t \not\models L_2$

t is undecided for node 2

can t reach L_4 (or U_4)?

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

can s reach U_2? Yes $\quad t \models U_2$

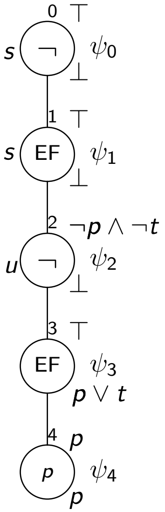
t is undecided for node 2

can s reach L_2? No $\quad t \not\models L_2$

can t reach L_4 (or U_4)?

IICTL Example

Property: $\mathsf{AG\,EF}\,p = \neg EF \neg EF p$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

can s reach U_2? Yes $\quad t \models U_2$

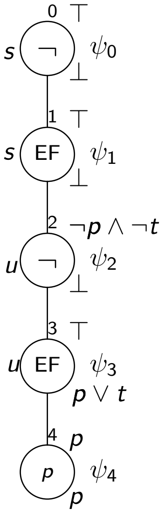
can s reach L_2? No $\quad t \not\models L_2$

t is undecided for node 2

can t reach L_4 (or U_4)?

IICTL Example

Property: $\mathrm{AG}\,\mathrm{EF}\,p = \neg EF \neg EFp$



$I \wedge \neg U_0$? Yes $\quad s \models U_0$

$I \wedge \neg L_0$? No $\quad s \not\models L_0$

s is *undecided* for node 0

can s reach U_2? Yes $\quad t \models U_2$

t is undecided for node 2

can s reach L_2? No $\quad t \not\models L_2$

can t reach L_4 (or U_4)?

IICTL Algorithm

IICTL

1. Construct the parse-graph of the formula
2. Initialize bounds
3. Are all initial states in lower bound of root node?
   Yes: property holds
4. Is any of the initial states not in upper bound of root?
   Yes: property fails
5. There is an *undecided* state s. Decide s recursively and generalize.
6. Repeat step 3