# IA169 Model Checking

## Property directed reachability (PDR/IC3)

Jan Strejček

Faculty of Informatics
Masaryk University

# Short history of IC3/PDR

IC3

- the tool introduced in 2010
  (3rd place in Hardware Model Checking Competition 2010)
- abbreviation for Incremental Construction of Inductive Clauses for Indubitable Correctness
- described in A. R. Bradley: *SAT-Based Model Checking Without Unrolling*, VMCAI 2011.

PDR

- name for the technique implemented in IC3
- abbreviation for Property Directed Reachability
- suggested by N. Een, A. Mishchenko, and R. Brayton
- they also simplified and improved the algorithm

# Short history of IC3/PDR

- originally formulated for finite systems where states are valuations of Boolean variables: good for HW, not for SW
- later generalized for other kinds of systems, including programs
- combined with predicate abstraction, k-induction, ...

IC3/PDR is currently considered to be one of the most powerfull verification techniques.

# Agenda and sources

agenda

- representation of a finite system by Boolean formulas
- property directed reachability

source

- N. Een, A. Mishchenko, and R. Brayton: *Efficient Implementation of Property Directed Reachability*, FMCAD 2011.

Special thanks to Marek Chalupa for providing me his slides.

# Other important papers about IC3/PDR

- K. Hoder and N. Bjorner: *Generalized Property Directed Reachability*, SAT 2012.
- A. Cimatti, A. Griggio: *Software Model Checking via IC3*, CAV 2012.
- A. R. Bradley: *Understanding IC3*, SAT 2012.
- T. Welp, A. Kuehlmann: *QF_BV Model Checking with Property Directed Reachability*, DATE 2013.
- A. Cimatti, A. Griggio, S. Mover, S. Tonetta: *IC3 Modulo Theories via Implicit Predicate Abstraction*, TACAS 2014.
- J. Birgmeier, A. R. Bradley, G. Weissenbacher: *Counterexample to Induction-Guided-Abstraction-Refinement (CTIGAR)*, CAV 2014.
- D. Jovanović, B. Dutertre: *Property-Directed k-Induction*, FMCAD 2016.
- A. Gurfinkel, A. Ivrii: *K-Induction without Unrolling*, FMCAD 2017.

# Representation of a finite system by Boolean formulas

# Formalization of the problem

finite state machine

- set of state variables $\bar{x} = \{x_1, x_2, \dots, x_n\}$
- states are valuations $v : \bar{x} \rightarrow \{0, 1\}$
- initial states given by a propositional formula $I$ over $\bar{x}$
- transition relation given by a propositional formula $T$ over $\bar{x} \cup \bar{x}'$, where $\bar{x}' = \{x'_1, \dots, x'_n\}$ describe the target states

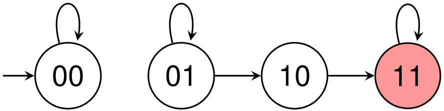
property

- given by a propositional formula $P$ over $\bar{x}$

**The problem**

To decide whether all reachable states of a given finite state machine $(\bar{x}, I, T)$ satisfy a given property $P$.

# Example




$$\bar{x} = \{x_1, x_2\}$$

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

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

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

# Property directed reachability

# Terminology and notation

- for any formula $F$ over $\bar{x}$, $F'$ denotes the same formula over $\bar{x}'$
- cube is a conjunction of literals
- clause is a disjunction of literals
- negation of a cube is a clause (and vice versa)
- a cube with all variables of $\bar{x}$ represents at most one state
- a set of clauses $R = \{c_1, \dots, c_k\}$ is interpreted as conjunction $c_1 \land \dots \land c_k$
- each formula can be identified with a set of states (and vice versa)

# Intuition

A set $S$ of states is inductive invariant if $S \wedge T \implies S'$.

We are looking for an inductive invariant $S$ satisfying

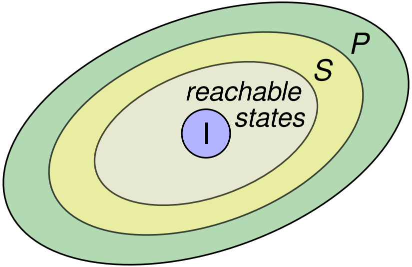
- $I \implies S$ (i.e. $S$ contains all reachable states) and
- $S \implies P$ (i.e. all states of $S$ satisfy the property).

# Intuition

A set $S$ of states is inductive invariant if $S \wedge T \implies S'$.

We are looking for an inductive invariant $S$ satisfying

- $I \implies S$ (i.e. $S$ contains all reachable states) and
- $S \implies P$ (i.e. all states of $S$ satisfy the property).

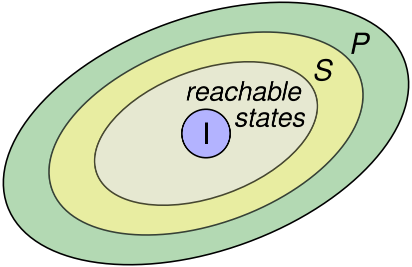


# Intuition

A set $S$ of states is inductive invariant if $S \wedge T \implies S'$.

We are looking for an inductive invariant $S$ satisfying

- $I \implies S$ (i.e. $S$ contains all reachable states) and
- $S \implies P$ (i.e. all states of $S$ satisfy the property).




Note that $P$ does not have to be an inductive invariant.

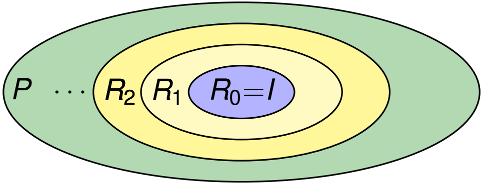
# Traces

The algorithm gradually builds traces, which are sequences $R_0, R_1, \ldots, R_N$ of formulas called frames such that

- $R_0 = I$

and for all $i < N$

- $R_i \implies R_{i+1}$
- $R_i \wedge T \implies R'_{i+1}$
- $R_i \implies P$



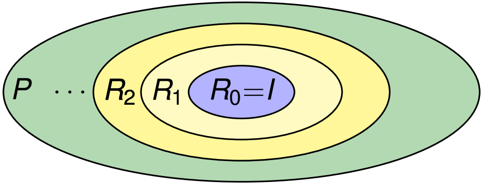
# Traces

The algorithm gradually builds traces, which are sequences $R_0, R_1, \ldots, R_N$ of formulas called frames such that

- $R_0 = I$

and for all $i < N$

- $R_i \implies R_{i+1}$
- $R_i \wedge T \implies R'_{i+1}$
- $R_i \implies P$




Intuitively, each $R_i$ represents a superset of states reachable from initial states in at most $i$ steps.

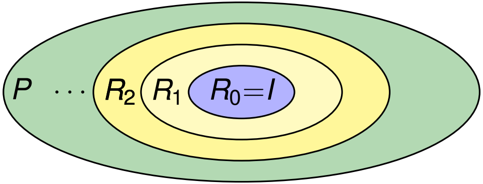
# Traces

The algorithm gradually builds traces, which are sequences $R_0, R_1, \ldots, R_N$ of formulas called frames such that

- $R_0 = I$

and for all $i < N$

- $R_i \implies R_{i+1}$
- $R_i \wedge T \implies R'_{i+1}$
- $R_i \implies P$




Intuitively, each $R_i$ represents a superset of states reachable from initial states in at most $i$ steps.

Moreover, for each $i > 0$ it holds that

- $R_i$ is a set of clauses
- $R_{i+1} \subseteq R_i$ (which implies $R_i \implies R_{i+1}$)

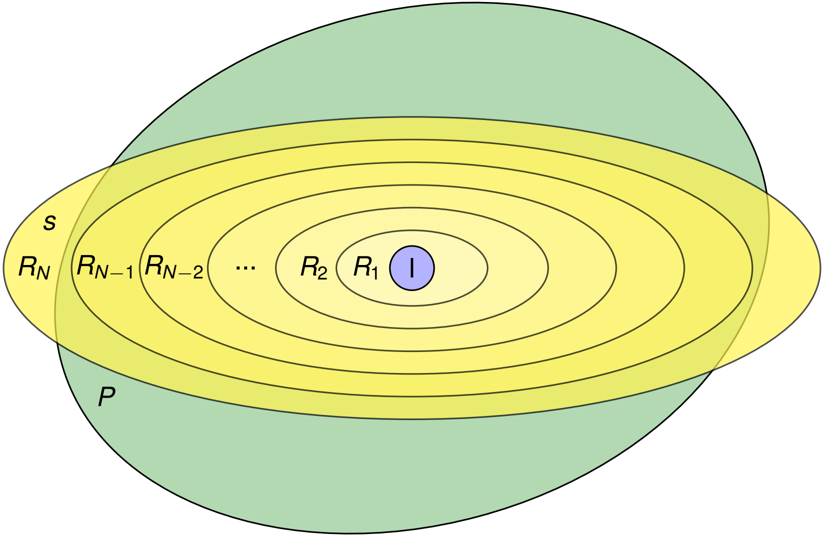
# Proof-obligations

- let $R_0, \ldots, R_N$ be a trace where $R_N \implies P$ does not hold
- let $s$ be a state satisfying $R_N \wedge \neg P$
- we want to prove that $s$ is not reachable in $N$ steps
  $\rightsquigarrow$ so called proof-obligation $(s, N)$

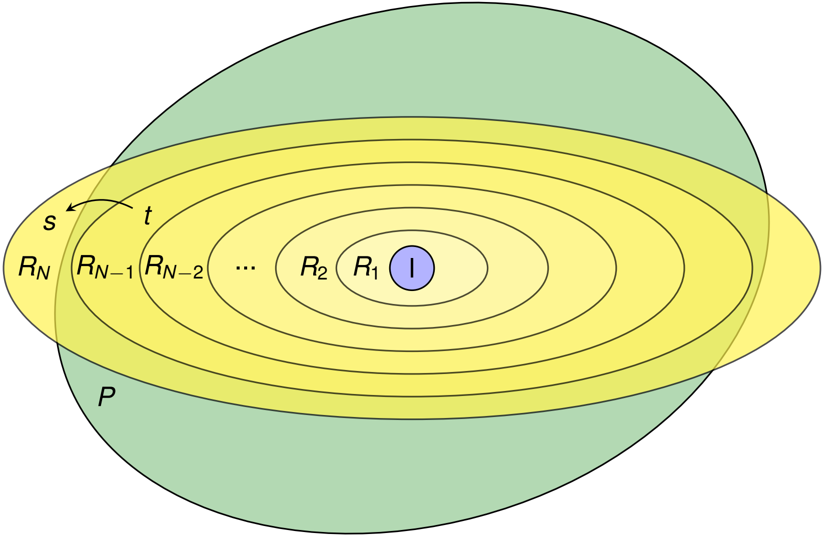
# Solving proof-obligation $(s, k)$

1. check satisfiability of $R_{k-1} \wedge T \wedge s'$
2. if unsatisfiable, then
   - $R_{k-1}$ is strong enough to block $s$
   - thus we can add the clause $\neg s$ to $R_k$
   - we add it also to all $R_1, \ldots, R_{k-1}$ to keep $R_{i+1} \subseteq R_i$ valid
   - proof-obligation solved
3. if satisfiable, then
   - $s$ has some immediate predecessor $t$ in $R_{k-1}$
   - if $k - 1 = 0$ then return property violated and extract counterexample from proof-obligations
   - if $k - 1 > 0$ then solve proof-obligation $(t, k - 1)$ and go to 1

# Proof-obligations



# Proof-obligations



# PDR: high level view

1. if $I \land \neg P$ is satisfiable then return property violated
2. $R_0 := I$
3. $N := 0$
4. while $R_N \land \neg P$ is satisfiable do
   - find a state $s$ satisfying $R_N \land \neg P$
   - solve proof-obligation $(s, N)$
5. $R_{N+1} := \emptyset$
6. $N := N + 1$
7. propagate learned clauses
   - for each $i$ from 1 to $N - 1$
   - for each clause $c \in R_i$, if $R_i \land T \implies c'$ then add $c$ to $R_{i+1}$
8. if $R_i = R_{i+1}$ for some $i$ then return property satisfied
   ($R_i$ is inductive invariant)
9. go to 4

# Termination

Termination follows from finiteness of considered systems

- each proof-obligation must be solved in finitely many steps (either successfully or by detection of proterty violation)
- if the shortest path to a state violating $P$ has $j$ steps, then some state violating $P$ is discovered when $N = j$
- if $P$ is satisfied, an inductive invariant is eventually found as
  - there are only finitely many sets of states
  - $R_0, R_1, \ldots, R_N$ always represent sets ordered by inclusion
  - if $R_i$ and $R_{i+1}$ become semantically equivalent, then clause propagation makes them also syntactically equivalent

# Termination

Termination follows from finiteness of considered systems

- each proof-obligation must be solved in finitely many steps (either successfully or by detection of proterty violation)
- if the shortest path to a state violating $P$ has $j$ steps, then some state violating $P$ is discovered when $N = j$
- if $P$ is satisfied, an inductive invariant is eventually found as
  - there are only finitely many sets of states
  - $R_0, R_1, \ldots, R_N$ always represent sets ordered by inclusion
  - if $R_i$ and $R_{i+1}$ become semantically equivalent, then clause propagation makes them also syntactically equivalent

Still, for a system with $\bar{x} = \{x_1, \ldots, x_n\}$, we may need a trace with up to $2^n$ elements to find an inductive invariant.

# PDR: important tricks

The presented algorithm is correct, but slow.
PDR uses several tricks to boost efficiency, in particular it

- generalizes blocked states
- uses relative induction in proof-obligation solving
- blocks states in future frames

# Generalization of blocked states

- the presented proof-obligation algorithm adds $\neg s$ to $R_k$ when $s$ is blocked, i.e. $R_{k-1} \wedge T \wedge s'$ is unsatisfiable
- PDR generalizes this state to a set of states that are blocked for the same reason
- there are several ways to achieve that
  - use ternary simulation
  - use unsat cores
  - use interpolants
  - manually drop parts of $s$

# Generalization of blocked states

- the presented proof-obligation algorithm adds $\neg s$ to $R_k$ when $s$ is blocked, i.e. $R_{k-1} \wedge T \wedge s'$ is unsatisfiable
- PDR generalizes this state to a set of states that are blocked for the same reason
- there are several ways to achieve that
  - use ternary simulation
  - use unsat cores
  - use interpolants
  - manually drop parts of $s$

use of unsat cores

- one can build the cube $r'$ of the literals of $s'$ that appear in the unsat core and then add $\neg r$ to $R_k$
- the clause $\neg r$ is smaller than $\neg s$ and represents less states

# Relative induction in proof-obligation solving

- to solve proof-obligation $(s, k)$, we checked $R_{k-1} \wedge T \wedge s'$
- PDR checks satisfiability of $R_{k-1} \wedge \neg s \wedge T \wedge s'$ instead
- this query is more likely to be unsatisfied (it has one more clause) and state $s$ can be blocked sooner
- in fact, it checks whether $\neg s$ is inductive relative to $R_{k-1}$:
  the query is unsatisfiable iff $(R_{k-1} \wedge \neg s \wedge T) \implies \neg s'$
- intuitively, in this way we ignore self-loops of the system

# Relative induction in proof-obligation solving

- to solve proof-obligation $(s, k)$, we checked $R_{k-1} \wedge T \wedge s'$
- PDR checks satisfiability of $R_{k-1} \wedge \neg s \wedge T \wedge s'$ instead
- this query is more likely to be unsatisfied (it has one more clause) and state $s$ can be blocked sooner
- in fact, it checks whether $\neg s$ is inductive relative to $R_{k-1}$:
  the query is unsatisfiable iff $(R_{k-1} \wedge \neg s \wedge T) \implies \neg s'$
- intuitively, in this way we ignore self-loops of the system

- in fact, PDR combines this technique with the generalization of blocked clauses
- thus, PDR searches for a subclause (ideally minimal) $c \subseteq \neg s$ such that
  $I \implies c$ and $(R_{k-1} \wedge c \wedge T) \implies c'$

# The end

Thank you for your attention!

- oral exam (30 min preparation + 30 min exam)
- every student gets one randomly selected topic to explain
  - automata-based LTL model checking (without translation of LTL to BA)
  - CTL model checking
  - symbolic model checking for CTL
  - bounded model checking and k-induction
  - reachability in pushdown systems
  - abstraction and CEGAR
  - property directed reachability