Bachelor Thesis

Property Directed Reachability

Proposal

Jonas Werner

7. May 2018

1 Introduction

SAT-based model-checking is a useful technique for both software and hardware verification. Most modern model-checkers are based on interpolation [1]. Recently a novel algorithm was devised by Aaron Bradley [2] called `IC3`. Because it was so new, it came as a surprise that it won third place in the hardware model-checking competition (HWMCC) at CAV 2010.
The model-checking method behind `IC3` is called *Property Directed Reachability*, *PDR* for short, which is not based on interpolation but on backward-search.
Ultimate [3] is a program analysis framework consisting of multiple plugins that perform steps of a program analysis, like parsing source code, transforming programs from one representation to another, or analyse programs. Ultimate already has analysis-plugins using different model-checking techniques like trace abstraction [7] or lazy interpolation [8]. The goal of this Bachelor's Thesis is to implement a new analysis-plugin that uses PDR in Ultimate and to compare it with the other techniques.

2 Bit-Level PDR

In the following I will describe the basic principle behind PDR as a hardware-checker as used in `IC3`, therefore we use only boolean variables. It is however possible to use PDR as a software-checker as shown later in chapter 3.

2.1 Preliminaries

First some preliminary definitions and notations:

A *literal* is a variable or its negation, e.g., x or $\neg y$
A *clause* is a disjunction of literals, e.g., $x \vee \neg y$
A *cube* is a conjunction of literals, e.g., $x \wedge \neg y$
Therefore, the negation of a cube is a clause. $\neg(x \wedge \neg y) \equiv (\neg x \vee y)$

A *boolean transition system* is a tuple $S = (X, I, T)$ where X is a finite set of boolean variables, I is a cube representing the *initial state*, and T is a propositional formula over variables in X and $X' = \{x \in X \mid x' \in X'\}$, called transition relation, that describes updates to the variables.

For example, consider the transition system $U = (X, I, T)$ where
$X = \{x_1, x_2, x_3\}$
$I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$
$T = (x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')$
With transition graph:

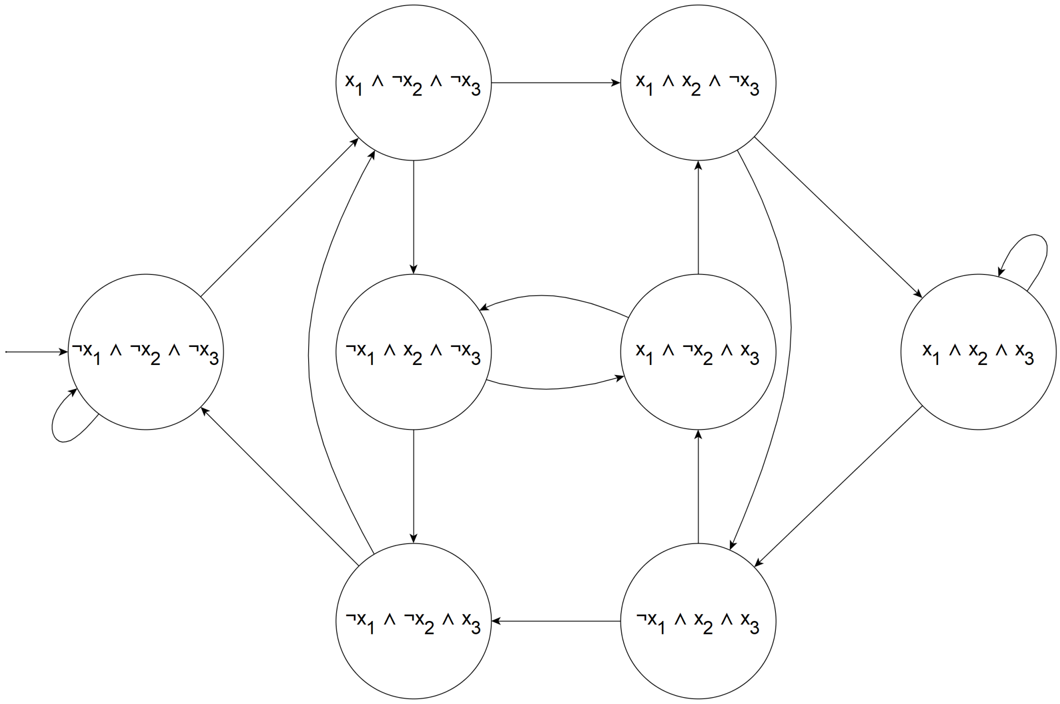


Figure 1: Transition Graph of U

Given a propositional formula ϕ over X we get a *primed formula* ϕ' by replacing each variable with its corresponding variable in X'.

A *state* in S is a cube containing each variable from X with a boolean valuation of it. For each possible valuation there is a corresponding state, resulting in $2^{|X|}$ states in S.
Like we see in the graph of U we have $2^{|X|} = 2^3 = 8$ states.

A *transition* from one state s to another state q exists if the conjunction of s, the transition relation, and q' is satisfiable.
For example in U the transition between the initial state $I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$ and state $r = x_1 \wedge \neg x_2 \wedge \neg x_3$ exists because

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{I} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_1' \wedge \neg x_2' \wedge \neg x_3'}_{r'}$$

is satisfiable.

Given a propositional formula P over X, called *property*, we want to verify that every state in S that is reachable from I satisfies P such that, P describes a set of *good states*, conversely $\neg P$ represent a set of *bad* states.
Regarding U, let $P = \neg x_1 \vee \neg x_2 \vee \neg x_3$ be given, making $\neg P = x_1 \wedge x_2 \wedge x_3$ a bad state.
We can use PDR to show that either $\neg P$ is unreachable from I or that there exists a sequence of transitions leading to $\neg P$ as counter-example.

2.2 Algorithm

A PDR-based algorithm tries to prove that a transition system $S = (X, I, T)$ satisfies a given property P by trying to find a formula F over X with the following qualities:

(1) $I \Rightarrow F$

(2) $F \wedge T \Rightarrow F'$

(3) $F \Rightarrow P$

F is called an *inductive invariant.*

To calculate an inductive invariant, PDR uses *frames* which are cubes of clauses representing an over-approximation of reachable states in at most i transitions from I.
PDR maintains a sequence of frames $[F_0, ..., F_k]$, called a *trace*, it is organized so that it fulfills the following characteristics:

(I) $F_0 = I$

(II) $F_{i+1} \subseteq F_i$, therefore $F_i \Rightarrow F_{i+1}$

(III) $F_i \wedge T \Rightarrow F'_{i+1}$

(IV) $F_i \Rightarrow P$

Now to the algorithm itself:
Start with checking for a *0-counter-example*, that means checking if $I \Rightarrow P$, by testing whether the formula $I \wedge \neg P$ is satisfiable. If it is, then I is a 0-counter-example, the algorithm terminates. If the formula is unsatisfiable, initialize the first frame $F_0 = I$, fulfilling (I), and moving on.

Let $[F_0, F_1, ..., F_k]$ be the current trace.
The algorithm repeats the following three phases until termination:

1. Next Transition
Check whether the next state is a good state meaning $F_k \wedge T \Rightarrow P'$ is valid, by testing the satisfiability of $F_k \wedge T \wedge \neg P'$

- If the formula is *satisfiable*, for each satisfying assignment $\vec{x} = (x_1, x_2, ..., x_{|X|}, x'_1, x'_2, ..., x'_{|X'|})$ get a new bad state $a = x_1 \wedge x_2 \wedge ... \wedge x_{|X|}$ and create tuple (a, k), this tuple is called a *proof-obligation*.
- If the formula is *unsatisfiable*, continue with the next phase.

2. Blocking-Phase
If there are proof-obligations:
Take proof-obligation (b, i) and try to block the bad state b by checking if frame F_{i-1} can reach b in one transition, i.e., test $F_{i-1} \wedge T \wedge b'$ for satisfiability.

- If the formula is *satisfiable*, it means that F_i is not strong enough to block b. For each satisfying assignment $\vec{x} = (x_1, x_2, ..., x_{|X|}, x'_1, x'_2, ..., x'_{|X'|})$ get a new bad state $c = x_1 \wedge x_2 \wedge ... \wedge x_{|X|}$ creating the new proof-obligation $(c, i-1)$.
- If the formula is *unsatisfiable*, strengthen frame F_i with $\neg b$ meaning $F_i = F_i \wedge \neg b$, blocking b at F_i

This continues recursively until either a proof-obligation $(d, 0)$ is created proving that there exists a counter-example terminating the algorithm, or every proof-obligation is blocked.

3. Propagation-Phase
Add a new frame $F_{k+1} = P$ and propagate clauses from F_k forward, meaning for all clauses c in F_k check $F_k \wedge T \wedge \neg c'$ for satisfiability. If that formula is unsatisfiable, strengthen F_{k+1} with c: $F_{k+1} = F_{k+1} \wedge c$, else do nothing and continue with the next clause. Because of this phase rule (II) is fulfilled.

After propagating all possible clauses, if $F_{k+1} \equiv F_k$ the algorithm found a fixpoint and terminates returning that P always holds with F_k being the inductive invariant.

To illustrate the procedure further consider the pseudo-code:

Algorithm 1 PDR-prove

procedure PDR-PROVE(I, T, P)
  check for 0-counter-example
  $F_0 = new\ frame(I)$
  $trace.push(F_0)$ ▷ first element of trace is initial states

  loop
   Blocking Phase:
   while $\exists$ cube c, s.t. $trace.last() \land T \land c'$ is SAT and $c \Rightarrow \neg P$ **do**
    recursively block proof-obligation(c, trace.size() - 1)
    and strengthen the frames of the trace.
    if a proof-obligation(p, 0) is generated **then**
     return false ▷ counter-example found
   Propagation Phase:
   $F_{k+1} = new\ frame(P)$
   for all clause c $\in trace.last()$ **do**
    if $trace.last() \land T \land \neg c'$ is UNSAT **then**
     $F_{k+1} = F_{k+1} \land c$
   if $trace.last() == F_{k+1}$ **then**
    return true ▷ property proven
   $trace.push(F_{k+1})$

2.3 Examples

2.3.1 With Failing Property

To show an application of the algorithm reconsider the example transition system $U = (X, I, T)$ with
$X = \{x_1, x_2, x_3\}$,
$I = \neg x_1 \land \neg x_2 \land \neg x_3$,

$T = (x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')$

and the property:
$P = \neg x_1 \vee \neg x_2 \vee \neg x_3$ with bad state $\neg P = x_1 \wedge x_2 \wedge x_3$
We now want to verify whether P holds or if there is a counter-example.

1. Step: Check for 0-Counter-Example
We need to make sure that $I \Rightarrow P$, we do that by testing if $I \wedge \neg P$ is satisfiable:

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{\mathrm{I}} \wedge \underbrace{x_1 \wedge x_2 \wedge x_3}_{\neg P}$$

The formula is obviously unsatisfiable meaning there is no 0-counter-example, we continue by initializing $F_0 = I$

2. Step: First Transition
Check if $F_0 \wedge T \Rightarrow P'$, by testing if $F_0 \wedge T \wedge \neg P'$ is satisfiable:

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{F_0} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_1' \wedge x_2' \wedge x_3'}_{\neg P'}$$

Which it is not because $\neg x_1 \wedge (x_1 \vee \neg x_2') \wedge x_2'$ is unsatisfiable. We do not generate a proof-obligation so we can skip the blocking-phase and continue on with the propagation-phase.

3. Step: First Propagation-Phase
Initialize $F_1 = P$
Check each clause c in F_0 for $F_0 \wedge T \wedge \neg c'$ to strengthen F_1.

(a) $c = \neg x_1$

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{F_0} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_1'}_{\neg c'}$$

Satisfiable with $(\neg x_1, \neg x_2, \neg x_3, x_1', \neg x_2', \neg x_3')$
$\rightarrow$ Do not add $\neg x_1$ to F_1.

(b) $c = \neg x_2$

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{F_0} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_2'}_{\neg c'}$$

Unsatisfiable because $\neg x_1 \wedge (x_1 \vee \neg x_2') \wedge x_2'$ is not satisfiable
$\rightarrow$ Add $\neg x_2$ to F_1
$\rightarrow$ $F_1 = P \wedge \neg x_2$.

(c) $c = \neg x_3$

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{F_0} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_3'}_{\neg c'}$$

Unsatisfiable because $\neg x_2 \wedge (x_2 \vee \neg x_3') \wedge x_3'$ is not satisfiable
$\rightarrow$ Add $\neg x_3$ to F_1
$\rightarrow$ $F_1 = P \wedge \neg x_2 \wedge \neg x_3$

With that the first propagation-phase is done resulting in

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

and because $F_1 \not\equiv F_0$ we continue.

4. Step: Second Transition
Check if $F_1 \wedge T \Rightarrow P'$ by testing $F_1 \wedge T \wedge \neg P'$ for satisfiability:

$$\underbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_2 \wedge \neg x_3}_{F_1} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_1' \wedge x_2' \wedge x_3'}_{\neg P'}$$

Which is unsatisfiable because $\neg x_2 \wedge (x_2 \vee \neg x_3') \wedge x_3'$ is not satisfiable. We do not generate a proof-obligation so we continue with the second propagation-phase.

5. Step: Second Propagation-Phase
Initialize $F_2 = P$
Check each clause c in F_1 for $F_1 \wedge T \wedge \neg c'$ to strengthen F_2. We skip P, as it is already part of F_2.

This works exactly as in the 3. step:

(a) $c = \neg x_2$

$$\underbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_2 \wedge \neg x_3}_{F_1} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_2'}_{\neg c'}$$

Satisfiable with $(x_1, \neg x_2, \neg x_3, x_1', x_2', \neg x_3')$
$\rightarrow$ Do not add $\neg x_2$ to F_2

(b) $c = \neg x_3$

$$\underbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_2 \wedge \neg x_3}_{F_1} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_3'}_{\neg c'}$$

Unsatisfiable because $\neg x_2 \wedge (x_2 \vee \neg x_3') \wedge x_3'$ is not satisfiable.
$\rightarrow$ Add $\neg x_3$ to F_2
$\rightarrow F_2 = P \wedge \neg x_3$

That concludes the second propagation-phase resulting in

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

and because $F_2 \not\equiv F_1$ we continue.

6. Step: Third Transition Step
Check $F_2 \wedge T \Rightarrow \neg P'$ by testing $F_2 \wedge T \wedge \neg P'$ for satisfiability

$$\underbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_3}_{F_2} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_1' \wedge x_2' \wedge x_3'}_{\neg P'}$$

This time $F_2 \wedge T \wedge \neg P'$ is satisfiable with assignment $(\underbrace{x_1, x_2, \neg x_3}_{s}, x_1', x_2', x_3')$,

we get the new bad state $s = x_1 \wedge x_2 \wedge \neg x_3$, and generate a proof-obligation

$(s, 2)$, which we now try to block in the blocking-phase.

7. Step: First Blocking-Phase
Try to block proof-obligation $(s, 2)$ by checking if $F_1 \wedge T \wedge s'$ is satisfiable.

$$\underbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_2 \wedge \neg x_3}_{F_1} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_1' \wedge x_2' \wedge \neg x_3'}_{s'}$$

This is again satisfiable with assignment $(\underbrace{x_1, \neg x_2, \neg x_3}_{\text{q}}, x_1', x_2', \neg x_3')$, we get the bad state $q = x_1 \wedge \neg x_2 \wedge \neg x_3$ and generate a new proof-obligation $(q, 1)$.

Try to block proof-obligation $(q, 1)$ by checking if $F_0 \wedge T \wedge q'$ is satisfiable.

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{F_0} \wedge \underbrace{(x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \wedge (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3')}_{T} \wedge \underbrace{x_1' \wedge \neg x_2' \wedge \neg x_3'}_{q'}$$

This too is satisfiable with assignment $(\underbrace{\neg x_1, \neg x_2, \neg x_3}_{\text{I}}, x_1', \neg x_2', \neg x_3')$, we get the bad state $I = x_1 \wedge \neg x_2 \wedge \neg x_3$ and generate a new proof-obligation $(I, 0)$.

With that we have found a counter-example, resulting in the termination of the algorithm returning the counter-example trace:

$$\underbrace{\neg x_1 \wedge \neg x_2 \wedge \neg x_3}_{I} \to \underbrace{x_1 \wedge \neg x_2 \wedge \neg x_3}_{q} \to \underbrace{x_1 \wedge x_2 \wedge \neg x_3}_{s} \to \underbrace{x_1 \wedge x_2 \wedge x_3}_{\neg P}$$

Assume proof-obligation $(s, 2)$ would have been blocked, meaning $F_1 \wedge T \wedge s'$ was unsatisfiable, then we would have updated $F_2 = F_2 \wedge \neg s$ making absolutely sure that s is not reachable, every future proof-obligation containing s would have been blocked by F_2.

2.3.2 With Passing Property

To show a transition system with an inductive invariant consider $B = (X, I, T)$ with
$X = \{x_1, x_2\}$,
$I = \neg x_1 \wedge \neg x_2$,
$T = (x_1 \vee \neg x_2 \vee x_2') \wedge (x_1 \vee x_2 \vee \neg x_1') \wedge (\neg x_1 \vee x_1') \wedge (\neg x_1 \vee \neg x_2') \wedge (x_2 \vee \neg x_2')$
and transition graph:

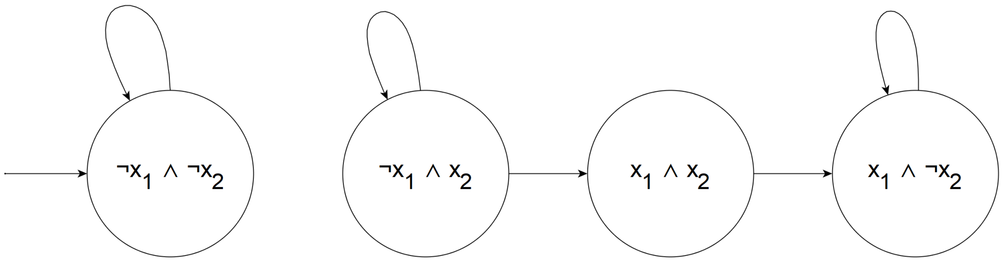


Figure 2: Transition Graph of B

Now given the property $P = \neg x_1 \vee x_2$, we want to check whether the bad state $\neg P = x_1 \wedge \neg x_2$ is reachable:

1. Step: Check for 0-Counter-Example
Check for 0-counter-example to make sure $I \Rightarrow P$ by testing $I \wedge \neg P$ for satisfiability:

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

The formula is unsatisfiable because $\neg x_1 \wedge x_1$ that means there is no 0-counter-example.

2. Step: First Transition
Initialize $F_0 = I$ and check if $F_0 \wedge T \Rightarrow P'$ by testing $F_0 \wedge T \wedge \neg P'$ for satisfiability:

$$\underbrace{\neg x_1 \wedge \neg x_2}_{F_0} \wedge \underbrace{(x_1 \vee \neg x_2 \vee x_2') \wedge (x_1 \vee x_2 \vee \neg x_1') \wedge (\neg x_1 \vee x_1') \wedge (\neg x_1 \vee \neg x_2') \wedge (x_2 \vee \neg x_2')}_{T} \wedge \underbrace{x_1' \wedge \neg x_2'}_{\neg P'}$$

Which is unsatisfiable because $\neg x_1 \wedge \neg x_2 \wedge (x_1 \vee x_2 \vee x_1') \wedge \neg x_1'$ is not satisfiable. We generate no proof-obligation and continue with the propagation-phase.

3. Step: First Propagation-Phase
Initialize $F_1 = P$
For each clause c in F_0 check $F_0 \wedge T \wedge \neg c'$ for satisfiability to strengthen F_1.

(a) $c = \neg x_1$

$$\neg x_1 \wedge \neg x_2 \wedge T \wedge x_1'$$

Unsatisfiable because $\neg x_1 \wedge \neg x_2 \wedge (x_1 \vee x_2 \vee x_1') \wedge \neg x_1'$ is not satisfiable.
$\rightarrow$ Add $\neg x_1$ to F_1
$\rightarrow F_1 = P \wedge \neg x_1$

(b) $c = \neg x_2$

$$\neg x_1 \wedge \neg x_2 \wedge T \wedge x_2'$$

Unsatisfiable because $\neg x_1 \wedge \neg x_2 \wedge (x_2 \vee \neg x_2') \wedge x_2'$ is not satisfiable.
$\rightarrow$ Add $\neg x_2$ to F_1
$\rightarrow F_1 = P \wedge \neg x_1 \wedge \neg x_2$

That concludes the propagation-phase resulting in

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

and because $F_1 \not\equiv F_0$ we continue.

4. Step: Second Transition
Check if $F_1 \wedge T \Rightarrow P'$ by testing $F_1 \wedge T \wedge \neg P'$ for satisfiability:

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

Which is unsatisfiable because $\neg x_1 \wedge \neg x_2 \wedge (x_1 \vee x_2 \vee \neg x_1') \wedge x_1'$ is not satisfiable. We again do not generate a proof-obligation, so that we continue with the second propagation-phase.

5. Step: Second Propagation-Phase
Initialize $F_2 = P$
For every clause c in F_1 check $F_1 \wedge T \wedge \neg c'$ for satisfiability, again skipping P.

(a) $c = \neg x_1$

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

Unsatisfiable because $\neg x_1 \wedge \neg x_2 \wedge (x_1 \vee x_2 \vee \neg x_1') \wedge x_1'$ is not satisfiable
$\rightarrow$ Add $\neg x_1$ to F_2
$\rightarrow F_2 = P \wedge \neg x_1$

(b) $c = \neg x_2$

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

Unsatisfiable because $\neg x_2 \wedge (x_2 \vee \neg x_2') \wedge x_2'$ is not satisfiable.
$\rightarrow$ Add $\neg x_2$ to F_2
$\rightarrow F_2 = P \wedge \neg x_1 \wedge \neg x_2$

With that the second propagation-phase ends, resulting in

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

The algorithm terminates returning that the property always holds and $(\neg x_1 \vee x_2) \wedge \neg x_1 \wedge \neg x_2$ being an inductive invariant.

3 Lifted PDR

We see that PDR is a useful hardware-model checking technique. If we want to use it on software we need to *lift* the algorithm from bit-level propositional logic to first-order logic. There are multiple ways to accomplish that, the following approach is based on the technique described in[6].
To use PDR on software we first need some new definitions an other preliminaries.

3.1 Preliminaries

A control flow graph (CFG) $\mathcal{A} = (X, L, G, \ell_0, \ell_E)$ is a tuple, consisting of a finite set of variables X, a finite set of locations L, a finite set of transitions $G \subseteq L \times FO \times L$, FO being a quantifier free first order logic formula over variables in X and $X' = \{x \in X \mid x' \in X'\}$, an initial location $\ell_0 \in L$, and an error location $\ell_E \in L$.

For example consider the CFG $(A) = (X, L, G, \ell_0, \ell_E)$ where $X = \{x\}, L = \{\ell_0, \ell_1, \ell_2, \ell_3, \ell_E\}, G = \{(\ell_0, x := 0, \ell_1), (\ell_1, x := x{+}1, \ell_2), (\ell_2, x = 1, \ell_E), (\ell_2, x \neq 1, \ell_3)\}$ with the graph:

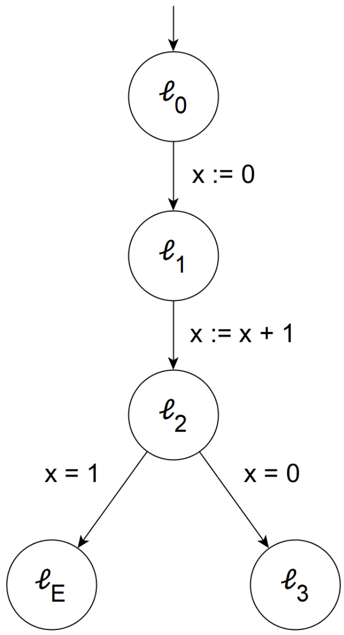


Figure 3: Graph of $\mathcal{A}$

The transition formula $T_{\ell_1 \to \ell_2}$ from one location ℓ_1 to another location ℓ_2 is defined as:

$$T_{\ell_1 \to \ell_2} = \begin{cases} (\ell_1, t, \ell_2), & (\ell_1, t, \ell_2) \in G \\ false, & otherwise \end{cases}$$

The lifted algorithm no longer works on boolean transition systems but on control flow graphs. It tries to verify ℓ_E is reachable by finding a feasible path from ℓ_0 to ℓ_E.

3.2 Lifted Algorithm

There are x main differences between bit-level PDR and lifted PDR:

- No longer blocking states but transition
  @ToDo: Better Explanation here
- Instead of a global set of Frames $[F_0, ..., F_k]$ assign each program location $\ell \in L \backslash \{\ell_E\}$ a local set of frames $[F_{0,\ell}, ..., F_{k,\ell}]$ which are now a cube of first-order formulas. Because of that proof-obligations get extended

by another parameter, lifted proof-obligations are tuples (t, ℓ, i), where t is a first-order formula, ℓ describes the location where t has to be blocked, and i is a frame number.

- Because of the structure of the CFA, it is already known which states lead to the error location, as it is easy to extract the transitions in G that have ℓ_E as target. Because of that the next transition phase, that was used to find proof-obligations before, is obsolete. If there exists a transition to ℓ_E there will be an initial proof-obligation in each iteration of the algorithm, which means that the blocking-phase can no longer be skipped.
- The propagation phase is slimmed, meaning that it only checks for termination by checking the frames of each location ℓ if $F_{i-1,l} = F_{i,\ell}$ for any $i \leq k$. There is no more propagating clauses.

In more detail:
Given a CFG $\mathcal{A} = (X, L, G, \ell_0, \ell_E)$ we want to check if ℓ_E is reachable:

Again start with checking for a 0-counter-example, this is easily done by checking if $\ell_0 = \ell_E$, if that is the case terminate and return that ℓ_E is indeed reachable, if not initialize level 0 frames for all locations $\ell \in L \backslash \{\ell_0, \ell_E\}$ as false. For ℓ_0 initialize it as true.

Let k be the current level, meaning each location $\ell \in L \backslash \{\ell_E\}$ has frames $[F_{0,\ell}, ..., F_{k,\ell}]$.
The algorithm repeats the following phases:

1. Next Level
Initialize for each $\ell \in L \backslash \{\ell_E\}$ a new frame $k + 1$ as true.
For each location $\ell \in L$ where $(\ell, t, \ell_E) \in G$ generate an initial proof-obligation (t, ℓ, k).

2. Blocking-Phase
If there are proof-obligations:
Take proof-obligation (ℓ, t, i) and check for each predecessor location ℓ_{pre} if the formula:

$$F_{i-1,\ell_{pre}} \wedge T_{\ell_{pre}\to\ell} \wedge t'$$

is satisfiable.

- If the formula is satisfiable, it means that t could not be blocked at ℓ on level i, generate an new proof-obligation $(p, \ell_{pre}, i-1)$ where p is the weakest precondition of t.
- If the formula is unsatisfiable, strengthen each frame $F_{j,\ell}$, $j \leq i$ with $\neg t$, meaning $F_{j,\ell} = F_{j,\ell} \wedge \neg t$, blocking t at ℓ on level i.

This continues recursively until either a proof-obligation $(d, \ell, 0)$ proving that there exists a feasible path to ℓ_E terminating the algorithm, or every proof-obligation is blocked.

3. Propagation-Phase
Check each $F_{i,\ell}$ if there exists an i where $F_{i,\ell} = F_{i-1,\ell} \neq$ true, if it does the algorithm terminates returning that ℓ_E is not reachable.

To illustrate the lifted algorithm further consider the updated pseudo-code:

@ToDo: Pseudocode

3.3 Example

3.3.1 Reachable Error State

@ToDo: Revise + Fix errors and notation
• To show an application of the lifted algorithm reconsider the example from earlier, we have CFA $\mathcal{A} = (X, L, G, \ell_0, \ell_E)$ with
$X = \{x\}$
$L = \{\ell_0, \ell_1, \ell_2, \ell_3, \ell_E\}$,
$G = \{(\ell_0, x := 0, \ell_1), (\ell_1, x := x+1, \ell_2), (\ell_2, x = 1, \ell_E), (\ell_2, x \neq 1, \ell_3)\}$

We now want to verify whether there exists a feasible trace from ℓ_0 to ℓ_E or not using the lifted algorithm:

1. Step: Check for 0-Counter-Example
Is $\ell_0 = \ell_E$?
No, it is not, we continue with initializing level 0 by adding to each $\ell \in L \backslash \{\ell_0, \ell_E\}$ a new frame $F_{0,\ell} = false$, for ℓ_0 add $F_{0,\ell_0} = true$.

location \ level	0
ℓ_0	true
ℓ_1	false
ℓ_2	false
ℓ_3	false

2. Step: Next Level
Initialize new frames for level 1 as true:

location \ level	0	1
ℓ_0	true	true
ℓ_1	false	true
ℓ_2	false	true
ℓ_3	false	true

To generate the initial proof-obligations, check G and take the transitions where ℓ_E is the target. There is one transition $(\ell_2, x = 1, \ell_E)$, that means we have to block $x = 1$ at ℓ_2 on level 1, we get proof-obligation $(x = 1, \ell_2, 1)$

3. Step: First Blocking Phase
We need to block the initial proof-obligation $(x = 1, \ell_2, 1)$. Let ℓ_{pre} be a predecessor of ℓ_2, we need to check the formula $F_{0,l_{pre}} \wedge T_{\ell_{pre} \to \ell_2} \wedge x' = 1$ for satisfiability. As there is only only predecessor ℓ_1 we test:

$$\underbrace{false}_{F_{0,\ell_1}} \wedge \underbrace{x' := x + 1}_{T_{\ell_1 \to \ell_2}} \wedge x' = 1$$

Which is obviously unsatisfiable, meaning we add $\neg(x = 1) \equiv x \neq 1$ to F_{0,ℓ_2} and F_{1,ℓ_2}, blocking $x = 1$ at ℓ_2 on level 1.

location \ level	0	1
ℓ_0	true	true
ℓ_1	false	true
ℓ_2	false $\wedge x \neq 1$	true $\wedge x \neq 1$
ℓ_3	false	true

Because there are no proof-obligations left we continue with the propagation-phase.

4. Step: First Propagation-Phase
Check all frames of location $\ell \in L$ if there exists an i so that $F_{i,\ell} = F_{i-1,\ell} \neq true$.
As there is none, we continue with the next level.

5. Step: Next Level Initialize new frames for level 2 as true:

location \ level	0	1	2
ℓ_0	true	true	true
ℓ_1	false	true	true
ℓ_2	false $\wedge x \neq 1$	true $\wedge x \neq 1$	true
ℓ_3	false	true	true

Again generate the initial proof-obligation which is the same as before but on level 2 now: $(x = 1, \ell_2, 2)$ and continue with the blocking-phase.

6. Step: Second-Blocking Phase

We need to block the proof-obligation $(x = 1, \ell_2, 2)$ by testing

$$\underbrace{true}_{F_{1,\ell_1}} \wedge \underbrace{x' := x + 1}_{T_{\ell_1 \to \ell_2}} \wedge x' = 1$$

for satisfiability. Which it is with $p : (x = 0)$, this is also the weakest precondition. We generate a new proof-obligation $(p, \ell_1, 1)$ meaning we need to block p at location ℓ_1 on level 1.
Take the new proof-obligation $(x = 0, \ell_1, 1)$ and check

$$\underbrace{true}_{F_{0,\ell_0}} \wedge \underbrace{x' := 0}_{T_{\ell_0 \to \ell_1}} \wedge \underbrace{x' = 0}_{p'}$$

for satisfiability.
Which is valid no matter what x is, we take $q : (x = 0)$ and generate the new proof-obligation $(x = 0, l_0, 0)$ and because this obligation has level 0 we terminate, stating that ℓ_E is reachable by the counter-example trace:

$$\ell_0 \to \ell_1 \to \ell_2 \to \ell_E$$

3.3.2 Unreachable Error State

@ToDo the whole thing

4 Goals

At the moment ULTIMATE uses interpolation based model-checkers. This bachelor's thesis aims at implementing a new PDR-based approach and then comparing it with the existing ones. Furthermore should the correctness of this new approach be tested by unit-tests.

5 Approach

A bachelor's thesis takes 12 weeks of work:

1. PREPLANNING: Deciding what classes are needed, which parts of the papers are to be implemented, and so on.

   Duration: 1 week

   Outcome: Rough plan on how to implement PDR in ULTIMATE.

2. IMPLEMENTING PDR: Implement the PDR-algorithm as described above with some changes to make it fit a software-verification role as detailed in [6].

   Duration: 4 weeks

   Outcome: A PDR based model-checking algorithm implemented in ULTIMATE.

3. IMPROVING PDR: Adding some perfomance improving techniques as described in [4], [5], and [6] to the PDR-algorithm.

   Duration: 2 weeks

   Outcome: An improved PDR-algorithm, tuned for better performance.

4. BUGFIXING: Finding and eliminating remaining bugs with help of unit tests.

   Duration: 1 week

   Outcome: A tested PDR-algorithm.

5. Analysis: Comparing the implementation with the existing model-checkers performance-wise.

   Duration: 1 week

   Outcome: Data comparing model-checking methods.

6. Writing the Thesis: Writing down my results in a thesis. Proof-reading and printing it.

   Duration: 3 weeks

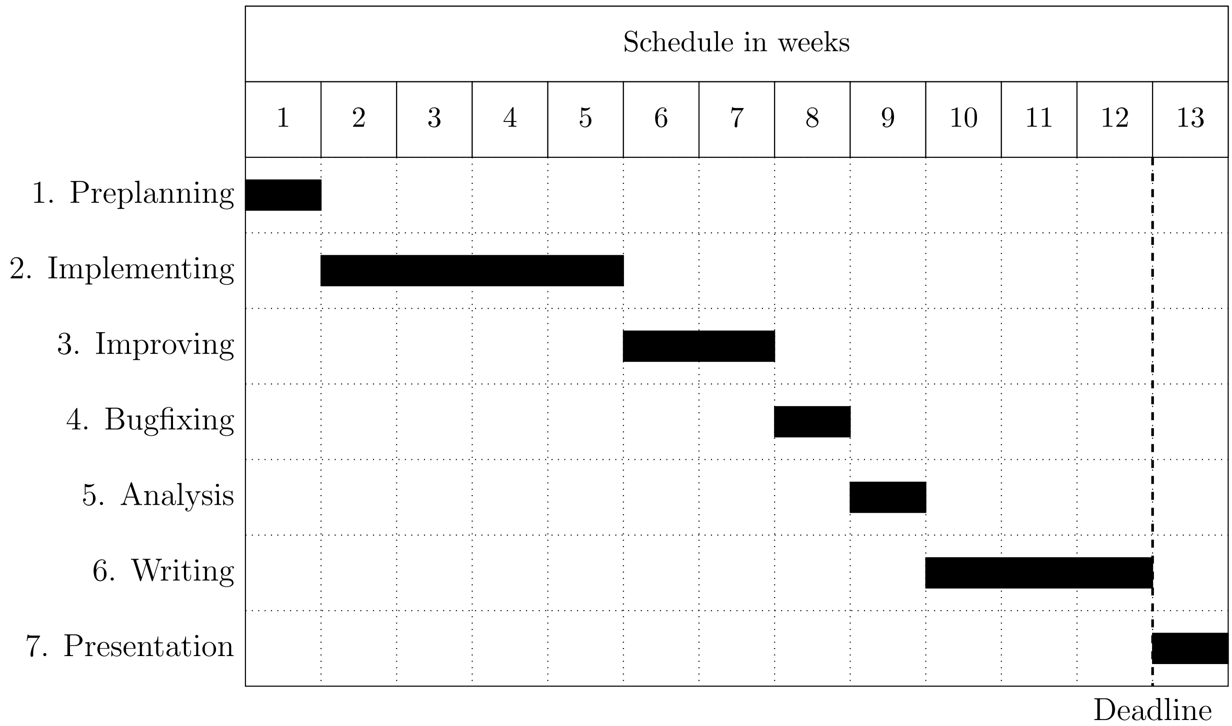
   Outcome: A Bachelor's thesis. Written and printed.

7. Preparing a final presentation: Preparing a presentation where I am able to show my results.

   Duration: 1 week *Note: can be finished after deadline*

   Outcome: A presentation of my results of the previous points.

6 Schedule



References

[1] Vizel Y., Gurfinkel A. (2014) Interpolating Property Directed Reachability. In: Biere A., Bloem R. (eds) Computer Aided Verification. CAV 2014. Lecture Notes in Computer Science, vol 8559. Springer, Cham

[2] Bradley A.R. (2011) SAT-Based Model Checking without Unrolling. In: Jhala R., Schmidt D. (eds) Verification, Model Checking, and Abstract Interpretation. VMCAI 2011. Lecture Notes in Computer Science, vol 6538. Springer, Berlin, Heidelberg

[3] Ultimate: `https://ultimate.informatik.uni-freiburg.de`

[4] N. Een, A. Mishchenko and R. Brayton, "Efficient implementation of property directed reachability," 2011 Formal Methods in Computer-Aided Design (FMCAD), Austin, TX, 2011, pp. 125-134. Reachability"

[5] Cimatti A., Griggio A. (2012) Software Model Checking via IC3. In: Madhusudan P., Seshia S.A. (eds) Computer Aided Verification. CAV 2012. Lecture Notes in Computer Science, vol 7358. Springer, Berlin, Heidelberg

[6] T. Lange, M. R. Neuhauber and T. Noll, "IC3 software model checking on control flow automata," 2015 Formal Methods in Computer-Aided Design (FMCAD), Austin, TX, 2015, pp. 97-104.

[7] Heizmann M., Hoenicke J., Podelski A. (2013) Software Model Checking for People Who Love Automata. In: Sharygina N., Veith H. (eds) Computer Aided Verification. CAV 2013. Lecture Notes in Computer Science, vol 8044. Springer, Berlin, Heidelberg

[8] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. 2002. Lazy abstraction. In Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '02). ACM, New York, NY, USA, 58-70