Implementing PDR in CPAchecker

Gernot Zorneck

Faculty of Computer Science and Mathematics
University of Passau

September 23, 2016
Outline

1. Introduction

2. Preliminaries

3. Original PDR
   - Concepts
   - Algorithm

4. PDR on Control Flow Automata: IC3CFA
   - Changes to standard PDR
   - Example
   - Implementation

5. Summary
Outline

1 Introduction

2 Preliminaries

3 Original PDR
   - Concepts
   - Algorithm

4 PDR on Control Flow Automata: IC3CFA
   - Changes to standard PDR
   - Example
   - Implementation

5 Summary
Original IC3

Overview

- IC3: Incremental Construction of Inductive Clauses for Indubitable Correctness
- Also known as PDR: Property Directed Reachability
- Aaron Bradley: “SAT-Based Model Checking without Unrolling”, VMCAI 2011
- Symbolic model checking algorithm for finite state systems (bit-level)
- Based on SAT solving, (relative) inductivity, backward analysis
- No unrolling of transition relation needed
- Highly incremental - lots of small SAT-queries
- Quickly became a staple part in most modern model checkers
- Adapted to infinite state systems such as software (C-programs, . . . )
Inductive Strengthening

- Property is inductive $\Rightarrow$ property is invariant
- **But**: Not every invariant property can be proved by induction
Inductive Strengthening

- Property is inductive $\Rightarrow$ property is invariant
- But: Not every invariant property can be proved by induction
- Idea: Strengthen property
- Math example:
  \[
  \sum_{i=1}^{n} \frac{1}{i^2} \leq 2 \quad \text{vs.} \quad \sum_{i=1}^{n} \frac{1}{i^2} \leq 2 - \frac{1}{n}
  \]
Inductive Strengthening

- Property is inductive $\implies$ property is invariant
- **But** : Not every invariant property can be proved by induction
- Idea : Strengthen property
- Math example : $\sum_{i=1}^{n} \frac{1}{i^2} \leq 2$ vs. $\sum_{i=1}^{n} \frac{1}{i^2} \leq 2 - \frac{1}{n}$
- Plan : Create strengthening of property and prove it by induction
- This will prove the property
Outline

1 Introduction

2 Preliminaries

3 Original PDR
   - Concepts
   - Algorithm

4 PDR on Control Flow Automata: IC3CFA
   - Changes to standard PDR
   - Example
   - Implementation

5 Summary
Literal/Clause/Cube

- A **literal** is a propositional variable or its negation ($x, \neg y, \ldots$)
- A **clause** is a disjunction of literals ($x \lor \neg y$)
- A **cube** is a conjunction of literals ($x \land \neg y$)
- Therefore, the negation of a cube is a clause ($\neg (x \land \neg y) \equiv (\neg x \lor y)$)

Transition System

A **Transition System** $S : (\bar{x}, I(\bar{x}), T(\bar{x}, \bar{x}'))$ consists of

- a set $\bar{x}$ of state variables
- the initial configuration of the system $I(\bar{x})$
- the transition relation $T(\bar{x}, \bar{x}')$
### Preliminaries - Cont.

#### (Relative) Inductivity

Given a transition system $S : (\bar{x}, I(\bar{x}), T(\bar{x}, \bar{x}'))$:

- $P$ is inductive, if $I \Rightarrow P$ and $P \land T \Rightarrow P'$
- $P$ is inductive *relative* to $F$, if $I \Rightarrow P$ and $F \land P \land T \Rightarrow P'$

#### Safety property: $P$

A boolean formula that is always true for a given transition system

#### Inductive Strengthening

An inductive strengthening of a safety property $P$ is a formula $F$, so that $F \land P$ is inductive
### State
Assignment of values to **all** state variables of the transition system. Represented by a cube

### Control Flow Automaton (CFA)
A *Control flow automaton* $A = \{ L, G, l_0, l_E \}$ consists of
- a set of locations $L = \{0, \ldots, n\}$ representing the program counter
- edges from $G \subseteq L \times QFFO \times L$ labeled with quantifier-free first order formulas describing the transition
- an initial location $l_0$
- an error location $l_E$
Outline

1. Introduction
2. Preliminaries
3. Original PDR
   - Concepts
   - Algorithm
4. PDR on Control Flow Automata: IC3CFA
   - Changes to standard PDR
   - Example
   - Implementation
5. Summary
Original PDR

General Concepts

Frame : $F_i$
- Overapproximation of reachable states in at most $i$ steps from initial states
- Set of clauses (act as constraints regarding reachability)
- As formula : conjunction of clauses (CNF)

Counterexample to Inductiveness : CTI
State that can reach a non-property state (bad state) in one or more steps
Original PDR

General Concepts

Frame : $F_i$
- Overapproximation of reachable states in at most $i$ steps from initial states
- Set of clauses (act as constraints regarding reachability)
- As formula: conjunction of clauses (CNF)

Counterexample to Inductiveness : CTI
State that can reach a non-property state (bad state) in one or more steps

- Maintain series of stepwise overapproximations $F_0, \ldots, F_k$ for increasing $k$
- $F_0 = I$ and initially $F_i = P$ for $i \neq 0$ (assume $P$ is invariant)
- Continuously refine frames by adding reachability information
- Derived from recursively backward-analyzing CTIs
Original PDR

General Concepts

Basic Invariants

- $F_0 \Leftrightarrow I$
- $F_i \Rightarrow P, \quad \forall 0 \leq i \leq k$ - “every frame satisfies $P$”
- $F_i \Rightarrow F_{i+1}, \quad \forall 0 \leq i < k$ - “every $F_{i+1}$ is more general than $F_i$”
  \[ \text{clauses}(F_{i+1}) \subseteq \text{clauses}(F_i) \]
- $F_i \land T \Rightarrow F'_{i+1}, \quad \forall 0 \leq i < k$ - “states in $F_i$ transition to states in $F_{i+1}$”
Original PDR

General Concepts

Basic Invariants

- $F_0 \iff I$
- $F_i \Rightarrow P, \quad \forall 0 \leq i \leq k$ - “every frame satisfies $P$”
- $F_i \Rightarrow F_{i+1}, \quad \forall 0 \leq i < k$ - “every $F_{i+1}$ is more general than $F_i$”
  \[
  \text{clauses}(F_{i+1}) \subseteq \text{clauses}(F_i)
  \]
- $F_i \land T \Rightarrow F'_{i+1}, \quad \forall 0 \leq i < k$ - “states in $F_i$ transition to states in $F_{i+1}$”

⇒ Fixpoint reached if $\exists i$ so that $F_i = F_{i+1}$
⇒ Property holds
⇒ $F_i$ is an inductive strengthening of the safety property $P$
PDR: Identify CTIs
Algorithm

check for 0-/1-step counterexample \((I \land \neg P / I \land T \land \neg P')\)

for \(k = 1\) to 

- while (CTI exists \(\equiv F_k \land P \land T \Rightarrow P'\) not true)
  - get CTI \(s\) from model
    - % Blocking Phase %
    - prove \(s\) is unreachable in \(\leq k\) steps
      (this is where new clauses are learned)
    - if not possible \(\rightarrow\) error found

- % Propagation Phase %
- for \(i = 1\) to \(k\) and all clauses \(c\) in \(F_i\)
  - if \(c\) became inductive
    \(\equiv F_i \land c \land T \Rightarrow c'\) is true : add \(c\) to \(F_{i+1}\)
  - if \(\exists\ i\) so that \(F_i = F_{i+1}\) \(\rightarrow\) property holds
Blocking a state \( s \) at \( F_i \) : Proof Obligation \((s, i)\)

**Block state \( s \) at \( F_i \) \equiv Prove \( s \) is unreachable in \( \leq i \) steps**

- If \( i = 0 \) and \( s \) intersects with initial states \( \rightarrow \) error found
- Check: \( \neg s \) inductive relative to \( F_{i-1} \equiv F_{i-1} \land \neg s \land T \Rightarrow \neg s' \) is true
- No: try to block predecessor \( p \) of \( s \) at \( F_{i-1} \) first (DFS). Add **Proof Obligations** \((p, i - 1)\) and \((s, i)\)
- Yes: add \( \neg s \) to all frames \( F_1, \ldots, F_i \). Also add PO \((s, i + 1)\) if \( i < k \)
- Pick PO with lowest frame number next
- Retry previously failed attempts until \( s \) could be blocked at \( F_i \)
Algorithm : Important Improvements

Generalization

- Blocking one state \( s \) at a time is ineffective
- When adding \( \neg s \) at level \( i \) : find \( c \subseteq \neg s \) that is still inductive and add \( c \) instead
- \( c \) may exclude many more states than \( \neg s \)  
  \[ (\neg x \lor y) \rightarrow \neg x \]  
- Drop literals that don’t actually contribute to result of induction query  
  \[ F_{i-1} \land \neg s \land T \Rightarrow \neg s' \]  
- Use unsat-core, ternary simulation, . . .
Algorithm : Important Improvements

Lifting

- Similar intention as with generalization
- When computing a predecessor $p$ of state $s$: find set of states that also transition to $s$
- Represented by a sub-cube of $p$

Subsumption

- Suppose $F_i = \{s\}$ with $s = x \lor y$ and we can add $\hat{s} = x$
- Note that $\hat{s} \Rightarrow s$ (or alternatively $\text{literals}(\hat{s}) \subset \text{literals}(s)$)
- $s$ doesn’t represent more reachability info than $\hat{s}$
- Simply remove $s$
- Avoids redundancy and keeps frames small (easier SAT-queries)
Outline

1. Introduction

2. Preliminaries

3. Original PDR
   - Concepts
   - Algorithm

4. PDR on Control Flow Automata : IC3CFA
   - Changes to standard PDR
   - Example
   - Implementation

5. Summary
PDR on Control Flow Automata

Based on “IC3 Software Model Checking on Control Flow Automata” by T. Lange et al.

- Apply PDR directly to CFA
- Use SMT-solver instead of SAT-solver
- Check reachability of error location
- Use single transitions between locations (no unrolling needed)
- Create frames $F_0, \ldots, F_k$ for every location
- Represents $k$-step reachability for this location, starting at $l_0$
PDR on Control Flow Automata

Based on “IC3 Software Model Checking on Control Flow Automata” by T. Lange et al.

### PDR Relative Inductivity Check

When trying to block a state $s$ at level $i \equiv (s,i)$

- $F_{i-1} \land \neg s \land T \land s'$  
  (meaning: $F_{i-1} \land \neg s \land T \Rightarrow \neg s'$)

### Adjusted Relative Inductivity Check

When trying to block a state $s$ at location $l$ at level $i \equiv (s,l,i)$

- **Case 1**: $F_{i-1,l_{\text{pred}}} \land T_{l_{\text{pred}} \rightarrow l} \land s'$, if $l \neq l_{\text{pred}}$

- **Case 2**: $F_{i-1,l_{\text{pred}}} \land \neg s \land T_{l_{\text{pred}} \rightarrow l} \land s'$, if $l = l_{\text{pred}}$

where $l_{\text{pred}}$ is a predecessor location of $l$.

- Unsat $\rightarrow$ add $\neg s$ to all $F_{j,l}$ where $j \leq i$

- Sat $\rightarrow$ get predecessor state $p$ and add POs $(p, l_{\text{pred}}, i - 1)$ and $(s, l, i)$
IC3CFA

Example

Initialization

- No 0-/1-step counterexamples
- $F_{i,l} = true$, for all locations $l$ and levels $i$ (we have no known safety property!)
- Except: $F_{0,l} = false$, for all non-initial locations

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_0$</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$l_1$</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
IC3CFA Example

First iteration: $k = 1$

- Transition still possible?
- $F_{l_0, l_1} \land T_{l_1 \rightarrow l_E} = true \land x = 1 \implies SAT$
- $\rightarrow x = 1$

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_0$</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$l_1$</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
**IC3CFA**

Example

First iteration: $k = 1$

- Try to block $x = 1$ at $l_1$ at level 1
- **Predecessor** $l_0$:
  - $F_{0,l_0} \land T_{l_0 \rightarrow l_1} \land s' =
    \begin{align*}
    true \land x' = 0 \land x' = 1 : & \text{ UNSAT} \\
    \rightarrow \text{ add } x \neq 1 \text{ to } F_{1,l_1} \text{ and } F_{0,l_1} 
    \end{align*}$

---

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_0$</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$l_1$</td>
<td>false, (x \neq 1)</td>
<td>true, (x \neq 1)</td>
<td>true</td>
</tr>
</tbody>
</table>

---
**IC3CFA Example**

First iteration: $k = 1$

- Try to block $x = 1$ at $l_1$ at level 1
- **Predecessor** $l_1$:
  - $F_{0,l_1} \land \neg s \land T_{l_1 \rightarrow l_1} \land s' = (false \land x \neq 1) \land x \neq 1 \land (x \neq 1 \land x' = x + 1) \land x' = 1 : UNSAT$
  - $\rightarrow$ add $x \neq 1$ to $F_{1,l_1}$ and $F_{0,l_1}$

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_0$</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$l_1$</td>
<td>false, $x \neq 1$</td>
<td>true, $x \neq 1$</td>
<td>true</td>
</tr>
</tbody>
</table>
**IC3CFA**

**Example**

First iteration: \( k = 1 \)

- Transition still possible?
- \( F_{1,l_1} \land T_{l_1 \rightarrow l_E} = (true \land x \neq 1) \land x = 1 : UNSAT \)
- Termination? → No
- → continue with next iteration

```latex
\begin{tabular}{|c|c|c|c|}
\hline
loc / lvl & 0 & 1 & 2 \\
\hline
\hline
l_0 & true & true & true \\
\hline
l_1 & false, \( x \neq 1 \) & true, \( x \neq 1 \) & true \\
\hline
\end{tabular}
```
IC3CFA

Example

Second iteration: $k = 2$

- Transition still possible?
- $F_{2,l_1} \land T_{l_1 \rightarrow l_E} =$
  
  $true \land x = 1 : SAT$

- $\rightarrow x = 1$

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_0$</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$l_1$</td>
<td>false, $x \neq 1$</td>
<td>true, $x \neq 1$</td>
<td>true</td>
</tr>
</tbody>
</table>
IC3CFA Example

Second iteration : $k = 2$

- Try to block $x = 1$ at $l_1$ at level 2
- **Predecessor** $l_0$ :
  - $F_{1,l_0} \land T_{l_0 \rightarrow l_1} \land s' =
    \text{true} \land x' = 0 \land x' = 1 : \text{UNSAT}$
  - $\rightarrow$ add $x \neq 1$ to $F_{2,l_1}$ and $F_{1,l_1}$ and $F_{0,l_1}$

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_0$</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$l_1$</td>
<td>false, $x \neq 1$</td>
<td>true, $x \neq 1$</td>
<td>true, $x \neq 1$</td>
</tr>
</tbody>
</table>
IC3CFA

Example

Second iteration : $k = 2$

- Try to block $x = 1$ at $l_1$ at level 2
- **Predecessor $l_1$**:
  - $F_{l_1,l_1} \land \neg s \land T_{l_1 \rightarrow l_1} \land s' = (true \land x \neq 1) \land x \neq 1 \land (x \neq 1 \land x' = x + 1) \land x' = 1 : SAT \rightarrow x = 0$
- Proof-obligations : $(1, l_1, x = 0), (2, l_1, x = 1)$

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_0$</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>$l_1$</td>
<td>false, $x \neq 1$, true, $x \neq 1$</td>
<td>true, $x \neq 1$</td>
<td>true, $x \neq 1$</td>
</tr>
</tbody>
</table>
IC3CFA

Example

Second iteration : \( k = 2 \)

- Pick lowest Proof-obligation \((1, l_1, x = 0)\)
- **Predecessor** \( l_0 \):
  - \( F_{0,l_0} \land T_{l_0 \rightarrow l_1} \land s' = true \land x' = 0 \land x' = 0 : SAT \rightarrow x = 0 \)
  - Proof-obligations : \((0, l_0, x = 0), (1, l_1, x = 0), (2, l_1, x = 1)\)
  - Next : \((0, l_0, x = 0) \rightarrow \text{Error found !}\)

<table>
<thead>
<tr>
<th>loc / lvl</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_0 )</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>( l_1 )</td>
<td>false, ( x \neq 1 )</td>
<td>true, ( x \neq 1 )</td>
<td>true, ( x \neq 1 )</td>
</tr>
</tbody>
</table>
Remark: Dealing with infinite state space

Weakest Preconditions

- Use weakest preconditions on local transitions to calculate exact predecessors
- Can be expensive for large transitions
Remark: Dealing with infinite state space

Weakest Preconditions

- Use weakest preconditions on local transitions to calculate exact predecessors
- Can be expensive for large transitions

Predicate Abstraction

- Get concrete predecessors from model of SAT-query (like original PDR)
- Apply predicate abstraction and work with abstract states
- Random example: \((x = 0 \land y = 0) \rightarrow x = y\)
- When finding abstract transition with no concrete counterpart
  \(\rightarrow\) abstraction was too broad
  \(\rightarrow\) interpolate and refine abstraction \((x = y \rightarrow (x = y \land x \geq 0))\)
- Similar to CTIGAR
Implementation in CPAchecker

Transitions

- CPAchecker can be configured to arbitrary block size
- Large Block Encoding currently used for PDR
- PredicateCPA used to get path formulas of edges between locations

Predicate Abstraction

- Component PredicateAbstractionManager of PredicateCPA used for computing abstraction based on current predicates
- SMT-solver used to get interpolant that leads to new abstraction predicate
Outline

1. Introduction
2. Preliminaries
3. Original PDR
   - Concepts
   - Algorithm
4. PDR on Control Flow Automata : IC3CFA
   - Changes to standard PDR
   - Example
   - Implementation
5. Summary
Summary

- PDR is a symbolic model checking algorithm for finite state systems based on SAT-solving, relative inductiveness, inductive strengthening.
- Blocking phase: Identify CTI and recursively block it.
- Propagation phase: Push clauses to next frame if they became inductive after blocking phase.
- PDR can be extended to infinite state systems in multiple ways.
  - One way: Apply PDR directly to CFA (IC3CFA).
  - Give every location its own set of stepwise overapproximations (frames).
  - Check reachability of error location using single transitions between locations.
## Outlook

### What still needs to be done
- Predicate abstraction
- Check if it pairs well with location local frames

### For the future
- One prover environment for each frame
- Keep frame clauses on prover stack (exploit incremental nature of PDR)
- Parallel implementation (PDR is suitable for this)