IC3, PDR, and Friends

Arie Gurfinkel
ariecmu.edu

Abstract. We describe the IC3/PDR algorithms and their various generalizations. Our goal is to give a brief overview of the algorithms and describe them using unified notation. Many crucial optimizations and implementation details are omitted.

1 Constrained Horn Clauses

Given the sets $\mathcal{F}$ of function symbols, $\mathcal{P}$ of predicate symbols, and $\mathcal{V}$ of variables, a Constrained Horn Clause (CHC) is a First Order Logic (FOL) formula of the form:

$$\forall V \cdot (\phi \land p_1[X_1] \land \cdots \land p_k[X_k] \rightarrow h[X]), \text{ for } k \geq 0$$

where: $\phi$ is a constraint over $\mathcal{F}$ and $\mathcal{V}$ with respect to some background theory $\mathcal{A}$; $X_i, X \subseteq \mathcal{V}$ are (possibly empty) vectors of variables; $p_i[X_i]$ is an application $p(t_1,\ldots,t_n)$ of an $n$-ary predicate symbol $p \in \mathcal{P}$ for first-order terms $t_i$ constructed from $\mathcal{F}$ and $X_i$; and $h[X]$ is either defined analogously to $p_i$ or is $\mathcal{P}$-free (i.e., no $\mathcal{P}$ symbols occur in $h$). Here, $h$ is called the head of the clause and $\phi \land p_1[X_1] \land \cdots \land p_k[X_k]$ is called the body. A clause is called a query if its head is $\mathcal{P}$-free, and otherwise, it is called a rule. A rule with body true is called a fact. We say a clause is linear if its body contains at most one predicate symbol, otherwise, it is called non-linear. In this paper, we follow the Constraint Logic Programming (CLP) convention of representing Horn clauses as $h[X] \leftarrow \phi, p_1[X_1], \ldots, p_k[X_k]$.

A CHC with constraint $\phi$ is satisfiable if there exists an interpretation $\mathcal{I}$ of the predicate symbols $\mathcal{P}$ such that each constraint $\phi$ is true under $\mathcal{I}$. A set $\Pi$ of CHCs is satisfiable if there exists an interpretation $\mathcal{I}$ that satisfies all clauses in $\Pi$.

Satisfiability of a set $\Pi$ of linear CHC is reducible to satisfiability of 3 clauses of the form:

$$\text{Init}(X) \rightarrow P(X) \quad (1)$$

$$P(X) \rightarrow \text{Bad}(X) \quad (2)$$

$$P(X) \land \text{Tr}(X, X') \rightarrow P(X') \quad (3)$$
where $X$ is a set of variables, $X' = \{x' \mid x \in X\}$, $P$ is a new predicate, and $Init$, $Tr$, and $Bad$ are constraints. We call this reduced problem Safety, and present it as a triple $\langle Init, Tr, Bad \rangle$.

Satisfiability of a set $\Pi$ of non-linear CHC is reducible to satisfiability of 3 clauses of the form:

\[
\begin{align*}
Init(X) & \rightarrow P(X) \quad (4) \\
P(X) & \rightarrow Bad(X) \quad (5) \\
P(X) \land P(X^o) \land Tr(X, X^o, X') & \rightarrow P(X') \quad (6)
\end{align*}
\]

where, $X^o = \{x^o \mid x \in X\}$ and the rest is defined as before. We call this reduced problem Safety as well and present it as a triple $\langle Init, Tr, Bad \rangle$. Note that the only difference between the linear and non-linear case is that $Tr$ depends on two sets of state-variables: $X$ and $X^o$.

## 2 IC3 and PDR

The finite state model checking algorithm IC3 was introduced in [2] and its variant PDR in [3]. It maintains sets of clauses $F_0, \ldots, F_i, \ldots, F_N$, called a trace, that are properties of states reachable in $i$ steps from the initial states $Init$. Elements of $F_i$ are called lemmas. In the following, we assume that $F_0$ is initialized to $Init$. After establishing that $Init \rightarrow \neg Bad$, the algorithm maintains the following invariants (for $0 \leq i < N$):

**Input:** A safety problem $\langle Init(X), Tr(X, X'), Bad(X) \rangle$.

**Output:** Unreachable or Reachable

**Data:** A cex queue $Q$, where $c \in Q$ is a pair $\langle m, i \rangle$, $m$ is a cube over state variables, and $i \in \mathbb{N}$. A level $N$. A trace $F_0, F_1, \ldots$

Initially: $Q = \emptyset$, $N = 0$, $F_0 = Init$, $\forall i > 0 \cdot F_i = \emptyset$.

repeat

1. **Unreachable** If there is an $i < N$ s.t. $F_{i+1} \subseteq F_i$, return Unreachable.
2. **Reachable** If there is an $m$ s.t. $\langle m, 0 \rangle \in Q$ return Reachable.
3. **Unfold** If $F_N \rightarrow \neg Bad$, then set $N \leftarrow N + 1$.
4. **Candidate** If for some $m$, $m \rightarrow F_N \land Bad$, then add $\langle m, N \rangle$ to $Q$.
5. **Decide** If $\langle m, i \rangle$ is in $Q$ and there are $m_0$ and $m_1$ s.t. $m_1 \rightarrow m$, $m_0 \land m_1 \rightarrow F_i \land Tr \land m'$, then add $\langle m_0, i \rangle$ to $Q$.
6. **Conflict** For $0 \leq i < N$: given a candidate model $\langle m, i \rangle \in Q$ and clause $\varphi$, such that $\varphi \rightarrow \neg m$, if $Init \rightarrow \varphi$, and $\varphi \land F_i \land Tr \rightarrow \varphi'$, then add $\varphi$ to $F_j$, for $j \leq i + 1$.
7. **Leaf** If $\langle m, i \rangle \in Q$, $0 < i < N$ and $F_{i-1} \land Tr \land m'$ is unsatisfiable, then add $\langle m, i + 1 \rangle$ to $Q$.
8. **Induction** For $0 \leq i < N$ and a clause $(\varphi \lor \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $Init \rightarrow \varphi$ and $\varphi \land F_i \land Tr \rightarrow \varphi'$, then add $\varphi$ to $F_j$, for each $j \leq i + 1$.

until $\infty$;

**Algorithm 1:** IC3/PDR.
Invariant 1

\[ F_i \rightarrow \neg \text{Bad} \quad F_i \rightarrow F_{i+1} \quad F_i \land Tr \rightarrow F_{i+1}' \]

That is, each \( F_i \) is safe, the trace is monotone, and \( F_{i+1} \) is inductive relative to \( F_i \). In practice, the algorithm enforces monotonicity by maintaining \( F_{i+1} \subseteq F_i \).

Alg. 1 summarizes, in a simplified form, a variant of the IC3 algorithm. The algorithm maintains a queue of counter-examples \( Q \). Each element of \( Q \) is a tuple \( \langle m, i \rangle \) where \( m \) is a monomial over \( v \) and \( 0 \leq i \leq N \). Intuitively, \( \langle m, i \rangle \) means that a state \( m \) can reach a state in \( \text{Bad} \) in \( N - i \) steps. Initially, \( Q \) is empty, \( N = 0 \) and \( F_0 = \text{Init} \). Then, the rules are applied (possibly in a non-deterministic order) until either \textbf{Unreachable} or \textbf{Reachable} rule is applicable.

\textbf{Unfold} rules extends the current trace and increases the level at which counterexample is searched. \textbf{Candidate} picks a set of bad states. \textbf{Decide} extends a counter-example from the queue by one step. \textbf{Conflict} blocks a counterexample and adds a new lemma. \textbf{Leaf} moves the counterexample to the next level. Finally, \textbf{Induction} generalizes a lemma inductively. A typical schedule of the rules is to first apply all applicable rules except for \textbf{Induction} and \textbf{Unfold}, followed by \textbf{Induction} at all levels, then \textbf{Unfold}, and then repeating the cycle.

\textit{Queue}. The queue is ordered by the level:

\[ \langle m, i \rangle < \langle n, j \rangle \iff i < j \quad (7) \]

This drives the algorithm to the shortest counterexample.

\textit{Inductive Generalization}. The \textbf{Conflict} and \textbf{Induction} rules are based on the principle of inductive generalization. Let \( F_0, \ldots, F_i, \ldots, F_N \) be a valid trace, and let \( \varphi \) be a clause that is relatively inductive to \( F_i \):

\[ \text{Init} \implies \varphi \quad \varphi \land F_i \land Tr \implies \varphi' \quad (8) \]

Let \( G = G_0, \ldots, G_N \) be defined as follows:

\[ G_j = \begin{cases} F_j \cup \{ \varphi \} & \text{if } j \leq i + 1 \\ F_j & \text{if } i + 1 < j \leq N \end{cases} \quad (9) \]

Then \( G \) is a valid trace. The proof is by induction on \( i \) and follows from monotonicity of the trace.

\textit{Generalizing predecessors}. The \textbf{Decide} rule picks a predecessor \( m_0 \) in \( Tr \) of some (partial) state \( m \). While it is possible to simply pick a predecessor state, the rule attempts to find a generalized predecessor instead. The conditions of the rule is sufficient to ensure that \( m_0 \) is an implicant of \( \psi = (F_i \land \exists X' \cdot (Tr \land m')) \).

Finding a prime implicant of \( \psi \) would have been even better, but is too expensive in practice.
Input: A safety problem \((\text{Init}(X), \text{Tr}(X, X'), \text{Bad}(X))\).

Output: Unreachable or Reachable

Data: A cex queue \(Q\), where a cex \(c \in Q\) is a pair \((m, i)\), \(m\) is a conjunction of constraints over state variables, and \(i \in \mathbb{N}\). A trace \(F_0, F_1, \ldots\)

Notation: \(\mathcal{F}(A) = (A(X) \land \text{Tr}) \lor \text{Init}(X')\).

All rules of IC3/PDR from Alg. 1, with \textbf{Decide} and \textbf{Conflict} replaced by the following:

\textbf{Decide} If \((P, i + 1) \in Q\) and there is a model \(m(X, X')\) s.t. \(m \models \mathcal{F}(F) \land P'\), add \((P', i)\) to \(Q\), where \(P' = \text{MBP}(X', m, \mathcal{F}(F)) \land P'\).

\textbf{Conflict} For \(0 \leq i < N\), given a counterexample \((P, i + 1) \in Q\) s.t. \(\mathcal{F}(F) \land P'\) is unsatisfiable, add \(P^\uparrow = \text{ITP}(\mathcal{F}(F)(X_0, X), P)\) to \(F\) for \(j \leq i + 1\).

Algorithm 2: \textbf{APDR}.

Propagating lemmas. The \textbf{Induction} rule propagates lemmas to higher level, optionally generalizing them as possible. This makes the trace “more” inductive, eventually leading to convergence.

Long counterexamples. The \textbf{Leaf} rule lifts blocked counterexamples to higher levels. As a side-effect, it makes it possible to discover counterexamples longer than the current exploration bound \(N\). For example, assume that \(m\) is blocked at level \(i\). This means that there is a path of length \(N - i\) from \(m\) to \(\text{Bad}\) (but no path of length at most \(i\) from \text{Init} to \(m\)). Assume that \textbf{Leaf} lifted \(m\) to level \(j > i\), and then \(m\) was reachable from \text{Init}. Then, the discovered counterexample is a concatenation of a path of length \(k\) from \text{Init} to \(m\) and a path of length \(N - i\) from \(m\) to \(\text{Bad}\). The total length of the counterexample is \((N - i + k)\) which is bigger than \(N\).

3 Extending IC3/PDR to Theories

Extending IC3 to theories (such as Linear Arithmetic) requires changing \textbf{Decide} and \textbf{Conflict} rules to the ones shown in Alg. 2 [1]. The \textbf{Decide} rule computes a predecessor using an under-approximation of existential quantifier elimination called \textbf{Model Based Projection (MBP)}. The \textbf{Conflict} computes new lemmas using \textbf{Craig Interpolation (ITP)}. Note that \textbf{Conflict} no longer based on the principle of inductive generalization. In the following, we briefly define MBP and ITP.

\textit{Model Based Projection.} Let \(\varphi\) be a formula, \(U \subseteq Vars(\varphi)\) a subset of variables of \(\varphi\), and \(P\) a model of \(\varphi\). Then, \(\psi = \text{MBP}(U, P, \varphi)\) is a model based projection if \(a\) \(\psi\) is a monomial, \(b\) \(Vars(\psi) \subseteq Vars(\varphi) \setminus U\), \(c\) \(P \models \psi\), \(d\) \(\psi \Rightarrow \exists \mathbf{V} \cdot \varphi\). Furthermore, for a fixed \(U\) and a fixed \(\varphi\), MBP is finite. In [5], an MBP function is defined for LRA based on Loos-Weispfenning quantifier elimination. Note that finiteness of MBP ensures that \textbf{Decide} can only be applied finitely many times for a fixed set of lemmas \(F\).
Thus, it is worst case exponential even for CHC over propositional constraints. 

Decide alternating between based on projection, GPDR is incomplete for LRA. That is, it might get stuck is via the use of interpolation in the rule. However, since GPDR is based on projection, GPDR is incomplete for LRA. That is, it might get stuck alternating between Decide and Conflict rules, never making progress.

Craig Interpolation. Given two formulas and such that is unsatisfiable, a Craig interpolant is a formula such that and . We further require that the interpolant is a clause. An algorithm for extracting LRA clause interpolants from the theory lemmas produced during DPLL(T) proof is given in [4].

4 Generalized PDR

GPDR algorithm [4] shown in Alg. 3 extends IC3/PDR to non-linear CHC and to constraints over Linear Rational Arithmetic (LRA). The main difference is that each element of the queue is a tuple of counterexamples. Intuitively, the tuple corresponds to leaves of a counterexample tree. Each application of the Decide rule expands one leaf of a counterexample. The extension to Linear Arithmetic is via the use of interpolation in the Conflict rule. However, since GPDR is based on projection, GPDR is incomplete for LRA. That is, it might get stuck alternating between Decide and Conflict rules, never making progress.

This version of GPDR does not cache reachability information. Hence, it might need to expand the derivation tree completely to find a counterexample. Thus, it is worst case exponential even for CHC over propositional constraints.
Successor uses reachability cache to compute a new reachable state.
uses reachability cache to skip over right-most predicate application.
cideMust
DecideMust
for linear CHC since reachability is known before the Reach rule checks whether a Reachable state became reachable. This is inefficient for linear CHC since reachability is known before the Reach set is computed.

The single Decide rule of APDR is replaced by three rules: Successor, DecideMust, and DecideMay. DecideMay is most similar to Decide. DecideMust uses reachability cache to skip over right-most predicate application. Successor uses reachability cache to compute a new reachable state.

For linear CHC, Spacer is equivalent to APDR.

5 Spacer

Spacer [5], shown in Alg. 4 extends APDR to non-linear CHC. Unlike other variants of IC3/PDR discussed so far, it maintains the set of reachable states Reach. This set is used, among other things, to cache reachability information.

We briefly outline the key difference between Spacer and APDR. First, the Reachable rule checks whether a Bad state became reachable. This is inefficient for linear CHC since reachability is known before the Reach set is computed.

Initially: $Q = \emptyset$, $N = 0$, $F_0 = \text{Init}$, $\forall i > 0 \cdot F_i = \emptyset$, Reach = Init

Require: $\text{Init} \rightarrow \neg \text{Bad}$
repeat
Unreachable If there is an $i < N$ s.t. $F_{i+1} \subseteq F_i$ return Unreachable.
Reachable If Reach $\land$ Bad is satisfiable, return Reachable.
Unfold If $F_N \rightarrow \neg \text{Bad}$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.
Candidate If for some $m$, $m \rightarrow F_N \land \text{Bad}$, then add $\langle m, N \rangle$ to $Q$.
Successor If there is $\langle m, i + 1 \rangle \in Q$ and a model $M \models \psi$, where $\psi = F(\lor \text{Reach}) \land m'$. Then, add $s$ to Reach, where $s' \in \text{MBP}(\{X, X'^o\}, \psi)$.
DecideMust If there is $\langle m, i + 1 \rangle \in Q$, and a model $M \models \psi$, where $\psi = F(F_i, \lor \text{Reach}) \land m'$. Then, add $s$ to $Q$, where $s \in \text{MBP}(\{X^o, X'\}, \psi)$.
DecideMay If there is $\langle m, i + 1 \rangle \in Q$ and a model $M \models \psi$, where $\psi = F(F_i) \land m'$. Then, add $s$ to $Q$, where $s' \in \text{MBP}(\{X, X'\}, \psi)$.
Conflict If there is an $\langle m, i + 1 \rangle \in Q$, s.t. $F(F_i) \land m'$ is unsatisfiable. Then, add $\varphi = \text{ITP}(F(F_i), m')$ to $F_j$, for all $0 \leq j \leq i + 1$.
Leaf If $\langle m, i \rangle \in Q$, $0 < i < N$ and $F(F_{i-1}) \land m'$ is unsatisfiable, then add $\langle m, i + 1 \rangle$ to $Q$.
Induction For $0 \leq i < N$ and a clause $(\varphi \lor \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $F(\varphi \land F_i) \rightarrow \varphi'$, then add $\varphi$ to $F_j$, for all $j \leq i + 1$.

until $\infty$;

Algorithm 4: Rule-based description of Spacer.

Input: A safety problem $(\text{Init}(X), \text{Tr}(X, X^o, X'), \text{Bad}(X))$.
Output: Unreachable or Reachable
Data: A cex queue $Q$, where a cex $c \in Q$ is a pair $\langle m, i \rangle$, $m$ is a cube over state variables, and $i \in \mathbb{N}$. A level $N$. A set of reachable states Reach. A trace $F_0, F_1, \ldots$
Notation: $F(A, B) = \text{Init}(X^o) \lor (A(X) \land B(X^o) \land \text{Tr})$, and $F(A) = F(A, A)$

Spacer 5 Spacer
References


