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

$$\forall V \cdot (\phi \land p_1[X_1] \land \ldots \land p_k[X_k] \rightarrow h[X])$$

for $k \geq 0$

where: $\phi$ is a constraint over $F$ and $V$ with respect to some background theory $A$; $X_i, X \subseteq V$ 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 $p \in P$ for first-order terms $t_i$ constructed from $F$ and $X_i$; and $h[X]$ is either defined analogously to $p_i$ or is $P$-free (i.e., no $P$ symbols occur in $h$). Here, $h$ is called the head of the clause and $\phi \land p_1[X_1] \land \ldots \land p_k[X_k]$ is called the body. A clause is called a query if its head is $P$-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 $I$ of the predicate symbols $P$ such that each constraint $\phi$ is true under $I$. A set $\Pi$ of CHCs is satisfiable if there exists an interpretation $I$ that satisfies all clauses in $\Pi$.

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

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

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

$$P(X) \land \text{Tr}(X, X') \rightarrow P(X') \quad (3)$$

* Copyright 2015 Carnegie Mellon University This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN AS-IS BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. This material has been approved for public release and unlimited distribution. DA-0002438
Input: A safety problem \((\text{Init}(X), \text{Tr}(X,X'), \text{Bad}(X))\).

Output: \text{Unreachable} or \text{Reachable}

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

Initially: \(Q = \emptyset, N = 0, F_0 = \text{Init}, \forall i > 0 \cdot F_i = \emptyset\).

repeat
\begin{align*}
\text{Unreachable} & \quad \text{If there is an } i < N \text{ s.t. } F_{i+1} \subseteq F_i \text{ return Unreachable.} \\
\text{Reachable} & \quad \text{If there is an } m \text{ s.t. } (m,0) \in Q \text{ return Reachable.} \\
\text{Candidate} & \quad \text{If for some } m, m \to F_N \land \text{Bad}, \text{ then add } (m,N) \text{ to } Q. \\
\text{Decide} & \quad \text{If } (m,i+1) \in Q \text{ and there are } m_0 \text{ and } m_1 \text{ s.t. } m_1 \to m, m_0 \land m_1' \text{ is} \\
& \quad \text{satisfiable, and } m_0 \land m_1' \to F_i \land \text{Tr} \land m', \text{ then add } (m_0,m') \text{ to } Q. \\
\text{Conflict} & \quad \text{For } 0 \leq i < N \text{: given a candidate model } (m,i+1) \in Q \text{ and clause } \varphi, \text{ such} \\
& \quad \text{that } \varphi \to \neg m, \text{ if } \text{Init} \to \varphi, \text{ and } \varphi \land F_i \land \text{Tr} \to \varphi', \text{ then add } \varphi \text{ to } F_i, \text{ for } j \leq i + 1. \\
\text{Leaf} & \quad \text{If } (m,i) \in Q, 0 < i < N \text{ and } F_{i-1} \land \text{Tr} \land m' \text{ is unsatisfiable, then add } (m,i+1) \\
& \quad \text{to } Q. \\
\text{Induction} & \quad \text{For } 0 \leq i < N \text{ and a clause } (\varphi \lor \psi) \in F_i, \text{ if } \varphi \notin F_{i+1}, \text{ Init} \to \varphi \text{ and} \\
& \quad \varphi \land F_i \land \text{Tr} \to \varphi', \text{ then add } \varphi \text{ to } F_i, \text{ for each } j \leq i + 1.
\end{align*}

until \(\infty\);

**Algorithm 1: IC3/PDR.**

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

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

\begin{align*}
\text{Init}(X) \to P(X) \quad \text{(4)} \\
P(X) \to \text{Bad}(X) \quad \text{(5)} \\
P(X) \land P(X^o) \land \text{Tr}(X,X^o,X') \to P(X') \quad \text{(6)}
\end{align*}

where, \(X^o = \{x^o \mid x \in X\}\) and the rest is defined as before. We call this reduced problem \text{Safety} as well and present it as a triple \((\text{Init}, \text{Tr}, \text{Bad})\). Note that the only difference between the linear and non-linear case is that \(\text{Tr}\) 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 \textit{trace}, that are properties of states reachable in \(i\) steps from the initial states \(\text{Init}\). Elements of \(F_i\) are called \textit{lemmas}. In the following, we assume that \(F_0\) is initialized to \(\text{Init}\). After establishing that \(\text{Init} \to \neg \text{Bad}\), the algorithm maintains the following invariants (for \(0 \leq i < N\)):
Invariant 1

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

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

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.

Inductive Generalization. The Conflict and 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 \( F_i \):

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

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.

Generalizing predecessors. The Decide rule picks a predecessor \( m_0 \) in \( \text{Tr} \) 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 \( \psi = (F_i \land \exists X' \cdot (\text{Tr} \land m')) \). 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 level \(N\). 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 Decide and Conflict replaced by the following:

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

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

Algorithm 2: APDR.

**Propagating lemmas.** The 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 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 Leaf lifted \(m\) to level \(j > i\), and then \(m\) was reachable from 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 Decide and Conflict rules to the ones shown in Alg. 2 [1]. The Decide rule computes a predecessor using an under-approximation of existential quantifier elimination called Model Based Projection (MBP). The Conflict computes new lemmas using Craig Interpolation (ITP). Note that Conflict no longer based on the principle of inductive generalization. In the following, we briefly define MBP and ITP.

**Model Based Projection.** Let \(\varphi\) be a formula, \(U \subseteq \text{Vars}(\varphi)\) 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) \(\text{Vars}(\psi) \subseteq \text{Vars}(\varphi) \setminus U\), (c) \(P \models \psi\), (d) \(\psi \rightarrow \exists \mathcal{V} \cdot \varphi\). 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 Decide can only be applied finitely many times for a fixed set of lemmas \(F_j\).
Thus, it is worst case exponential even for CHC over propositional constraints. It might need to expand the derivation tree completely to find a counterexample.

Alternating between projection, GPDR is incomplete for LRA. That is, it might get stuck Conflict is via the use of interpolation in the rule. However, since Decide 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.

Craig Interpolation. Given two formulas \( A[x, z] \) and \( B[y, z] \) such that \( A \land B \) is unsatisfiable, a Craig interpolant \( I[z] = \text{Itp}(A[x, z], B[y, z]) \), is a formula such that \( A[x, z] \rightarrow I[z] \) and \( I[z] \rightarrow \neg B[y, z] \). 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 \( Q \) 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 Decide 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.

**Input:** A safety problem \((\text{Init}(X), Tr(X, X', X'), \text{Bad}(X))\).

**Output:** Unreachable or Reachable

**Data:** A cex queue \( Q \), where a cex \( (c_0, \ldots, c_k) \in Q \) is a tuple, each \( c_j = (m, i) \),

\( m \) is a cube over state variables, and \( i \in \mathbb{N} \). A level \( N \). A trace \( F_0, F_1, \ldots \)

**Notation:** \( F(A, B) = \text{Init}(X') \lor (A(X) \land B(X'') \land Tr) \), and \( F(A) = F(A, A) \)

**Initially:** \( Q = \emptyset \), \( N = 0 \), \( F_0 = \text{Init} \), \( \forall i > 0 \cdot F_i = \emptyset \)

**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 exists \( t \in Q \) s.t. for all \( (c, i) \in t \), \( i = 0 \), return Reachable.

Candidate If for some \( m, m \rightarrow F_N \land \text{Bad} \), then add \( \langle (m, N) \rangle \) to \( Q \).

Decide If there is a \( t \in Q \), with \( c = (m, i + 1) \in t \), \( m_0 \rightarrow m \land m_0'' \land m_1' \) is satisfiable, and \( l_0 \land m_0'' \land m_1' \rightarrow F_i \land F_i'' \land Tr \land m' \) then add \( t \) to \( Q \), where \( i = t \) with \( c \) replaced by two tuples \( (l_0, i), \) and \( (m_0, i) \).

Conflict If there is a \( t \in Q \) with \( c = (m, i + 1) \in t \), 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 there is \( t \in Q \) with \( c = (m, i) \in t \), \( 0 < i < N \) and \( F(F_{i-1}) \land m' \) is unsatisfiable, then add \( t \) to \( Q \), where \( i = t \) with \( c \) replaced by \( (m, i + 1) \).

Induction For \( 0 \leq i < N \) and a clause \( (\varphi \lor \psi) \in F_i \), if \( \varphi \notin F_{i+1} \), \( \mathcal{F}(\varphi \land F_i) \rightarrow \varphi' \), then add \( \varphi \) to \( F_j \), for all \( j \leq i + 1 \).

until \( \infty \);

**Algorithm 3:** GPDR.
**Input:** A safety problem \((\text{Init}(X), \text{Tr}(X, X_0, X'), \text{Bad}(X))\).

**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 \(\text{REACH}\). A trace \(F_0, F_1, \ldots\)

**Notation:** \(\mathcal{F}(A, B) = \text{Init}(X') \vee (A(X) \land B(X_0) \land \text{Tr})\), and \(\mathcal{F}(A) = \mathcal{F}(A, A)\)

**Initially:** \(Q = \emptyset\), \(N = 0\), \(F_0 = \text{Init}\), \(\forall i > 0 \cdot F_i = \emptyset\), \(\text{REACH} = \text{Init}\)

**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 \(\text{REACH} \land \text{Bad}\) 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 
  \[\psi = \mathcal{F}(\vee \text{REACH}) \land m'.\] Then, add \(s\) to \(\text{REACH}\), where \(s' \in \text{MBP}\{X, X_0\}, \psi\).
- **DecideMust** If there is \(\langle m, i + 1 \rangle \in Q\) and a model \(M \models \psi\), where 
  \[\psi = \mathcal{F}(F_i) \land m'.\] Then, add \(s\) to \(Q\), where \(s' \in \text{MBP}\{X, X', \psi\}\).
- **DecideMay** If there is \(\langle m, i + 1 \rangle \in Q\) and a model \(M \models \psi\), where 
  \[\psi = \mathcal{F}(F_i) \land m'.\] 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. \(\mathcal{F}(F_i) \land m'\) is unsatisfiable. Then, add 
  \(\varphi = \text{ITP}(\mathcal{F}(F_i), m')\) to \(F_j\), for all \(0 < j \leq i + 1\).
- **Leaf** If \(\langle m, i \rangle \in Q\), \(0 < i < N\) and \(\mathcal{F}(F_{i-1}) \land m'\) 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 \(\varphi \not\in F_{i+1}\), \(\mathcal{F}(\varphi \land F_i) \rightarrow \phi'\), 
  then add \(\varphi\) to \(F_j\), for all \(j \leq i + 1\).

until \(\infty\);

**Algorithm 4:** Rule-based description of Spacer.

### 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 \(\text{REACH}\). 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.

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**.
References


