IC3, PDR, and Friends

Arie Gurfinkel

Department of Electrical and Computer Engineering
University of Waterloo
arie.gurfinkel@uwaterloo.ca

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:

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

where: ϕ 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 $\phi \wedge p_1[X_1] \wedge \ldots \wedge p_k[X_k]$ 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 ϕ is satisfiable if there exists an interpretation $\mathcal{I}$ of the predicate symbols $\mathcal{P}$ such that each constraint ϕ is true under $\mathcal{I}$. A set Π of CHCs is satisfiable if there exists an interpretation $\mathcal{I}$ that satisfies all clauses in Π.

Satisfiability of a set Π of linear CHC is reducible to satisfiability of 3 clauses of the form:

$$Init(X) \rightarrow P(X) \tag{1}$$

$$P(X) \rightarrow Bad(X) \tag{2}$$

$$P(X) \wedge Tr(X, X') \rightarrow P(X') \tag{3}$$

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 $\langle Init, Tr, Bad \rangle$.

Input: A safety problem $\langle \mathit{Init}(X), \mathit{Tr}(X, X'), \mathit{Bad}(X) \rangle$.
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: $Q = \emptyset$, $N = 0$, $F_0 = \mathit{Init}$, $\forall i > 0 \cdot F_i = \emptyset$.
repeat

Unreachable If there is an $i < N$ s.t. $F_{i+1} \subseteq F_i$ **return** *Unreachable*.

Reachable If there is an m s.t. $\langle m, 0 \rangle \in Q$ **return** *Reachable*.

Unfold If $F_N \to \neg \mathit{Bad}$, then set $N \leftarrow N + 1$.

Candidate If for some m, $m \to F_N \wedge \mathit{Bad}$, then add $\langle m, N \rangle$ to Q.

Predecessor If $\langle m, i+1 \rangle \in Q$ and there are m_0 and m_1 s.t. $m_1 \to m$, $m_0 \wedge m_1'$ is satisfiable, and $m_0 \wedge m_1' \to F_i \wedge \mathit{Tr} \wedge m'$, then add $\langle m_0, i \rangle$ to Q.

NewLemma For $0 \leq i < N$: given a candidate model $\langle m, i+1 \rangle \in Q$ and clause φ, such that $\varphi \to \neg m$, if $\mathit{Init} \to \varphi$, and $\varphi \wedge F_i \wedge \mathit{Tr} \to \varphi'$, then add φ to F_j, for $j \leq i + 1$.

ReQueue If $\langle m, i \rangle \in Q$, $0 < i < N$ and $F_{i-1} \wedge \mathit{Tr} \wedge m'$ is unsatisfiable, then add $\langle m, i+1 \rangle$ to Q.

Push For $0 \leq i < N$ and a clause $(\varphi \vee \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $\mathit{Init} \to \varphi$ and $\varphi \wedge F_i \wedge \mathit{Tr} \to \varphi'$, then add φ to F_j, for each $j \leq i + 1$.

until ∞;

Algorithm 1: IC3/PDR.

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

$$\mathit{Init}(X) \to P(X) \tag{4}$$

$$P(X) \to \mathit{Bad}(X) \tag{5}$$

$$P(X) \wedge P(X^o) \wedge \mathit{Tr}(X, X^o, X') \to P(X') \tag{6}$$

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 $\langle \mathit{Init}, \mathit{Tr}, \mathit{Bad} \rangle$. 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 $\mathit{Init} \to \neg \mathit{Bad}$, the algorithm maintains the following invariants (for $0 \leq i < N$):

Invariant 1

$$F_i \to \neg Bad \qquad\qquad F_i \to F_{i+1} \qquad\qquad F_i \wedge Tr \to F'_{i+1}$$

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 $\boldsymbol{v}$ and $0 \leq i \leq N$. Intuitively, $\langle m, i\rangle$ means that a state m can reach a state in Bad in $N - i$ steps. Initially, Q is empty, $N = 0$ and $F_0 = Init$. 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. **Predecessor** extends a counter-example from the queue by one step. **NewLemma** blocks a counterexample and adds a new lemma. **ReQueue** moves the counterexample to the next level. Finally, **Push** generalizes a lemma inductively. A typical schedule of the rules is to first apply all applicable rules except for **Push** and **Unfold**, followed by **Push** at all levels, then **Unfold**, and then repeating the cycle.

Queue. The queue is ordered by the level:

$$\langle m, i\rangle < \langle n, j\rangle \iff i < j \tag{7}$$

This drives the algorithm to the shortest counterexample.

Inductive Generalization. The **NewLemma** and **Push** rules are based on the principle of inductive generalization. Let $F_0, \ldots, F_i, \ldots, F_N$ be a valid trace, and let φ be a clause that is *relatively inductive* to F_i:

$$Init \implies \varphi \qquad\qquad \varphi \wedge F_i \wedge Tr \implies \varphi' \tag{8}$$

Let $\boldsymbol{G} = G_0, \ldots, G_N$ be defined as follows:

$$G_j = \begin{cases} F_j \cup \{\varphi\} & \text{if } j \leq i+1 \\ F_j & \text{if } i+1 < j \leq N \end{cases} \tag{9}$$

Then $\boldsymbol{G}$ is a valid trace. The proof is by induction on i and follows from monotonicity of the trace.

Generalizing predecessors. The **Predecessor** 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 $\psi = (F_i \wedge \exists X' \cdot (Tr \wedge m'))$. Finding a prime implicant of ψ would have been even better, but is too expensive in practice.

Input: A safety problem $\langle Init(X), Tr(X, X'), Bad(X)\rangle$.
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 conjunction of constraints over state variables, and $i \in \mathbb{N}$. A level N. A trace $F_0, F_1, \ldots$
Notation: $\mathcal{F}(A) = (A(X) \wedge Tr) \vee Init(X')$.

All rules of IC3/PDR from Alg. 1, with **Predecessor** and **NewLemma** replaced by the following:

Predecessor If $\langle P, i+1\rangle \in Q$ and there is a model $m(X, X')$ s.t. $m \models \mathcal{F}(F_i) \wedge P'$, add $\langle P_\downarrow, i\rangle$ to Q, where $P_\downarrow = \textsc{Mbp}(X', m, \mathcal{F}(F_i) \wedge P')$.

NewLemma for $0 \leq i < n$, given a counterexample $\langle P, i+1\rangle \in Q$ s.t. $\mathcal{F}(F_i) \wedge P'$ is unsatisfiable, add $P^\uparrow = \textsc{Itp}(\mathcal{F}(F_i), P')$ to F_j for $j \leq i+1$.

Algorithm 2: $\mathcal{A}$PDR.

Propagating lemmas. The **Push** rule propagates lemmas to higher level, optionally generalizing them as possible. This makes the trace "more" inductive, eventually leading to convergence.

Long counterexamples. The **ReQueue** 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 *Bad* (but no path of length at most i from *Init* to m). Assume that **ReQueue** 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 *Init* to m and a path of length $N - i$ from m to *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 **Predecessor** and **NewLemma** rules to the ones shown in Alg. 2 [1]. The **Predecessor** rule computes a predecessor using an under-approximation of existential quantifier elimination called *Model Based Projection (MBP)*. The **NewLemma** computes new lemmas using *Craig Interpolation (ITP)*. Note that **NewLemma** no longer based on the principle of inductive generalization. In the following, we briefly define MBP and ITP.

Model Based Projection. Let φ be a formula, $U \subseteq Vars(\varphi)$ a subset of variables of φ, and P a model of φ. Then, $\psi = \textsc{Mbp}(U, P, \varphi)$ is a model based projection if (a) ψ is a monomial, (b) $Vars(\psi) \subseteq Vars(\varphi) \setminus U$, (c) $P \models \psi$, (d) $\psi \rightarrow \exists V \cdot \varphi$. Furthermore, for a fixed U and a fixed φ, $\textsc{Mbp}$ is finite. In [6], an MBP function is defined for LRA based on Loos-Weispfenning quantifier elimination. Note that finiteness of MBP ensures that **Predecessor** can only be applied finitely many times for a fixed set of lemmas F_i.

Input: A safety problem $\langle Init(X), Tr(X, X^o, X'), Bad(X)\rangle$.
Output: *Unreachable* or *Reachable*
Data: A cex queue Q, where a cex $\langle c_0, \ldots, c_k\rangle \in Q$ is a tuple, each $c_j = \langle m, i\rangle$, m is a cube over state variables, and $i \in \mathbb{N}$. A level N. A trace $F_0, F_1, \ldots$
Notation: $\mathcal{F}(A, B) = Init(X') \vee (A(X) \wedge B(X^o) \wedge Tr)$, and $\mathcal{F}(A) = \mathcal{F}(A, A)$
Initially: $Q = \emptyset$, $N = 0$, $F_0 = Init$, $\forall i > 0 \cdot F_i = \emptyset$
Require: $Init \rightarrow \neg Bad$
repeat

Unreachable If there is an $i < N$ s.t. $F_i \subseteq F_{i+1}$ **return** *Unreachable*.

Reachable if exists $t \in Q$ s.t. for all $\langle c, i\rangle \in t$, $i = 0$, **return** *Reachable*.

Unfold If $F_N \rightarrow \neg Bad$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.

Candidate If for some m, $m \rightarrow F_N \wedge Bad$, then add $\langle\langle m, N\rangle\rangle$ to Q.

Predecessor If there is a $t \in Q$, with $c = \langle m, i+1\rangle \in t$, $m_1 \rightarrow m$, $l_0 \wedge m_0^o \wedge m_1'$ is satisfiable, and $l_0 \wedge m_0^o \wedge m_1' \rightarrow F_i \wedge F_i^o \wedge Tr \wedge m'$ then add $\hat{t}$ to Q, where $\hat{t} = t$ with c replaced by two tuples $\langle l_0, i\rangle$, and $\langle m_0, i\rangle$.

NewLemma If there is a $t \in Q$ with $c = \langle m, i+1\rangle \in t$, s.t. $\mathcal{F}(F_i) \wedge m'$ is unsatisfiable. Then, add $\varphi = \textsc{Itp}(\mathcal{F}(F_i), m')$ to F_j, for all $0 \leq j \leq i + 1$.

ReQueue If there is $t \in Q$ with $c = \langle m, i\rangle \in t$, $0 < i < N$ and $\mathcal{F}(F_{i-1}) \wedge m'$ is unsatisfiable, then add $\hat{t}$ to Q, where $\hat{t}$ is t with c replaced by $\langle m, i+1\rangle$.

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

until ∞;

Algorithm 3: GPDR.

Craig Interpolation. Given two formulas $A[\boldsymbol{x}, \boldsymbol{z}]$ and $B[\boldsymbol{y}, \boldsymbol{z}]$ such that $A \wedge B$ is unsatisfiable, a Craig interpolant $I[\boldsymbol{z}] = \textsc{Itp}(A[\boldsymbol{x}, \boldsymbol{z}], B[\boldsymbol{y}, \boldsymbol{z}])$, is a formula such that $A[\boldsymbol{x}, \boldsymbol{z}] \rightarrow I[\boldsymbol{z}]$ and $I[\boldsymbol{z}] \rightarrow \neg B[\boldsymbol{y}, \boldsymbol{z}]$. 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 [5].

4 Generalized PDR

GPDR algorithm [5] 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 leafs of a counterexample tree. Each application of the **Predecessor** rule expands one leaf of a counterexample. The extension to Linear Arithmetic is via the use of interpolation in the **NewLemma** rule. However, since **Predecessor** is based on projection, GPDR is incomplete for LRA. That is, it might get stuck alternating between **Predecessor** and **NewLemma** rules, never making progress.

Input: A safety problem $\langle Init(X), Tr(X, X^o, X'), Bad(X) \rangle$.
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: $\mathcal{F}(A, B) = Init(X') \vee (A(X) \wedge B(X^o) \wedge Tr)$, and $\mathcal{F}(A) = \mathcal{F}(A, A)$
Initially: $Q = \emptyset$, $N = 0$, $F_0 = Init$, $\forall i > 0 \cdot F_i = \emptyset$, REACH $= Init$
Require: $Init \rightarrow \neg Bad$
repeat

Unreachable If there is an $i < N$ s.t. $F_i \subseteq F_{i+1}$ **return** *Unreachable.*

Reachable If REACH $\wedge$ Bad is satisfiable, **return** *Reachable.*

Unfold If $F_N \rightarrow \neg Bad$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.

Candidate If for some m, $m \rightarrow F_N \wedge Bad$, then add $\langle m, N \rangle$ to Q.

Successor If there is $\langle m, i+1 \rangle \in Q$ and a model M s.t. $M \models \psi$, where $\psi = \mathcal{F}(\vee \text{REACH}) \wedge m'$. Then, add s to REACH, where $s' \in \text{MBP}(\{X, X^o\}, \psi)$.

MustPredecessor If there is $\langle m, i+1 \rangle \in Q$, and a model M s.t. $M \models \psi$, where $\psi = \mathcal{F}(F_i, \vee \text{REACH}) \wedge m'$. Then, add s to Q, where $s \in \text{MBP}(\{X^o, X'\}, \psi)$.

MayPredecessor If there is $\langle m, i+1 \rangle \in Q$ and a model M s.t. $M \models \psi$, where $\psi = \mathcal{F}(F_i) \wedge m'$. Then, add s to Q, where $s^o \in \text{MBP}(\{X, X'\}, \psi)$.

NewLemma If there is an $\langle m, i+1 \rangle \in Q$, s.t. $\mathcal{F}(F_i) \wedge m'$ is unsatisfiable. Then, add $\varphi = \text{ITP}(\mathcal{F}(F_i), m')$ to F_j, for all $0 \leq j \leq i+1$.

ReQueue If $\langle m, i \rangle \in Q$, $0 < i < N$ and $\mathcal{F}(F_{i-1}) \wedge m'$ is unsatisfiable, then add $\langle m, i+1 \rangle$ to Q.

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

until ∞;

Algorithm 4: Rule-based description of SPACER.

This version of GPDR does not cache reachability information. Hence, it might need to expand the derivation tree completely to find a a counterexample. Thus, it is worst case exponential even for CHC over propositional constraints.

5 Spacer

SPACER [6], shown in Alg. 4 extends $\mathcal{A}$PDR 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 $\mathcal{A}$PDR. 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 $\mathcal{A}$PDR is replaced by three rules: **Successor**, **MustPredecessor**, and **MayPredecessor**. **MayPredecessor** is most similar to **Decide**. **MustPredecessor** uses reachability cache to skip over rightmost predicate application. **Successor** uses reachability cache to compute a new reachable state.

For linear CHC, Spacer is equivalent to $\mathcal{A}$PDR.

Data: A cex queue Q, where $c \in Q$ is a triple $\langle m, i, t\rangle$, m is a cube over state variables, $i \in \mathbb{N}$, and $t \in \{may, must\}$. A level N. A trace $F_0, F_1, \ldots$ An invariant F_∞. A set of reachable states Reach.

Initially: $Q = \emptyset$, $N = 0$, $\textsc{Reach} = F_0 = \mathit{Init}$, $\forall i \geq 1 \cdot F_i = \top$, $F_\infty = \top$.

Require: $\mathit{Init} \to \neg \mathit{Bad}$

repeat

- **Unreachable** If $F_\infty \to \neg \mathit{Bad}$
  - **return** *Unreachable.*
- **Reachable** If $\langle m, i, must\rangle \in Q$, $m \cap (\vee \textsc{Reach}) \neq \emptyset$
  - **return** *Reachable.*
- **Unfold** If $F_N \to \neg \mathit{Bad}$, then set $N \leftarrow N + 1$.
- **Candidate** If for some m, $m \to F_N \wedge \mathit{Bad}$, then add $\langle m, N, must\rangle$ to Q.
- **Predecessor** If $\langle m, i+1, t\rangle \in Q$ and there are
  m_0 and m_1 s.t. $m_1 \to m$, $m_0 \wedge m_1'$ is satisfiable,
  and $m_0 \wedge m_1' \to F_i \wedge \mathit{Tr} \wedge m'$,
  then add $\langle m_0, i, t\rangle$ to Q.
- **NewLemma** For $0 \leq i < N$: given $\langle m, i+1\rangle \in Q$ and a clause φ, such that
  $\varphi \to \neg m$,
  if $(\vee \textsc{Reach}) \to \varphi$, and $\varphi \wedge F_i \wedge \mathit{Tr} \to \varphi'$, then
  add φ to F_j, for $j \leq i+1$.
- **ReQueue** If $\langle m, i, must\rangle \in Q$, and $F_{i-1} \wedge \mathit{Tr} \wedge m'$ is unsatisfiable, then add
  $\langle m, i+1, must\rangle$ to Q.
- **Push** For $1 \leq i$ and a clause $(\varphi \vee \psi) \in F_i \setminus F_{i+1}$,
  if $(\vee \textsc{Reach}) \to \varphi$ and $\varphi \wedge F_i \wedge \mathit{Tr} \to \varphi'$, then
  add φ to F_j, for each $j \leq i+1$.
- **MaxIndSubset** If there is $i > N$ s.t. $F_{i+1} \subseteq F_i$, then
  $F_\infty \leftarrow F_i$, and $\forall j \geq i \cdot F_j \leftarrow F_\infty$.
- **Successor** If $\langle m, i+1, t\rangle \in Q$ and exist m_0, m_1 s.t.
  $m_0 \wedge m_1'$ are satisfiable and $m_0 \wedge m_1' \to (\vee \textsc{Reach}) \wedge \mathit{Tr} \wedge m'$, then
  add m_1 to Reach.
- **MayEnqueue** For $i \geq 1$ and a clause $\varphi \in F_i \setminus F_{i+1}$,
  if $(\vee \textsc{Reach}) \to \varphi$, add $\langle \neg\varphi, i+1, may\rangle \in Q$.
- **ResetQ** $Q \leftarrow \emptyset$.
- **ResetReach** $\textsc{Reach} \leftarrow \mathit{Init}$.

until ∞;

Algorithm 5: Rule-based description of Quip.

6 Quip

Quip [4], shown in Alg. 5 extends IC3/PDR with *may* and *must* proof obligation. Similarly to Spacer, it maintains a set of reachable states. The two key differences are the introduction of **MaxIndSubset** and **MayEnqueue** rules.

The **MaxIndSubset** rule identifies when a frame i represents an inductive (but possibly not safe) invariant. The set of all such invariants is kept in a special frame F_∞. The **MayEnqueue** rule creates *may* proof obligations for frame F_{i+1} using lemmas in frame F_i. This corresponds to pushing lemmas forward more aggressively than the original IC3/PDR. Finally, two additional rules **ResetQ** and **ResetReach** are added to optionally reset the obligation queue and the set of reachable states.

References

1. N. Bjørner and A. Gurfinkel. Property directed polyhedral abstraction. In D. D'Souza, A. Lal, and K. G. Larsen, editors, *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, volume 8931 of *Lecture Notes in Computer Science*, pages 263–281. Springer, 2015.
2. A. R. Bradley. SAT-Based Model Checking without Unrolling. In R. Jhala and D. A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.
3. N. Eén, A. Mishchenko, and R. K. Brayton. Efficient implementation of property directed reachability. In P. Bjesse and A. Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 125–134. FMCAD Inc., 2011.
4. A. Gurfinkel and A. Ivrii. Pushing to the top. In R. Kaivola and T. Wahl, editors, *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 65–72. IEEE, 2015.
5. K. Hoder and N. Bjørner. Generalized property directed reachability. In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2012.
6. A. Komuravelli, A. Gurfinkel, and S. Chaki. SMT-Based Model Checking for Recursive Programs. In A. Biere and R. Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 17–34. Springer, 2014.