Solving Constrained Horn Clauses with SMT

Arie Gurfinkel

International Summer School on Satisfiability, Satisfiability Modulo Theories, and Automated Reasoning

SAT/SMT/AR 2018

July 6, 2018
Automated Verification

Deductive Verification

• A user provides a program and a verification certificate
  – e.g., inductive invariant, pre- and post-conditions, function summaries, etc.
• A tool automatically checks validity of the certificate
  – this is not easy! (might even be undecidable)
• Verification is manual but machine certified

Algorithmic Verification (My research area)

• A user provides a program and a desired specification
  – e.g., program never writes outside of allocated memory
• A tool automatically checks validity of the specification
  – and generates a verification certificate if the program is correct
  – and generates a counterexample if the program is not correct
• Verification is completely automatic – “push-button”
Algorithmic Logic-Based Verification

Program + Spec

Verification Condition (in Logic)

Decision Procedure

Yes

No

Safety Properties

Constrained Horn Clauses

Spacer
Spacer: Solving SMT-constrained CHC

Spacer: a solver for SMT-constrained Horn Clauses

- now the default (and only) CHC solver in Z3
  - [https://github.com/Z3Prover/z3](https://github.com/Z3Prover/z3)
  - dev branch at [https://github.com/agurfinkel/z3](https://github.com/agurfinkel/z3)

Supported SMT-Theories

- Linear Real and Integer Arithmetic
- Quantifier-free theory of arrays
- *Universally quantified theory of arrays + arithmetic (work in progress)*
- Best-effort support for many other SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic

Support for Non-Linear CHC

- for procedure summaries in inter-procedural verification conditions
- for compositional reasoning: abstraction, assume-guarantee, thread modular, etc.
Contributors

Arie Gurfinkel
Anvesh Komuravelli

Nikolaj Bjorner
(Krystof Hoder)
Yakir Vizel
Bernhard Gleiss
Matteo Marescotti
Logic-based Algorithmic Verification

- Simulink
- CoCoSim
- Lustre
- Zustre
- T2
- C/C++
- Java
- CPR
- SeaHorn
- JayHorn
- Termination for C
- Spacer

concurrent/distributed systems
Constrained Horn Clauses (CHC)

A Constrained Horn Clause (CHC) is a FOL formula of the form

$$\forall V . (\phi \land p_1[X_1] \land \ldots \land p_n[X_n] \rightarrow h[X]),$$

where

- $A$ is a background theory (e.g., Linear Arithmetic, Arrays, Bit-Vectors, or combinations of the above)
- $\phi$ is a constrained in the background theory $A$
- $p_1, \ldots, p_n, h$ are $n$-ary predicates
- $p_i[X]$ is an application of a predicate to first-order terms
CHC Notation and Terminology

Rule

\[ h[X] \leftarrow p_1[X_1], \ldots, p_n[X_n], \phi \]

Query

false \leftarrow p_1[X_1], \ldots, p_n[X_n], \phi.

Fact

h[X] \leftarrow \phi.

Linear CHC

h[X] \leftarrow p[X_1], \phi.

Non-Linear CHC

h[X] \leftarrow p_1[X_1], \ldots, p_n[X_n], \phi.

for \ n > 1
CHC Satisfiability

A **model** of a set of clauses $II$ is an extension of the model of the background theory with an interpretation of each predicate $p_i$ that makes all clauses in $II$ valid.

A set of clauses is **satisfiable** if it has a model, and is unsatisfiable otherwise.

Given a theory $A$, a model $M$ is **$A$-definable**, if each $p_i$ in $M$ is definable by a formula $\psi_i$ in $A$.

In the context of program verification:

- a program satisfies a property iff corresponding CHCs are satisfiable.
- verification certificates correspond to models.
- counterexamples correspond to derivations of false.
Horn Clauses for Program Verification

De Angelis et al. Verifying Array Programs by Transforming Verification Conditions. VMCAI'14

Bjørner, Gurfinkel, McMillan, and Rybalchenko: Horn Clause Solvers for Program Verification
Horn Clauses for Concurrent / Distributed / Parameterized Systems

\[
\begin{align*}
R(g, p_1, l, \ldots, p_k, l) & \iff dist(p_1, \ldots, p_k) \land R(g, p_1, l, \ldots, p_k, l) \\
R(g', p_1, l', \ldots, p_k, l) & \iff dist(p_1, \ldots, p_k) \land R(g', l', \ldots, l, p_k, l) \\
R(g', p_1, l, \ldots, p_k, l') & \iff dist(p_1, \ldots, p_k) \land R(g', l', \ldots, l, p_k, l') \\
R(g', p_1, l, \ldots, p_k, l') & \iff dist(p_1, \ldots, p_k) \land R(g, l, \ldots, l, l', p_k, l') \\
false & \iff dist(p_1, \ldots, p_r) \land \left( \bigwedge_{j=1}^{m} (p_j = p_j \land (g, l_j) \in E) \right) \land RConj(1, \ldots, r)
\end{align*}
\]

Figure 4: Horn constraints encoding a homogeneous infinite system with the help of a $k$-indexed invariant. $S_k$ is the symmetric group on \{1, \ldots, k\}, i.e., the group of all permutations of $k$ numbers; as an optimisation, any generating subset of $S_k$, for instance transpositions, can be used instead of $S_k$. In (10), we define $r = \max\{m, k\}$.

Rybalchenko et al. Synthesizing Software Verifiers from Proof Rules. PLDI'12

Hoenicke et al. Thread Modularity at Many Levels. POPL'17

Gurfinkel et al. SMT-Based Verification of Parameterized Systems. FSE 2016

Figure 3: $VC_2(T)$ for two-quantifier invariants.
Is this program correct?

\[
z = x; \ i = 0;
\]
\[
assume (y > 0);
\]
\[
while (i < y) {
    \z = \z + 1;
    \ i = \ i + 1;
}\]
\[
assert(z == x + y);
\]

\[
z = x \& i = 0 \& y > 0 \quad \rightarrow \quad Inv(x, y, z, i)
\]
\[
Inv(x, y, z, i) \& i < y \& z1 = z+1 \& i1 = i+1 \quad \rightarrow \quad Inv(x, y, z1, i1)
\]
\[
Inv(x, y, z, i) \& i \geq y \& z \neq x+y \quad \rightarrow \quad false
\]
In SMT-LIB

(set-logic HORN)

;;; Inv(x, y, z, i)
(declare-fun Inv (Int Int Int Int) Bool)

(assert
(forall ( (A Int) (B Int) (C Int) (D Int))
  (=> (and (> B 0) (= C A) (= D 0))
       (Inv A B C D)))
)

(assert
(forall ( (A Int) (B Int) (C Int) (D Int) (C1 Int) (D1 Int) )
  (=>
     (and (Inv A B C D) (< D B) (= C1 (+ C 1)) (= D1 (+ D 1)))
     (Inv A B C1 D1)
  )
)
)

(assert
(forall ( (A Int) (B Int) (C Int) (D Int))
  (=> (and (Inv A B C D) (> D B) (not (= C (+ A B))))
     false
  )
)
)

(check-sat)
(get-model)

$ z3 add-by-one.smt2
sat
(model
  (define-fun Inv ((x!0 Int) (x!1 Int) (x!2 Int) (x!3 Int)) Bool
    (and (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!3)) 0)
         (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!1)) 0)
         (<= (+ x!0 x!3 (* (- 1) x!2)) 0)))
)

Inv(x, y, z, i)

z = x + i

z <= x + y
Procedures for Solving CHC(T)

Predicate abstraction by lifting Model Checking to HORN
• QARMC, Eldarica, …
Maximal Inductive Subset from a finite Candidate space (Houdini)
• TACAS’18: hoice, FreqHorn

Machine Learning
• PLDI’18: sample, ML to guess predicates, DT to guess combinations

Abstract Interpretation (Poly, intervals, boxes, arrays…)
• Approximate least model by an abstract domain (SeaHorn, …)

Interpolation-based Model Checking
• Duality, QARMC, …

SMT-based Unbounded Model Checking (IC3/PDR)
• Spacer, Implicit Predicate Abstraction
Safety Verification Problem

Is Bad reachable?

 INIT

 Bad
Safety Verification Problem

Is Bad reachable?

Yes. There is a counterexample!
Safety Verification Problem

Is Bad reachable?

No. There is an inductive invariant
Programs, Cexs, Invariants

A program $P = (V, \text{Init}, \mathcal{T}r; \text{Bad})$

- Notation: $\mathcal{F}(X) = \exists u . (X \land \mathcal{T}r) \lor \text{Init}$

$P$ is UNSAFE if and only if there exists a number $N$ s.t.

$$Init(X_0) \land \left( \bigwedge_{i=0}^{N-1} Tr(X_i, X_{i+1}) \right) \land Bad(X_N) \not\rightarrow \bot$$

$P$ is SAFE if and only if there exists a safe inductive invariant $Inv$ s.t.

$$Init \Rightarrow Inv$$

$$Inv(X) \land Tr(X, X') \Rightarrow Inv(X')$$

$$Inv \Rightarrow \neg Bad$$
IC3, PDR, and Friends (1)

IC3: A SAT-based Hardware Model Checker
• Incremental Construction of Inductive Clauses for Indubitable Correctness
• A. Bradley: SAT-Based Model Checking without Unrolling. VMCAI 2011

PDR: Explained and extended the implementation
• Property Directed Reachability
• N. Eén, A. Mishchenko, R. K. Brayton: Efficient implementation of property directed reachability. FMCAD 2011

PDR with Predicate Abstraction (easy extension of IC3/PDR to SMT)
• A. Cimatti, A. Griggio, S. Mover, St. Tonetta: IC3 Modulo Theories via Implicit Predicate Abstraction. TACAS 2014
• J. Birgmeier, A. Bradley, G. Weissenbacher: Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR). CAV 2014
IC3, PDR, and Friends (2)

GPDR: Non-Linear CHC with Arithmetic constraints
• Generalized Property Directed Reachability
• K. Hoder and N. Bjørner: Generalized Property Directed Reachability. SAT 2012

SPACER: Non-Linear CHC with Arithmetic
• fixes an incompleteness issue in GPDR and extends it with under-approximate summaries
• A. Komuravelli, A. Gurfinkel, S. Chaki: SMT-Based Model Checking for Recursive Programs. CAV 2014

PolyPDR: Convex models for Linear CHC
• simulating Numeric Abstract Interpretation with PDR
• N. Bjørner and A. Gurfinkel: Property Directed Polyhedral Abstraction. VMCAI 2015

ArrayPDR: CHC with constraints over Arithmetic + Arrays
• Required to model heap manipulating programs
• A. Komuravelli, N. Bjørner, A. Gurfinkel, K. L. McMillan: Compositional Verification of Procedural Programs using Horn Clauses over Integers and Arrays. FMCAD 2015
SMT-based Model Checking

Generalizing from bounded proofs

A counterexample of length $N$ exists?

- $T$, $N=0$

- No + bounded proof

Generalize proof

- SMT

Is a safe inductive invariant?

- No, $N := N + 1$

- YES

SMT

candidate $Inv$
**IC3/PDR/Spacer Overview**

**Input:** Safety problem \( \langle \text{Init}(X), \text{Tr}(X, X'),\text{Bad}(X) \rangle \)

\[
F_0 \leftarrow \text{Init} ; N \leftarrow 0 \text{ repeat } \\
\quad \text{G} \leftarrow \text{PdrMkSafe}([F_0, \ldots, F_N], \text{Bad}) \\
\quad \text{if } G = [ ] \text{ then return } \text{Reachable}; \\
\quad \forall 0 \leq i \leq N \cdot F_i \leftarrow G[i] \\
\quad F_0, \ldots, F_N \leftarrow \text{PdrPush}([F_0, \ldots, F_N]) \\
\quad \text{if } \exists 0 \leq i < N \cdot F_i = F_{i+1} \text{ then return } \text{Unreachable}; \\
\quad N \leftarrow N + 1 ; F_N \leftarrow \emptyset \\
\text{until } \infty;
\]

**bounded safety**

**strengthen result**
IC3/PDR/Spacer In Pictures: MkSafe

- \( x = 3, y = 0 \)
- \( x = 1, y = 0 \)
- \( x \neq 3 \lor y \neq 0 \)
IC3/PDR in Pictures: Push

Algorithm Invariants

\[ F_i \rightarrow \neg \text{Bad} \quad \text{Init} \rightarrow F_i \]

\[ F_i \rightarrow F_{i+1} \quad F_i \land Tr \rightarrow F_{i+1} \]
IC3/PDR: Solving Linear (Propositional) CHC

Unreachable and Reachable
- terminate the algorithm when a solution is found

Unfold
- increase search bound by 1

Candidate
- choose a bad state in the last frame

Decide
- extend a cex (backward) consistent with the current frame
- choose an assignment $s$ s.t. $(s \land F_i \land Tr \land cex')$ is SAT

Conflict
- construct a lemma to explain why cex cannot be extended
- Find a clause $L$ s.t. $L \Rightarrow \neg cex$, $Init \Rightarrow L$, and $L \land F_i \land Tr \Rightarrow L'$

Induction
- propagate a lemma as far into the future as possible
- (optionally) strengthen by dropping literals
Decide Rule: Generalizing Predecessors

**Decide** If \( \langle m, i + 1 \rangle \in Q \) and there are \( m_0 \) and \( m_1 \) s.t. \( m_1 \rightarrow m \), \( m_0 \land m'_1 \) is satisfiable, and \( m_0 \land m'_1 \rightarrow F_i \land Tr \land m' \), then add \( \langle m_0, i \rangle \) to \( Q \).

**Decide** rule chooses a (generalized) predecessor \( m_0 \) of \( m \) that is consistent with the current frame.

Simplest implementation is to extract a predecessor \( m_0 \) from a satisfying assignment of \( M \models F_i \land Tr \land m' \):
- \( m_0 \) can be further generalized using ternary simulation by dropping literals and checking that \( m' \) remains forced.

An alternative is to let \( m_0 \) be an implicant (not necessarily prime) of \( F_i \land \exists X'.(Tr \land m') \):
- finding a prime implicant is difficult because of the existential quantification
- we settle for an arbitrary implicant. The side conditions ensure it is not trivial.
Conflict Rule: Inductive Generalization

A clause $\varphi$ is inductive relative to $F$ iff

- $\text{Init} \rightarrow \varphi$ (Initialization) and $\varphi \land F \land \text{Tr} \rightarrow \varphi$ (Inductiveness)

Implemented by first letting $\varphi = \neg m$ and generalizing $\varphi$ by iteratively dropping literals while checking the inductiveness condition.

**Theorem:** Let $F_0, F_1, \ldots, F_N$ be a valid IC3 trace. If $\varphi$ is inductive relative to $F_i$, $0 \cdot i < N$, then, for all $j \cdot i$, $\varphi$ is inductive relative to $F_j$.

- Follows from the monotonicity of the trace
  - if $j < i$ then $F_j \rightarrow F_i$
  - if $F_j \rightarrow F_i$ then $(\varphi \land F_i \land \text{Tr} \rightarrow \varphi) \rightarrow (\varphi \land F_j \land \text{Tr} \rightarrow \varphi')$
From Propositional PDR to Solving CHC

Theories with Infinite Models
- infinitely many satisfying assignments
- can’t simply enumerate (in decide)
- can’t block one assignment at a time (in conflict)

Non-Linear Horn Clauses
- multiple predecessors (in decide)

The problem is undecidable in general, but we want an algorithm that makes progress
- doesn’t get stuck in a decidable sub-problem
CHC OVER LINEAR ARITHMETIC
IC3/PDR: Solving Linear (Propositional) CHC

Unreachable and Reachable
  • terminate the algorithm when a solution is found

Unfold
  • increase search bound by 1

Candidate
  • choose a bad state in the last frame

Decide
  • extend a cex (backward) consistent with the current frame
  • choose an assignment \( s \) s.t. \( (s \land R_i \land Tr \land cex') \) is SAT

Conflict
  • construct a lemma to explain why cex cannot be extended
  • Find a clause \( L \) s.t. \( L \Rightarrow \neg cex', \text{ Init } \Rightarrow L, \text{ and } L \land R_i \land Tr \Rightarrow L' \)

Induction
  • propagate a lemma as far into the future as possible
  • (optionally) strengthen by dropping literals

Theory dependent
\[( (F_i \land Tr) \lor Init' ) \Rightarrow \varphi' \]
\[\varphi' \Rightarrow \neg c' \]

Looking for $\varphi'$

**ARITHMETIC CONFLICT**
Craig Interpolation Theorem

**Theorem** (Craig 1957)
Let $A$ and $B$ be two First Order (FO) formulae such that $A \Rightarrow \neg B$, then there exists a FO formula $I$, denoted $ITP(A, B)$, such that

$$A \Rightarrow I \quad I \Rightarrow \neg B$$

$$\text{atoms}(I) \subseteq \text{atoms}(A) \cap \text{atoms}(B)$$

A Craig interpolant $ITP(A, B)$ can be effectively constructed from a resolution proof of unsatisfiability of $A \land B$

In Model Checking, Craig Interpolation Theorem is used to safely over-approximate the set of (finitely) reachable states
Craig Interpolant
Craig Interpolation for Linear Arithmetic

Useful properties of existing interpolation algorithms [CGS10] [HB12]

- $I \in \text{ITP} (A, B)$ then $\neg I \in \text{ITP} (B, A)$
- if $A$ is syntactically convex (a monomial), then $I$ is convex
- if $B$ is syntactically convex, then $I$ is co-convex (a clause)
- if $A$ and $B$ are syntactically convex, then $I$ is a half-space
Arithmetic Conflict

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

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

Counterexample is blocked using Craig Interpolation

• summarizes the reason why the counterexample cannot be extended

Generalization is not inductive

• weaker than IC3/PDR
• inductive generalization for arithmetic is still an open problem
Computing Interpolants for IC3/PDR

Much simpler than general interpolation problem for $A \land B$

- $B$ is always a conjunction of literals
- $A$ is dynamically split into DNF by the SMT solver
- DPLL(T) proofs do not introduce new literals

Interpolation algorithm is reduced to analyzing all theory lemmas in a DPLL(T) proof produced by the solver

- every theory-lemma that mixes $B$-pure literals with other literals is interpolated to produce a single literal in the final solution
- interpolation is restricted to clauses of the form $(\land B_i \Rightarrow \lor A_j)$

Interpolating (UNSAT) Cores (*ongoing work with Bernhard Gleiss*)

- improve interpolation algorithms and definitions to the specific case of PDR
- classical interpolation focuses on eliminating non-shared literals
- in PDR, the focus is on finding good generalizations
Back to addition example…

\[
z = x; \quad i = 0;
\]

\[\text{assume} \ (y > 0);\]

\[\text{while} \ (i < y)\]

\[z = z + 1;\]

\[\text{assert}(z = x + y);\]

\[
z = x \land i = 0 \land y > 0 \quad \Rightarrow \quad \text{Inv}(x, y, z, i)
\]

\[\text{Inv}(x, y, z, i) \land i < y \land z1 = z+1 \land i1 = i+1 \quad \Rightarrow \quad \text{Inv}(x, y, z1, i1)\]

\[\text{Inv}(x, y, z, i) \land i >= y \land z != x+y \quad \Rightarrow \quad \text{false}\]
Lemma Generation Example

Transition Relation

\[ x = x_0 \land z = z_0 + 1 \land i = i_0 + 1 \land y > i_0 \]

Pob

\[ i \geq y \land x + y > z \]

Farkas explanation for unsat

\[ x_0 + y_0 \leq z_0, \ x \leq x_0, z_0 < z, \ i \leq i_0 + 1 \]

\[ x + i \leq z \]

\[ i \geq y, \ x+y > z \]

\[ x + i > z \]

false

Learn lemma:

\[ x + i \leq z \]
\[ s \subseteq \text{pre}(c) \]

\[ \equiv s \Rightarrow \exists X'. Tr \land c' \]

Computing a predecessor \( s \) of a counterexample \( c \)

**ARITHMETIC DECIDE**
Model Based Projection

**Definition:** Let $\varphi$ be a formula, $U$ a set of variables, and $M$ a model of $\varphi$. Then $\psi = \text{MBP} (U, M, \varphi)$ is a Model Based Projection of $U$, $M$ and $\varphi$ iff

1. $\psi$ is a monomial
2. $\text{Vars}(\psi) \subseteq \text{Vars}(\varphi) \setminus U$
3. $M \models \psi$
4. $\psi \Rightarrow \exists U \cdot \varphi$

Model Based Projection under-approximates existential quantifier elimination relative to a given model (i.e., satisfying assignment)
Expensive to find a quantifier-free

$$\psi(y) \equiv \exists x \cdot \varphi(x, y)$$

1. Find model M of $\varphi(x, y)$

2. Compute a partition containing M
Quantifier Elimination for Linear Real Arithmetic

\[ \exists x \cdot \bigwedge_i s_i < x \land \bigwedge_j x < t_j \]

\[ = \bigwedge_i \bigwedge_j \text{resolve}(s_i < x, x < t_j, x) \]

\[ = \bigwedge_i \bigwedge_j s_i < t_j \]

Quadratic increase in the formula size
Quantifier Elimination with an order side-cond

\[
\left( \bigwedge_{j \neq 0} t_0 \leq t_j \right) \land \exists x \cdot \bigwedge_i s_i < x \land \bigwedge_j x < t_j
\]

\[
= \left( \bigwedge_{j \neq 0} t_0 \leq t_j \right) \land \bigwedge_i \text{resolve}(s_i < x, x < t_0, x)
\]

\[
= \left( \bigwedge_{j \neq 0} t_0 \leq t_j \right) \land \bigwedge_i s_i < t_0
\]

Quantifier elimination is simplified by a choice of a minimal upper bound

- For each choice of minimal upper bound, no increase in term size
- Dually, can use largest lower bound

How to chose an order on terms?!

- MBP == use the order chosen by the model
MBP for Linear Rational Arithmetic

Compute a **single** disjunct from LW-QE that includes the model

- Use the Model to uniquely pick a substitution term for $x$

\[
Mb_{p_x}(M, x = s \land L) = L[x \leftarrow s]
\]

\[
Mb_{p_x}(M, x \neq s \land L) = Mb_{p_x}(M, s < x \land L) \text{ if } M(x) > M(s)
\]

\[
Mb_{p_x}(M, x \neq s \land L) = Mb_{p_x}(M, -s < -x \land L) \text{ if } M(x) < M(s)
\]

\[
Mb_{p_x}(M, \bigwedge_i s_i < x \land \bigwedge_j x < t_j) = \bigwedge_i s_i < t_0 \land \bigwedge_j t_0 \leq t_j \text{ where } M(t_0) \leq M(t_i), \forall i
\]

MBP techniques have been developed for

- Linear Rational Arithmetic, Linear Integer Arithmetic
- Theories of Arrays, and Recursive Data Types
Arithmetic Decide

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

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

Compute a predecessor using Model Based Projection

To ensure progress, Decide must be finite

- finitely many possible predecessors when all other arguments are fixed

Alternatively

- Completeness can follow from an interaction of Decide and Conflict
  – but requires more rules to propagate implicants backward (as in PDR) and forward (as in Spacer and Quip)
SOLVING NON-LINEAR CHC
Horn Clauses for Program Verification

De Angelis et al. Verifying Array Programs by Transforming Verification Conditions. VMCAI'14

Bjørner, Gurfinkel, McMillan, and Rybalchenko: Horn Clause Solvers for Program Verification
Horn Clauses for Concurrent / Distributed / Parameterized Systems

For assertions $R_1, \ldots, R_N$ over $V$ and $E_1, \ldots, E_N$ over $V, V'$,

- **CM1**: $\text{init}(V) \rightarrow R_i(V)$
- **CM2**: $R_i(V) \land \rho_i(V, V') \rightarrow R_i(V')$
- **CM3**: $(\forall_{i \in 1..N} \{ j \}) R_i(V) \land \rho_i(V, V') \rightarrow E_j(V, V')$
- **CM4**: $R_i(V) \land E_i(V, V') \land \rho_i'(V, V') \rightarrow R_i(V')$
- **CM5**: $R_1(V) \land \cdots \land R_N(V) \land \text{error}(V) \rightarrow \text{false}$

multi-threaded program $P$ is safe

---

**Rybalchenko et al. Synthesizing Software Verifiers from Proof Rules. PLDI'12**

**Hojjat et al. Horn Clauses for Communicating Timed Systems. HCVS'14**

**Hoenicke et al. Thread Modularity at Many Levels. POPL'17**

---

Gurfinkel et al. SMT-Based Verification of Parameterized Systems. FSE 2016

---

**Figure 3**: $VC_2(T)$ for two-quantifier invariants.

- \begin{align*}
  \text{Init}(i, j, \overline{v}) \land \text{Init}(j, i, \overline{v}) \\
  \text{Init}(i, i, \overline{v}) \land \text{Init}(j, j, \overline{v}) \Rightarrow I_2(i, j, \overline{v}) \quad (3) \\
  I_2(i, j, \overline{v}) \land \text{Tr}(i, \overline{v}, \overline{v}') \Rightarrow I_2(i, j, \overline{v}') \quad (4) \\
  I_2(i, j, \overline{v}) \land I_2(i, k, \overline{v}) \land I_2(j, k, \overline{v}) \land \text{Tr}(k, \overline{v}, \overline{v}') \land k \neq i \land k \neq j \Rightarrow I_2(i, j, \overline{v}') \quad (5) \\
  I_2(i, j, \overline{v}) \Rightarrow \neg \text{Bad}(i, j, \overline{v})
  \end{align*}

**Figure 4**: Horn constraints encoding a homogeneous infinite system with the help of a $k$-indexed invariant. $S_k$ is the symmetric group on $\{1, \ldots, k\}$, i.e., the group of all permutations of $k$ numbers; as an optimisation, any generating subset of $S_k$, for instance transpositions, can be used instead of $S_k$. In (10), we define $r = \max\{m, k\}$. 

---

49
Non-Linear CHC Satisfiability

Satisfiability of a set of arbitrary (i.e., linear or non-linear) CHCs is reducible to satisfiability of THREE clauses of the form

\[
\text{Init}(X) \rightarrow P(X)
\]

\[
P(X) \land P(X^o) \land \text{Tr}(X, X^o, X') \rightarrow P(X')
\]

\[
P(X) \rightarrow \neg \text{Bad}(X)
\]

where, \(X' = \{x' \mid x \in X\}\), \(X^o = \{x^o \mid x \in X\}\), \(P\) a fresh predicate, and \(\text{Init}, \text{Bad}, \text{and Tr}\) are constraints.
Generalized GPDR

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

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

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

**Require:** \(\text{Init} \not\rightarrow \neg \text{Bad}\)

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 \(F_N \rightarrow \neg \text{Bad}\), then set \(N \leftarrow N + 1\) and \(Q \leftarrow \emptyset\).
- **Candidate** If for some \(m, m \rightarrow F_N \land \text{Bad}\), then add \(\langle \langle m, N \rangle \rangle\) to \(Q\).
- **Decide** If there is a \(t \in Q\), with \(c = \langle m, i + 1 \rangle \in t, m_1 \rightarrow m, l_0 \land m_0^o \land m_1'\) is satisfiable, and \(l_0 \land m_0^o \land m_1' \rightarrow F_i \land F_i^o \land \text{Tr} \land m'\) then add \(\hat{t}\) to \(Q\), where \(\hat{t} = t\) with \(c\) replaced by two tuples \(\langle l_0, i \rangle\), and \(\langle m_0, i \rangle\).
- **Conflict** If there is a \(t \in Q\) with \(c = \langle m, i + 1 \rangle \in t\), s.t. \(F_i \land m'\) is unsatisfiable. Then, add \(\varphi = \text{Itp}(F_i, m')\) to \(F_j\), for all \(0 \leq j \leq i + 1\).
- **Leaf** If there is \(t \in Q\) with \(c = \langle m, i \rangle \in t, 0 < i < N\) and \(F_{i-1} \land m'\) is unsatisfiable, then add \(\hat{t}\) to \(Q\), where \(\hat{t}\) is \(t\) with \(c\) replaced by \(\langle m, i + 1 \rangle\).
- **Induction** For \(0 \leq i < N\) and a clause \((\varphi \lor \psi) \in F_i\), if \(\varphi \not\in F_{i+1}\), \(F_i \land F_{i+1} \rightarrow \phi'\) then add \(\varphi\) to \(F_j\), for all \(j \leq i + 1\).

until \(\infty\);
Counterexamples to non-linear CHC

A set $S$ of CHC is unsatisfiable iff $S$ can derive FALSE

- we call such a derivation a counterexample

For linear CHC, the counterexample is a path
For non-linear CHC, the counterexample is a tree

\[
\begin{align*}
S' & \in S_2 \land S^0_3 \land \text{Tr} \\
S' & \in S_0 \land S^0_1 \land \text{Tr}
\end{align*}
\]

\[
\begin{align*}
S_2 & \in \text{Init} \\
S_3 & \in \text{Init} \\
S_0 & \in \text{Init} \\
S_1 & \in \text{Init}
\end{align*}
\]
At each step, one CTI in the frontier is chosen and its two children are expanded.
GPDR: Splitting predecessors

Consider a clause

\[ P(x) \land P(y) \land x > y \land z = x + y \implies P(z) \]

How to compute a predecessor for a proof obligation \( z > 0 \)

Predecessor over the constraint is:

\[
\exists z \cdot x > y \land z = x + y \land z > 0 \\
= x > y \land x + y > 0
\]

Need to create two separate proof obligation

- one for \( P(x) \) and one for \( P(y) \)
- gpdr solution: split by substituting values from the model (incomplete)
GPDR: Deciding predecessors

**Decide** If there is a \( t \in Q \), with \( c = \langle m, i + 1 \rangle \in t \), \( m_1 \rightarrow m \), \( l_0 \wedge m_0^0 \wedge m'_1 \) is satisfiable, and \( l_0 \wedge m_0^0 \wedge m'_1 \rightarrow F_i \wedge F_i^0 \wedge Tr \wedge m' \) then add \( \hat{t} \) to \( Q \), where \( \hat{t} = t \) with \( c \) replaced by two tuples \( \langle l_0, i \rangle \), and \( \langle m_0, i \rangle \).

Compute two predecessors at each application of GPDR/Decide

Can explore both predecessors in parallel
- e.g., BFS or DFS exploration order

Number of predecessors is unbounded
- incomplete even for finite problem (i.e., non-recursive CHC)

No caching/summarization of previous decisions
- worst-case exponential for Boolean Push-Down Systems
Input: A safety problem \( \langle \text{Init}(X), \text{Tr}(X, X^0, X'), \text{Bad}(X) \rangle \).

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

Notation: \( F(A, B) = \text{Init}(X') \lor (A(X) \land B(X^0) \land \text{Tr}) \), and \( F(A) = F(A, A) \)

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

Require: \( \text{Init} \rightarrow \neg \text{Bad} \)

Repeat

Unreachable If there is an \( i < N \) s.t. \( F_i \subseteq F_{i+1} \) 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 \( M M \models \psi \), where \( \psi = F(\forall \text{REACH}) \land m' \). 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 \( M M \models \psi \), where \( \psi = F(F_i, \forall \text{REACH}) \land m' \). Then, add \( s \) to \( Q \), where \( s \in \text{MBP}(\{X^0, X'\}, \psi) \).

DecideMay If there is \( \langle m, i + 1 \rangle \in Q \) and a model \( M M \models \psi \), 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 \( \varphi \not\in F_{i+1} \), \( F(\varphi \land F_i) \rightarrow \varphi' \), then add \( \varphi \) to \( F_j \), for all \( j \leq i + 1 \).

Until \( \infty \);
Unfold the derivation tree in a fixed depth-first order

- use MBP to decide on counterexamples

Learn new facts (reachable states) on the way up

- use MBP to propagate facts bottom up
Successor Rule: Computing Reachable States

**Successor** If there is \( \langle m, i + 1 \rangle \in Q \) and a model \( \mathcal{M} M \models \psi \), where 
\[
\psi = \mathcal{F}(\forall \text{REACH}) \land m'.
\]
Then, add \( s \) to \( \text{REACH} \), where 
\[
s' \in \text{MBP}(\{X, X^o\}, \psi).
\]

Computing new reachable states by under-approximating forward image using MBP

- since MBP is finite, guarantee to exhaust all reachable states

Second use of MBP

- orthogonal to the use of MBP in Decide
- \( \text{REACH} \) can contain auxiliary variables, but might get too large

For Boolean CHC, the number of reachable states is bounded

- complexity is polynomial in the number of states
- same as reachability in Push Down Systems
Decide Rule: Must and May refinement

**DecideMust** If there is \( (m, i + 1) \in Q \), and a model \( M M \models \psi \), where 
\[
\psi = \mathcal{F}(F_i, \lor \text{REACH}) \land m'.
\]
Then, add \( s \) to \( Q \), where 
\[
s \in \text{MBP}(\{X^o, X'\}, \psi).
\]

**DecideMay** If there is \( (m, i + 1) \in Q \) and a model \( M M \models \psi \), where 
\[
\psi = \mathcal{F}(F_i) \land m'.
\]
Then, add \( s \) to \( Q \), where 
\[
s^o \in \text{MBP}(\{X, X'\}, \psi).
\]

**DecideMust**
- use computed summary to skip over a call site

**DecideMay**
- use over-approximation of a calling context to guess an approximation of the call-site
- the call-site either refutes the approximation (**Conflict**) or refines it with a witness (**Successor**)

**DecideMust** If there is \( (m, i + 1) \in Q \), and a model \( M M \models \psi \), where 
\[
\psi = \mathcal{F}(F_i, \lor \text{REACH}) \land m'.
\]
Then, add \( s \) to \( Q \), where 
\[
s \in \text{MBP}(\{X^o, X'\}, \psi).
\]

**DecideMay** If there is \( (m, i + 1) \in Q \) and a model \( M M \models \psi \), where 
\[
\psi = \mathcal{F}(F_i) \land m'.
\]
Then, add \( s \) to \( Q \), where 
\[
s^o \in \text{MBP}(\{X, X'\}, \psi).
\]
Conclusion and Future Work

Spacer: an SMT-based procedure for deciding CHC modulo theories

- extends IC3/PDR from SAT to SMT
- interpolation to over-approximate a possible model
- model-based projection to summarize derivations

The curse of interpolation

- interpolation is fantastic at quickly discovering good lemmas
- BUT it is highly unstable: small changes to input (or code) drastically change what is discovered
- what is easy today might be difficult tomorrow 😞

Many open problems

- Parallel solving (see FMCAD’17)
- Supporting extra theories: bit-vectors, uninterpreted functions, EPR
- Stability – reduce reliance on interpolation
- Exploration strategies, transformations, heuristics, …
Constrained Horn Clauses (CHC) is a fragment of First Order Logic (FOL) that is sufficiently expressive to describe many verification, inference, and synthesis problems including inductive invariant inference, model checking of safety properties, inference of procedure summaries, regression verification, and sequential equivalence. The CHC competition (CHC-COMP) will compare state-of-the-art tools for CHC solving with respect to performance and effectiveness on a set of publicly available benchmarks. The winners among participating solvers are recognized by measuring the number of correctly solved benchmarks as well as the runtime.

Web: https://chc-comp.github.io/
Gitter: https://gitter.im/chc-comp/Lobby
GitHub: https://github.com/chc-comp
Farkas Lemma

Let $M = t_1 \geq b_1 \land \ldots \land t_n \geq b_n$, where $t_i$ are linear terms and $b_i$ are constants. $M$ is unsatisfiable iff $0 \geq 1$ is derivable from $M$ by resolution.

$M$ is unsatisfiable iff $M \vdash 0 \geq 1$

- e.g., $x + y > 10, -x > 5, -y > 3 \vdash (x+y-x-y) > (10 + 5 + 3) \vdash 0 > 18$

$M$ is unsatisfiable iff there exist Farkas coefficients $g_1, \ldots, g_n$ such that

- $g_i \geq 0$
- $g_1 \times t_1 + \ldots + g_n \times t_n = 0$
- $g_1 \times b_1 + \ldots + g_n \times b_n \geq 1$
Interpolation for Linear Real Arithmetic

Let $M = A \land B$ be UNSAT, where

- $A = t_1 \geq b_1 \land \ldots \land t_i \geq b_i$, and
- $B = t_{i+1} \geq b_i \land \ldots \land t_n \geq b_n$

Let $g_1, \ldots, g_n$ be the Farkas coefficients witnessing UNSAT

Then

- $g_1 \times (t_1 \geq b_1) + \ldots + g_i \times (t_i \geq b_i)$ is an interpolant between $A$ and $B$
- $g_{i+1} \times (t_{i+1} \geq b_i) + \ldots + g_n \times (t_n \geq b_n)$ is an interpolant between $B$ and $A$

- $g_1 \times t_1 + \ldots + g_i \times t_i = - (g_{i+1} \times t_{i+1} + \ldots + g_n \times t_n)$
- $\neg (g_{i+1} \times (t_{i+1} \geq b_i) + \ldots + g_n \times (t_n \geq b_n))$ is an interpolant between $A$ and $B$
Craig Interpolation for Linear Arithmetic

Useful properties of existing interpolation algorithms [CGS10] [HB12]

- $I \in \text{ITP} (A, B)$ then $\neg I \in \text{ITP} (B, A)$
- if $A$ is syntactically convex (a monomial), then $I$ is convex
- if $B$ is syntactically convex, then $I$ is co-convex (a clause)
- if $A$ and $B$ are syntactically convex, then $I$ is a half-space