Algorithmic Logic-Based Verification with SeaHorn

Arie Gurfinkel

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

http://ece.uwaterloo.ca/~agurfink

based on work with Teme Kahsai, Jorge Navas, Anvesh Komuravelli, Jeffrey Gennari, Ed Schwartz, and many others
Automated Software Analysis

Program → Automated Analysis

Software Model Checking with Predicate Abstraction
  e.g., Microsoft's SDV

Abstract Interpretation with Numeric Abstraction
  e.g., ASTREE, Polyspace

Correct
Incorrect
Turing, 1936: “undecidable”
Alan M. Turing. “Checking a large routine”, 1949

How can one check a routine in the sense of making sure that it is right? programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
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

• 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

Safety Properties

Constrained Horn Clauses

Spacer

Yes

No
SeaHorn

A fully automated verification framework for LLVM-based languages.

http://seahorn.github.io
Temesghen Kahsai (NASA/CMU)

Jorge Navas (SRI)

http://seahorn.github.io
SeaHorn Usage

Example: in test.c, check that \( x \) is always greater than or equal to \( y \)

```c
extern int nd();
extern void __VERIFIER_error() __attribute__((noreturn));
void assert (int cond) { if (!cond) __VERIFIER_error (); }
int main()
{
    int x,y;
    x=1; y=0;
    while (nd () )
    {
        x=x+y;
        y++;
    }
    assert (x>=y);
    return 0;
}
```

SeaHorn command:

```
$ -> sea pf test.c
```

SeaHorn result:

```
---------------------------------------------------------------------
SEAHORN
---------------------------------------------------------------------
PROPERTY (line 12) | TRUE
TIME(ms)            | 0.06
---------------------------------------------------------------------
```
SeaHorn Philosophy

Build a state-of-the-art Software Model Checker

• useful to “average” users
  – user-friendly, efficient, trusted, certificate-producing, …
• useful to researchers in verification
  – modular design, clean separation between syntax, semantics, and logic, …

Stand on the shoulders of giants

• reuse techniques from compiler community to reduce verification effort
  – SSA, loop restructuring, induction variables, alias analysis, …
  – static analysis and abstract interpretation
• reduce verification to logic
  – verification condition generation
  – Constrained Horn Clauses

Build reusable logic-based verification technology

• “SMT-LIB” for program verification
Three-Layers of a Program Verifier

Compiler
• compiles surface syntax a target machine
• embodies syntax with semantics

Verification Condition Generator
• transforms a program and a property to a verification condition in logic
• employs different abstractions, refinements, proof-search strategies, etc.

Automated Theorem Prover / Reasoning Engine
• discharges verification conditions
• general purpose constraint solver
• SAT, SMT, Abstract Interpreter, Temporal Logic Model Checker,…
SeaHorn Architecture

- **Front-end**
  - LLVM Opt
  - Devirt/Exc Elim
  - Property Instr
  - Lifting Assert

- **Middle-end**
  - Heap Disambig
  - Array Abstraction
  - VC Generation: small, large, flat...
  - Precision: scalars, pointers, memory

- **Back-end**
  - Template Inv
    - BMC
    - Crab
    - Spacer

- C/C++ → LLVM bitcode

- Verbally:
  - Precision: scalars, pointers, memory
  - BMC
  - Crab
  - Spacer
DEMO
Property-Directed Test-Case Generation

**Software Model Checker**
- Efficient complete traces
- Precise abstract traces
- Trace

**Directed Symbolic Execution**
- Executable harness
- Interacton between SMC and SE

**PDTG**
- Property
- Program
A Counterexample Harness

```c
if (get_input() == 0x1234 &&
    get_input() == 0x8765) {
    __VERIFIER_error();
} else {
    return 0;
}
```

`get_input()` is an external function

Program considered buggy if and only if `__VERIFIER_error()` is reachable

```c
void __get_input() {
    static int x = 0;
    switch (x++) {
        case 0: return 0x1234;
        case 1: return 0x8765;
        default: return 0;
    }
}
```

Implementation of external functions linked to original source code

Causes program to execute `__VERIFIER_error()`
Generating Harnesses for Linux Device Drivers

```c
void *ldv_ptr(void)
{
    void *tmp;
    tmp = __c();
    return tmp;
}
```

```c
void *is_got = ldv_ptr();
if (is_got <= (long)2012)
{
    ...
}
```

- Sample from Linux Device Verification (LDV) project\(^1\)
- Harness functions returning pointers are tricky
  - May not be reasonable addresses
  - Might return “new” memory
- Original program instrumented with memory read/store hooks that control access to external memory

\(^1\)http://linuxtesting.org/ldv
Virtual External Memory

Accesses to external “virtual” memory are mapped to real memory
- opportunistically allocate memory for new accesses
- ignore invalid stores, return a default value for an invalid load
SMT-BASED DECISION PROCEDURE FOR DECIDING CHC
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
Symbolic Reachability Problem

\[ P = (V, \text{Init}, \text{Tr}, \text{Bad}) \]

\( P \) is UNSAFE if and only if there exists a number \( N \) s.t.

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

\( P \) is SAFE if and only if there exists a safe inductive invariant \( \text{Inv} \) s.t.

\[
\begin{align*}
\text{Init} & \Rightarrow \text{Inv} \\
\text{Inv}(X) \land \text{Tr}(X, X') & \Rightarrow \text{Inv}(X')
\end{align*}
\]

\[
\text{Inv} \Rightarrow \neg \text{Bad}
\]
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

A **model** of a set of clauses is an interpretation of the predicates $p_i$ and $h$ that makes all clauses **valid**

A set of clauses is **satisfiable** iff it has a model
int x = 1;
int y = 0;
while (*) {
    x = x + y;
    y = y + 1;
}
assert(x ≥ y);

l₀ : 
x = 1
y = 0

l₁ : b₁ = nondet()

l₂ :
x = x + y
y = y + 1

l₃ :
b₂ = x ≥ y

l₄ :

l₅ :
p₁(x, y) ←
p₀, x = 1, y = 0.

l₆ :
p₂(x, y) ← p₁(x, y).

l₇ :
p₃(x, y) ← p₁(x, y).

l₈ :
p₁(x', y') ←
p₂(x, y),
    x' = x + y,
    y' = y + 1.

l₉ :
p₄ ← (x ≥ y), p₃(x, y).

l₁₀ :
p₅ ← (x < y), p₃(x, y).

l₁₁ :
l₄ ← p₄.

l₁₂ :
l₄ ← p₅.

l₁₃ :
l₄ ← p₆.
Spacer: Solving SMT-constrained CHC

Spacer: a solver for SMT-constrained Horn Clauses
- stand-alone implementation in a fork of Z3
- [http://bitbucket.org/spacer/code](http://bitbucket.org/spacer/code)

Support for Non-Linear CHC
- model procedure summaries in inter-procedural verification conditions
- model assume-guarantee reasoning
- uses MBP to under-approximate models for finite unfoldings of predicates
- uses MAX-SAT to decide on an unfolding strategy

Supported SMT-Theories
- Best-effort support for arbitrary SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic
- Full support for Linear arithmetic (rational and integer)
- Quantifier-free theory of arrays
  - only quantifier free models with limited applications of array equality
Verification by Evolving Approximations

approx. 1

solver

Inductive Invariant

Lemma1

Lemma2

Lemma3

Safe?

No

approx. 2

solver

Inductive Invariant

Lemma1

Lemma2

Lemma3

Safe?

No

approx. 3

solver

Inductive Invariant

Lemma1

Lemma2

Lemma3

Safe?

No
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)
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
Algorithm Overview

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

\[ F_0 \leftarrow \text{Init} ; N \leftarrow 0 \text{ repeat} \]

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

bounded safety

strengthen result
IC3/PDR In Pictures: MkSafe

\[ x = 3, y = 0 \]

\[ x = 1, y = 0 \]

\[ x < y \]
IC3/PDR in Pictures: Push

Algorithm Invariants

\[ R_i \rightarrow \neg \text{Bad} \]
\[ \text{Init} \rightarrow R_i \]
\[ R_i \rightarrow R_{i+1} \]
\[ R_i \land \rho \rightarrow R_{i+1} \]
Logic-based Algorithmic Verification

- Simulink
- C/C++
- Java
- CPR
- concurrent/distributed systems
- CoCoSim
- Lustre
- Zustre
- T2
- SeaHorn
- JayHorn
- Spacer
- Termination for C
SV-COMP 2015

4th Competition on Software Verification held at TACAS 2015

Goals

• Provide a snapshot of the state-of-the-art in software verification to the community.
• Increase the visibility and credits that tool developers receive.
• Establish a set of benchmarks for software verification in the community.

Participants:

• Over 22 participants, including most popular Software Model Checkers and Bounded Model Checkers

Benchmarks:

• C programs with error location (programs include pointers, structures, etc.)
• Over 6,000 files, each 2K – 100K LOC
• Linux Device Drivers, Product Lines, Regressions/Tricky examples
• http://sv-comp.sosy-lab.org/2015/benchmarks.php
Results for DeviceDriver category
Applications of SeaHorn at NASA

Absence of Buffer Overflows
• Open source auto-pilots
  – paparazzi and mnav autopilots
• Automatically instrument buffer accesses with runtime checks
• Use SeaHorn to validate that run-time checks never fail
  – slower than pure abstract interpretation
  – BUT, much more precise!

Verify Level 5 requirements of the LADEE software stack
• Manually encode requirements in Simulink model
• Verify that the requirements hold in auto-generated C

Memory safety of C++ controller code
• ongoing…
Conclusion

SeaHorn ([http://seahorn.github.io](http://seahorn.github.io))
- a state-of-the-art Software Model Checker
- LLVM-based front-end
- CHC-based verification engine
- a framework for research in logic-based verification

The future
- making SeaHorn useful to the consumers of verification technology
  - counterexamples, build integration, property specification, proofs,
- Concurrent / distributed / embedded systems
  - cyber-physical systems
  - very challenging but there are many opportunities
- richer properties
  - termination[TACAS’16], liveness, synthesis
IC3/PDR

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

repeat

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

\text{Reachable} If there is an \(m\) s.t. \(\langle m, 0 \rangle \in Q\) \return Reachable.

\text{Unfold} If \(F_N \rightarrow \neg \text{Bad}\), then set \(N \leftarrow N + 1\).

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

\text{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 \(m_0 \land m_1' \rightarrow F_i \land \text{Tr} \land m'\), then add \(\langle m_0, i \rangle\) to \(Q\).

\text{Conflict} For \(0 \leq i < N\): given a candidate model \(\langle m, i + 1 \rangle \in Q\) and clause \(\varphi\), such that \(\varphi \rightarrow \neg m\), if \(\text{Init} \rightarrow \varphi\), and \(\varphi \land F_i \land \text{Tr} \rightarrow \varphi'\), then add \(\varphi\) to \(F_j\), for \(j \leq i + 1\).

\text{Leaf} If \(\langle m, i \rangle \in Q\), \(0 < i < N\) and \(F_{i-1} \land \text{Tr} \land m'\) is unsatisfiable, then add \(\langle m, i + 1 \rangle\) to \(Q\).

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

until \(\infty\);
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 s s.t. \((s \land R_i \land Tr \land \text{cex'})\) is SAT

Conflict
- Find a lemma that explains why cex cannot be extended
- Find \(L\) s.t. \(L \Rightarrow \neg \text{cex}\) and \(L \land R_i \land Tr \Rightarrow L'\)

Induction
- Propositional generalization (drop literals from the lemma)
Looking for $\varphi'$

ARITHMETIC CONFLICT

$$(((F_i \land Tr) \lor Init') \Rightarrow \varphi')$$

$$\varphi' \Rightarrow c'$$
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

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

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

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

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

A

B

I
Examples of Craig Interpolation for Theories

Boolean logic

\[ A = (\neg b \land (\neg a \lor b \lor c) \land a) \quad B = (\neg a \lor \neg c) \]

\[ \text{ITP}(A, B) = a \land c \]

Equality with Uninterpreted Functions (EUF)

\[ A = (f(a) = b \land p(f(a))) \quad B = (b = c \land \neg p(c)) \]

\[ \text{ITP}(A, B) = p(b) \]

Linear Real Arithmetic (LRA)

\[ A = (z + x + y > 10 \land z < 5) \quad B = (x < -5 \land y < -3) \]

\[ \text{ITP}(A, B) = x + y > 5 \]
Alternative Definition of an Interpolant

Let $F = A(x, z) \land B(z, y)$ be UNSAT, where $x$ and $y$ are distinct

- Note that for any assignment $v$ to $z$ either
  - $A(x, v)$ is UNSAT, or
  - $B(v, y)$ is UNSAT

An interpolant is a circuit $I(z)$ such that for every assignment $v$ to $z$

- $I(v) = A$ only if $A(x, v)$ is UNSAT
- $I(v) = B$ only if $B(v, y)$ is UNSAT

A proof system $S$ has a feasible interpolation if for every refutation $\pi$ of $F$ in $S$, $F$ has an interpolant polynomial in the size of $\pi$

- propositional resolution has feasible interpolation
- extended resolution does not have feasible interpolation
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 \textit{unsatisfiable} iff $0 \geq 1$ is derivable from $M$ by resolution.

$M$ is \textit{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 \textit{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

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

Let $g_1, ..., g_n$ be the Farkas coefficients witnessing UNSAT

Then

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

- $g_1 \cdot t_1 + ... + g_i \cdot t_i = - (g_{i+1} \cdot t_{i+1} + ... + g_n \cdot t_n)$
- $\neg(g_{i+1} \cdot (t_{i+1} \geq b_i) + ... + g_n \cdot (t_n \geq b_n))$ 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
Arithmetic Conflict

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

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 $P'^\uparrow = \text{ITP}(\mathcal{F}(F_i), P')$ 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
\[ s \subseteq pre(c) \]

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

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

**ARITHMETIC DECIDE**
Model Based Projection

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

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

For a fixed set of variables $U$ and a formula $\phi$, MBP is a function from models to formulas.

MBP is *finite* if its range (as a function defined above) is finite.
1. Find model M of $\varphi(x,y)$

2. Compute a partition containing M
Loos Weispfenning Quantifier Elimination

\( \varphi \) is LRA formula in Negation Normal Form

\( E \) is set of \( x = t \) atoms, \( U \) set of \( x < t \) atoms, and \( L \) set of \( s < x \) atoms

There are no other occurrences of \( x \) in \( \varphi[x] \)

\[ \exists x. \varphi[x] \equiv \varphi[\infty] \lor \bigvee_{x = t \in E} \varphi[t] \lor \bigvee_{x < t \in U} \varphi[t - \epsilon] \]

where

\[ (x < t')[t - \epsilon] \equiv t \leq t' \quad (s < x)[t - \epsilon] \equiv s < t \quad (x = e)[t - \epsilon] \equiv \text{false} \]

The case of lower bounds is dual

- using \(-\infty\) and \( t + \epsilon \)
LW-Quantifier Elimination Example

\[ \exists x . \varphi[x] \equiv \exists x . (x = e \land \psi_1) \lor (s < x \land x < t) \lor (x < t \land \psi_2) \]
\[ \equiv \varphi[e] \lor \varphi[t - \epsilon] \lor \varphi[\infty] \]
\[ \equiv (\psi_1 \lor (s < e \land e < t) \lor (e < t \land \psi_2)) \lor (s < t \land t \leq t) \lor (t \leq t \land \psi_2) \lor false \]
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 \(x\)

\[
Mbp_x(M, x = s \land L) = L[x \leftarrow s]
\]

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

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

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

MBP techniques have been developed for

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

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

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 $\langle P_{\perp}, i \rangle$ to $Q$, where $P_{\perp} = \text{MBP}(X', m, \mathcal{F}(F_i) \land P').$

Compute a predecessor using an under-approximation of quantifier elimination – called Model Based Projection

To ensure progress, Decide must be finite

- finitely many possible predecessors when all other arguments are fixed

Alternatives

- Completeness can follow from the Conflict rule only
  - for Linear Arithmetic this means using Fourier-Motzkin implicants
- Completeness can follow from an interaction of Decide and Conflict
PROPERTY-DIRECTED TEST
CASE GENERATION
Property-Directed Test-Case Generation

- **Property Directed Test Case Generation (PDTG)**
  - Efficient complete traces
  - Interaction between SMC and SE
  - Executable harness

- **Software Model Checker**
  - Directed Symbolic Execution
  - Precise abstract traces
  - Trace

- **Program**
  - Property
A Counterexample Harness

```c
if (get_input() == 0x1234 &&
    get_input() == 0x8765) {
    __VERIFIER_error();
} else {
    return 0;
}
```

get_input() is an external function

Program considered buggy if and only if `__VERIFIER_error()` is reachable

```c
void __get_input() {
    static int x = 0;
    switch (x++) {
        case 0: return 0x1234;
        case 1: return 0x8765;
        default: return 0;
    }
}
```

Implementation of external functions linked to original source code

Causes program to execute `__VERIFIER_error()`
Generating Harnesses for Linux Device Drivers

```c
void *ldv_ptr(void)
{
    void *tmp;
    tmp = __c();
    return tmp;
}

... 

void *is_got = ldv_ptr();
if (is_got <= (long)2012)
{
    ... }
```

- Sample from Linux Device Verification (LDV) project\(^1\)
- Harness functions returning pointers are tricky
  - May not be reasonable addresses
  - Might return “new” memory

- Original program instrumented with memory read/store hooks that control access to external memory

\(^1\)http://linuxtesting.org/ldv
Virtual External Memory

Accesses to external “virtual” memory are mapped to real memory
- opportunistically allocate memory for new accesses
- ignore invalid stores, return a default value for an invalid load