Lectures

Friday 15:45–17:15

Arrays, Polynomials, Quantifiers, Fixed-points, ...

Summary
Plan

I. Arrays
II. Polynomials
III. Quantifiers
IV. Fixed-points
Takeaways

Learn Additional decision procedures

Modeling with quantifiers and solving
Symbolic Reasoning

High computational Complexity

Practical problems often have **structure** that can be exploited.
# Research around Z3

## Decision Procedures

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular Difference Logic is Hard</td>
<td>TR 08 B, Blass Gurevich, Muthuvathi.</td>
</tr>
<tr>
<td>Linear Functional Fixed-points.</td>
<td>CAV 09 B. &amp; Hendrix.</td>
</tr>
<tr>
<td>Efficient, Generalized Array Decision Procedures</td>
<td>FMCAD 09 M &amp; B.</td>
</tr>
<tr>
<td>Quantifier Elimination as an Abstract Decision Procedure</td>
<td>IJCAR 10, B.</td>
</tr>
<tr>
<td>Cutting to the Chase</td>
<td>CADE 11, Jojanovitch, M</td>
</tr>
<tr>
<td>Polynomials</td>
<td>IJCAR 12, Jojanovitch, M.</td>
</tr>
</tbody>
</table>

## Combining Decision Procedures

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-based Theory Combination</td>
<td>SMT 07 M &amp; B.</td>
</tr>
<tr>
<td>Proofs, Refutations and Z3</td>
<td>IWIL 08 M &amp; B.</td>
</tr>
<tr>
<td>On Locally Minimal Nullstellensatz Proofs.</td>
<td>SMT 09 M &amp; Passmore.</td>
</tr>
<tr>
<td>A Concurrent Portfolio Approach to SMT Solving</td>
<td>CAV 09 Wintersteiger, Hamadi &amp; M.</td>
</tr>
<tr>
<td>Conflict Directed Theory Resolution</td>
<td>Cambridge Univ. Press 12, M &amp; B.</td>
</tr>
</tbody>
</table>

## Quantifiers, quantifiers, quantifiers

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient E-matching for SMT Solvers.</td>
<td>CADE 07 M &amp; B.</td>
</tr>
<tr>
<td>Relevancy Propagation.</td>
<td>TR 07 M &amp; B.</td>
</tr>
<tr>
<td>Deciding Effectively Propositional Logic using DPLL and substitution</td>
<td>IJCAR 08 M &amp; B.</td>
</tr>
<tr>
<td>Engineering DPLL(T) + saturation.</td>
<td>IJCAR 08 M &amp; B.</td>
</tr>
<tr>
<td>Complete instantiation for quantified SMT formulas</td>
<td>CAV 09 Ge &amp; M.</td>
</tr>
<tr>
<td>On deciding satisfiability by DPLL( Γ+ T) and unsound theorem proving.</td>
<td>CADE 09 Bonachina, M &amp; Lynch.</td>
</tr>
<tr>
<td>Generalized PDR</td>
<td>SAT 12 Hoder &amp; B.</td>
</tr>
</tbody>
</table>
Arrays
Arrays

- Applicative stores:

\[
\begin{align*}
\text{store}(a, i, v)[i] &= v \\
i \neq j \Rightarrow \text{store}(a, i, v)[j] &= a[j]
\end{align*}
\]

- Or, special combinator:

\[
\text{store}(a, i, v) = \lambda j \cdot \text{if } i = j \text{ then } v \text{ else } a[j]
\]

- **Note**: \(a[j]\) is short for \(\text{select}(a, j)\)
A program using arrays

```c
void swap(int[] a, int i, int j) {
    int tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}
```

\[ a_{\text{new}} = \text{store}(\text{store}(a, i, a[j]), j, a[i])) \]

To prove:

\[ a_{\text{new}}[j] = a[i], \quad a_{\text{new}}[i] = a[j] \]
Special combinator:

\[
store(a, i, v) = \lambda j . \text{if } i = j \text{ then } v \text{ else } a[j]
\]

Existential fragment is decidable by reduction to congruence closure using finite set of instances.

Idea: models for \( \text{select } (\_ [j]) \) are finite maps
What else are arrays?

- Special combinators:

\[
\text{store}(a, i, v) = \lambda j. \text{if } i = j \text{ then } v \text{ else } a[j]
\]

\[
K(v) = \lambda j . v
\]

\[
\text{map}_f(a, b) = \lambda j . f(a[j], b[j])
\]

- Result: Existential fragment is decidable and in \( NP \) by reduction to congruence closure using finite set of instances.
Extra special combinators:

\[
\text{store}(a, i, v) = \lambda j. \begin{cases} 
    v & \text{if } i = j \\
    a[j] & \text{else} 
\end{cases} \\
K(v) = \lambda j. v \\
\text{map}_f(a, b) = \lambda j. f(a[j], b[j])
\]

\[I = \lambda i. i\]

**Result:** Existential fragment is not decidable.
... But there are arrays#:

Not everything is undecidable with I.

\[ K(v) = \lambda j.v \]

\[ \text{map}_{ite}(a, b, c) = \lambda j.\text{ite}(a[j], b[j], c[j]) \]

\[ \text{map}_{=} (a, b) = \lambda j.(a[j] = b[j]) \]

\[ I = \lambda j.j \]

Then: \( \text{store}(a, i, v) = \text{map}_{ite}(\text{map}_{=} (K(i), I), K(v), a) \)

Theory of arrays# is decidable.
Can I access a *default* array value?

\[ \delta(a) \rightarrow \text{default} \]

\[ \delta(K(v)) = v \]

\[ \delta\left(\text{map}_f(a, b)\right) = f(\delta(a), \delta(b)) \]

\[ \delta(\text{store}(a, i, v)) = \delta(a) \quad \text{Only sound for infinite domains} \]
Let’s use CAL:

Simple set and bag operations:

\[
\begin{align*}
\emptyset &= K(\text{false}) \\
\{a\} &= \text{store}(\emptyset, a, \text{true}) \\
a \in A &= A[a] \\
A \cup B &= \text{map}_\lor(A, B) \\
A \cap B &= \text{map}_\land(A, B) \\
f\text{inite}(A) &= (\delta(A) = \text{false})
\end{align*}
\]

\[
\begin{align*}
\emptyset &= K(0) \\
\{a\} &= \text{store}(\emptyset, a, 1) \\
\text{multi}(a, A) &= A[a] \\
A \cup B &= \text{map}_+(A, B) \\
A \cap B &= \text{map}_{\text{min}}(A, B) \\
f\text{inite}(A) &= (\delta(A) = 0)
\end{align*}
\]

But not cardinality \(|A|\), power-set \(2^A\), …
A reduction approach

\[ \text{Sat}(T_{\text{Array}} \land \varphi) ? \]

Use saturation rules to reduce arrays to the theory of un-interpreted functions

\[ \text{Sat}(T_{\text{Equality}} \land \text{Closure}_{\text{Array}})(\varphi) \land \varphi) ? \]

Extract models for arrays as finite graphs
Idea: Saturate a state using proof rules.
A state contains:

- **Definitions**: \( a \equiv t \) – \( a \) is a name for term \( t \)
  
  Example: \( a \equiv \text{store}(b, i, 5) \)
  
  Example: \( r \equiv a \approx b, q \equiv i \approx j \)

- **Assertions**:
  
  Example: \( r, \neg q \)

- **Equalities**: \( a \sim b \) – \( a \) and \( b \) are equal

Result of proof rule is a consequence *clause*. 
Array Saturation Rules

- $a \sim b - a$ and $b$ are equal in current context
- $a \equiv t - a$ is a name for the term $t$
Congruence Closure

\[ w_1 \equiv f(v_1, \ldots, v_n), \ w_2 \equiv f(v'_1, \ldots, v'_n), \ v_1 \sim v'_1, \ldots, v_n \sim v'_n \]

\[ w_1 \simeq w_2 \]
**Bottlenecks**

- **Bottleneck:** Extensionality axiom is instantiated on every pair of array variables.

\[
\text{ext} \quad a : (\sigma \Rightarrow \tau), \quad b : (\sigma \Rightarrow \tau) \\
\frac{a \simeq b \lor a[k_{a,b}] \neq b[k_{a,b}]}{a \simeq b \lor a[k_{a,b}] \neq b[k_{a,b}]}
\]

- **Bottleneck:** Upwards propagation distributes index over all modifications of same array.

\[
\uparrow \quad a \equiv \text{store}(b, i, v), \quad w \equiv b'[j], \quad b \sim b' \\
i \simeq j \lor a[j] \simeq b[j]
\]
Bottlenecks and

**Bottleneck:** Extensionality axiom is instantiated on every pair of array variables.

**Optimization:** Restrict to variables asserted different, or shared.
Bottleneck: Upwards propagation distributes index over all modifications of same array.

Optimization: Only use for updates where ancestor has multiple children.

Formulas from programs are well-behaved.
Non-linear arithmetic

One of key ideas: Use partial solution to guide the search

Feasible Region

\(-4xy - 4x + y > 1\)

What is the core?

\(x^2 + y^2 < 1\)

Starting search
Partial solution: \(x = 0.5\)

Can we extend it to \(y\)?

\(x^3 + 2x^2 + 3y^2 - 5 < 0\)
Quantifiers
Equality-Matching

\[ p(\forall \ldots) \land a = g(b, b) \land b = c \land f(a) \neq c \land p(\forall x \ldots) \rightarrow f(g(b, b)) = b \]

\( g(c, x) \) matches \( g(b, b) \) with substitution \([x \mapsto b]\) modulo \( b = c \)

[de Moura, B. CADE 2007]
Quantifier Elimination

```scheme
(define-fun stamp () Bool
  (forall ((x Int))
    (=>
      (>= x 8)
      (exists ((u Int) (v Int))
        (and (>= u 0) (>= v 0)
          (= x (+ (* 3 u) (* 5 v))))))))
(simplify stamp)
(elim-quantifiers stamp)
```

Presburger Arithmetic,
Algebraic Data-types,
Quadratic polynomials

SMT integration to prune branches

[B. IJCAR 2010]
MBQI: Model based Quantifier Instantiation

(set-option :mbqi true)
(declare-fun f (Int Int) Int)
(declare-const a Int)
(declare-const b Int)

(assert (forall ((x Int)) (>= (f x x) (+ x a))))

(assert (< (f a b) a))
(assert (> a 0))
(check-sat)
(get-model)

(echo "evaluating (f (+ a 10) 20)"
(eval (f (+ a 10) 20))

[de Moura, Ge. CAV 2008]
[Bonachnia, Lynch, de Moura CADE 2009]
[de Moura, B. IJCAR 2010]
Superposition

1. $\forall x . (x; id) = x$
2. $\forall x . (id; x) = x$
3. $\forall x . (id \mid x) = x$
4. $\forall x y z u. (x \mid y); (z \mid u) \leq (x; z) \mid (y; u)$

$\forall p q . (p; q) \leq (p \mid q)$

5. $\forall x z u . x; (z \mid u) \leq (x; z) \mid (id; u)$ super-pose 1, 4
6. $\forall x z u . x; (z \mid u) \leq (x; z) \mid u$ super-pose 2,5
7. $\forall x z u . x; u \leq (x; id) \mid u$ super-pose 3,6
8. $\forall x z u . x; u \leq x \mid u$ super-pose 1,7

[de Moura, B. IJCAR 2008]

(disabled in release 4.1)
Solving Fixed-points

[Hoder, B SAT 2012]
mc(x) = x - 10 \quad \text{if } x > 100

mc(x) = mc(mc(x+11)) \quad \text{if } x \leq 100

assert (mc(x) \geq 91)
Formulate as Horn clauses.

\[
\forall X. \; X > 100 \rightarrow mc(X, X - 10)
\]
\[
\forall X, Y, R. \; X \leq 100 \land mc(X + 11, Y) \land mc(Y, R) \rightarrow mc(X, R)
\]
\[
\forall X, R. \; mc(X, R) \rightarrow R \geq 91
\]

Solve for \( mc \)
Formulate as Predicate Transformer:

$$ (mc)(X,R) = \begin{cases} X > 100 \land R = X - 10 \\ \lor X \leq 100 \land \exists Y. \ mc(X + 11, Y) \land mc(Y, R) \end{cases} $$

Check: $$ \mu (mc)(X,R) \rightarrow R \geq 91 $$
Instead of computing $\mu_{\mathcal{F}}(mc)(X,R)$, then checking $\mu_{\mathcal{F}}(mc)(X,R) \rightarrow R \geq 91$

Suffices to find post-fixed point $mc_{post}$ satisfying:

$$\forall X, R. \quad \mathcal{F}(mc_{post})(X, R) \rightarrow mc_{post}(X, R)$$

$$\forall X, R. \quad mc_{post}(X, R) \rightarrow R \geq 91$$
Program Verification (Safety)
as Solving fixed-points
as Satisfiability of Horn clauses
Procedures $\Rightarrow$ Horn Formulas

procedure $\pi$:
  requires $P(X)$;
  ensures $Q(X_{\text{old}},X)$;
  $\sigma$

assert $P(X)$;
$X_{\text{old}} := X$;
havoc $X$;
assume $Q(X_{\text{old}},X)$;

assert $R(X)$;
call $\pi$;
assert $S(X)$;

Verifying procedure calls

$R(X) \Rightarrow P(X)$

$(R(X_{\text{old}}) \land Q(X_{\text{old}},X) \Rightarrow S(X))$
Init \rightarrow Initial\ condition

Safe \rightarrow Safety\ assertion

\rho_i(X_i, Y, X'_i, Y') \rightarrow Transition\ relation\ of\ process\ i

R_i(X_i, Y) \rightarrow Summary\ of\ process\ i

E_i(Y, Y') \rightarrow Summary\ of\ process\ i's\ environment

Init \Rightarrow R_i(X_i, Y)

R_i(X_i, Y) \land \rho_i(X_i, Y, X'_i, Y') \Rightarrow R_i(X'_i, Y')

R_i(X_i, Y) \land E_i(Y, Y') \Rightarrow R_i(X_i, Y')

R_i(X_i, Y) \land \rho_i(X_i, Y, X'_i, Y') \Rightarrow E_j(Y, Y') \quad j \neq i

R_1(X_1) \land \ldots \land R_N(X_N) \Rightarrow Safe

[Predicate\ Abstraction\ and\ Refinement\ for\ Verifying\ Multi-Threaded\ Programs
Ashutosh\ Gupta,\ Corneliu\ Popeea,\ Andrey\ Rybalchenko,\ POPL\ 2011]
\[ \Gamma \vdash \{ x: \tau \mid P(x) \} \rightarrow \{ y: \sigma \mid Q(x, y) \} < \{ x: \tau \mid P'(x) \} \rightarrow \{ y: \sigma \mid Q'(x, y) \} \]

Extract sufficient Horn Conditions

\[ \Gamma \land P'(x) \Rightarrow P(x) \]

\[ \Gamma \land P'(x) \land Q(x, y) \Rightarrow Q'(x, y) \]
In a nutshell, solving partial correctness amounts to checking truth value of formulas of the form:

$$\exists \bar{P} \bigwedge \forall \bar{x} \ (P_i(\bar{x}) \land P_j(\bar{x}) \land \phi(\bar{x}) \Rightarrow P_k(\bar{x}))$$

E.g., satisfiability of:

$$\bigwedge \forall \bar{x} \ (P_i(\bar{x}) \land P_j(\bar{x}) \land \phi(\bar{x}) \Rightarrow P_k(\bar{x}))$$
Handling background axioms:

\[ \forall \vec{R}, \vec{f} . \text{Background}[\vec{R}, \vec{f}] \Rightarrow \exists \vec{P} \bigwedge \forall \vec{x} (P_i(\vec{x}) \land P_j(\vec{x}) \land \phi(\vec{R}, \vec{f}, \vec{x}) \Rightarrow P_k(\vec{x})) \]

**Remark:**
Abductive Logic Programming *amounts to* symbolic simulation:
- *Program + Abducibles* \(\models\) \(\exists\text{ans} . \text{Query}(\text{ans})\)
- *Abducibles + Integrity Constraints* is consistent

eg. solve for negation of above formula:
\[ \exists \text{Ab} . \text{IC}(\text{Ab}) \land (\forall P . \text{Program}(\text{Ab}, P) \rightarrow \exists \text{ans} . \text{Query}(\text{ans, Ab, P})) \]
A New PDR Engine for Fixedpoints

PDR (aka. IC3) – Property Directed Reachability algorithm

**Breakthrough** in Symbolic Model Checking of Hardware [Aaron Bradley, VMCAI 2011]

Original Algorithm Description in code. Tough to digest. Rule + strategy description could help deconstruct the steps.

Original Algorithm Applies to Hardware (Finite State Automata). Software has procedure calls.

Original Algorithm is for Finite State Systems

Open question what it meant to incorporate Infinite State systems (= theories)

[Hoder & Bjørner, SAT 2012]
Objective is to solve for $R$ such that

$$\mathcal{F}(R)(X) \rightarrow R(X) \quad R(X) \rightarrow Safe(X), \forall X$$

Elements of PDR encoded as transitions:

Over-approximate reachable states

$$R_0 := \mathcal{F}(\text{false}), R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_N$$

Search for counter-examples to $Safe$

Resolve and Propagate conflicts
Objective is to solve for $R$ such that

\[ \mathcal{F}(R)(X) \rightarrow R(X) \quad R(X) \rightarrow Safe(X), \quad \forall X \]

Initialize:

- \( R_0 := \mathcal{F}(false) \)
- \( R_1 := true \)

Main invariant:

- \( R_i \rightarrow Safe \rightarrow \mathcal{F}(R_i) \)
- \( R_{i+1} \rightarrow \mathcal{F}(R_i) \)
Is \( \forall x . (R(x) \equiv \mu X . \mathcal{F}(X)(x)) \rightarrow \forall y . R(y) \rightarrow S(y) \) valid?

Search for over-approximations of states
Recall:

\[ \mathcal{F}(mc)(X,R) = \begin{cases} 
X > 100 \land R = X - 10 \\
\lor X \leq 100 \land \exists Y. \ mc(X + 11, Y) \land mc(Y, R)
\end{cases} \]

Is \( mc(X, R) \land R < 91 \) feasible?

Start with summary \( mc(Y, R) := \text{true} \)

Is \( \text{true} \land R < 91 \) feasible?

Yes, e.g., \( R = 90 \models R < 91 \)

Is \( R = 90 \) reachable? (in \( \mu \mathcal{F}(mc) \))
Non-linear transformers

\[ M(87) = M(M(98)) = M(M(M(109))) = M(M(99)) = M(M(M(110))) = M(M(100)) = M(M(M(M(111)))) = M(M(101)) = M(91) = M(M(102)) = M(92) = M(M(103)) = M(93) \]

Checking against \( R_1, R_2, \ldots, R_N \) controls depth, but potentially wide tree. Our approach: build DAG by sharing states. Sharing is cheap, even no sharing works on Bebop benchmarks from the SLAM Research toolkit.
Search: Mile-high perspective

Modern SMT solver

Decisions: Assignments

Conflict Resolution

Conflict Clauses

Init $\rightarrow$ SP(Init) $\rightarrow$ WP

Bad $\leftarrow$ (Bad) $\leftarrow$ Bad

Fixedpoint solver
Arithmetic

Initially $y_1 := y_2 := 0$;

\[
P_1 :: \begin{cases}
\ell_0 : y_1 := y_2 + 1; \\
\ell_1 : \text{await } y_2 = 0 \lor y_1 \leq y_2;
\end{cases} \quad \parallel \quad P_2 :: \begin{cases}
\ell_0 : y_2 := y_1 + 1; \\
\ell_1 : \text{await } y_1 = 0 \lor y_2 \leq y_1;
\end{cases}
\]

$R(0,0,0,0)$. 

$T(L,M,Y_1,Y_2,L',M',Y_1',Y_2') \land R(L,M,Y_1,Y_2) \rightarrow R(L',M)$

$R(2,2,Y_1,Y_2) \rightarrow \text{false}$

Step($L,L',Y_1,Y_2,Y_1'$) $\rightarrow$ $T(L,M,Y_1,Y_2,L',M',Y_1',Y_2)$

Step($M,M',Y_2,Y_1,Y_2'$) $\rightarrow$ $T(L,M,Y_1,Y_2,L,M',Y_1,Y_2')$

Step($0,1,Y_1,Y_2,Y_2+1$).

$(Y_1 \leq Y_2 \lor Y_2 = 0) \rightarrow$ Step($1,2,Y_1,Y_2,Y_1$).

Step($2,3,Y_1,Y_2,Y_1$).

Step($3,0,Y_1,Y_2,0$).

\[\ell_0: y := \hat{y} + 1; \quad \text{goto } \ell_1\]

\[\ell_1: \text{await } \hat{y} = 0 \lor y \leq \hat{y}; \quad \text{goto } \ell_2\]

\[\ell_2: \text{critical}; \quad \text{goto } \ell_3\]

\[\ell_3: y := 0; \quad \text{goto } \ell_0\]
Search: Mile-high perspective
PDR(T): Conflict Resolution

initially \( y_1 := y_2 := 0 \);

\[ P_1 := \begin{cases} 
\ell_0 : y_1 := y_2 + 1; \\
\ell_1 : \text{await } y_2 = 0 \lor y_1 \leq y_2; \\
\ell_2 : \text{critical}; \\
\ell_3 : y_1 := 0;
\end{cases} \]

\[ || P_2 := \begin{cases} 
\ell_0 : y_2 := y_1 + 1; \\
\ell_1 : \text{await } y_1 = 0 \lor y_2 \leq y_1; \\
\ell_2 : \text{critical}; \\
\ell_3 : y_2 := 0;
\end{cases} \]

Conflict Resolution

\( L = 0 \)
\( M = 0 \)
\( Y_2 = 0 \)
\( Y_1 = 0 \)

\( L = 0 \)
\( M = 1 \)
\( Y_2 = 0 \)

\( L = 1 \)
\( M = 1 \)
\( Y_1 = 1 \)
\( Y_2 = 0 \)

\( L = 1 \)
\( M = 2 \)
\( Y_1 = 0 \)
\( Y_2 = 0 \)

\( L = 2 \)
\( M = 2 \)

\( Y_2 \geq Y_1 + 1 \land Y_1 \geq 0 \)
\( Y_2 \leq 0 \)
\( Y_2 \geq 1 \land Y_2 \leq 0 \)

Get Generalization from Farkas Lemma

Eg., resolve away blue internal variables
**PDR(T): Generalization from T-lemmas**

**Can we satisfy?**

\[ R(0, 0, 0, 0). \]

**Initial states**

\[ T(L, M, Y_1, Y_2, L', M', Y_1', Y_2'), R(L, M, Y_1, Y_2) \rightarrow R(L', M', Y_1', Y_2') \]

**Reachable states**

\[ R(L, M, Y_1, Y_2) \rightarrow \neg(L = 2 \land M = 2). \]

**Unsafe state is unreachable**

\[ L = 0 \land M = 1 \land Y_2 = 0 \quad \text{is unsatisfiable} \]

\[ \exists_j c_j \leq x_j \leq c_j \quad \text{is unsatisfiable} \]

\[ \neg M \lor 3 < x_3 \lor x_1 - x_2 > 2 \lor 2x_2 - x_3 > 1 \]

\[ \text{From } \neg M \quad \text{From } \neg \text{Pre} \]

**E.g., there is unsat core of:**

\[ \neg M \lor 3 < x_3 \lor x_1 - x_2 > 2 \lor 2x_2 - x_3 > 1 \]

\[ \text{From } \neg \text{Pre} \]

**Unsat proof uses T-lemmas**
Can we satisfy?
\[ R(0, 0, 0, 0). \]

Initial states
\[ T(L, M, Y1, Y2, L', M', Y1', Y2'), R(L, M, Y1, Y2) \rightarrow R(L', M', Y1', Y2') \]

Reachable states
\[ R(L, M, Y1, Y2) \rightarrow \neg (L = 2 \land M = 2). \]

Unsafe state is unreachable

Unsat proof uses T-lemmas

\[
\left( 5 > x_1 \vee 3 < x_3 \vee x_1 - x_2 > 2 \vee 2x_2 - x_3 > 1 \right) \\
\text{From } \neg M \\
\text{From } \neg \text{Pre}
\]

\[
2 \cdot (\neg x_1 \leq -5) \\
x_3 \leq 3 \\
2 \cdot (x_1 - x_2 \leq 2) \\
2x_2 - x_3 \leq 1 \\
\]

\[
-2x_1 \leq -10 \\
x_3 \leq 3 \\
2x_1 - 2x_2 \leq 4 \\
2x_2 - x_3 \leq 1 \\
2x_1 - x_3 \leq 5 \\
\text{Block any model satisfying this}
\]

\[
2x_1 - x_3 \leq 5 \\
0 \leq -2
\]
Observation:

**PDR + Model refinement using Farkas strengthening is a decision procedure for timed push-down systems**

Justification:

Every lemma produced is a sum of differences from the input

~

Acyclic path in difference graph.

⇒ Finite set of Farkas lemmas possible.
Objective:
synthesize inductive invariant proving property.

Reaching objective with **interpolants**:
- Synthesize interpolants, use for proving invariants. Be admired.
- Synthesize interpolants, evaluate on random formulas. Admire them.
- Write papers about interpolants. Admire the theorems.
- Review papers about generating interpolants. Watch Kevin Bacon.

Reaching objective with PDR:
.... Nevertheless, interpolants sneak in.
What is a Craig Interpolant?

Suppose $A \Rightarrow B$

A Craig Interpolant is formula $I$:

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

$$A \Rightarrow I, I \Rightarrow B$$

Horn version. Establish satisfiability of:

$$\forall x, y. A[x, y] \Rightarrow I(x),$$

$$\forall x, z. I(x) \Rightarrow B[x, z]$$

and find solution for $I$. 
**PDR(T): Interpolants as a side-effect**

**Intermediary solutions:**

\[
\forall X. \ (\mathcal{F}(\text{false})(X) \rightarrow R_1(X), \\
\forall X. \ (\mathcal{F}(R_1)(X) \rightarrow R_2(X), \\
\forall X. \ (\mathcal{F}(R_2)(X) \rightarrow R_3(X), \\
\forall X. \ R_3(X) \rightarrow \text{Safe}(X),
\]

**Observation:**

Farkas strengthening computes a “DAG interpolant” for LRA

\[\text{i.e., solves for non-recursive Horn clauses}\]
Symbolic Software Model Checking as:

Quantified **Horn Clause Satisfiability** Modulo Theories

PDR Generalized:
- as an abstract Transition System
- for Horn Clause Satisfiability over Theory of Arithmetic

- Using Farkas to generalize failed counter-example traces
- Difference Logic – a Model Checking algorithm for Timed Automata
- Interpolants from Model refinements
- Propagate also properties for predicates (**so far inefficient**)

I = IntSort()
B = BoolSort()
10 = Function('10', I, I, B)
11 = Function('11', I, I, B)

s = Fixedpoint()
s.register_relation(10, 11)
s.set_predicate_representation(10, 'interval_relation', 'bound_relation')
s.set_predicate_representation(11, 'interval_relation', 'bound_relation')

m, x, y = Ints('m x y')
s.declare_var(m, x, y)
s.rule(10(0,m), 0 < m)
s.rule(10(x+1,m), [10(x,m), x < m])
s.rule(11(x,m), [10(x,m), m <= x])

print "At 10 we learn that x, y are non-negative:"
print s.query(10(x,y))
print s.get_answer()

print "At 11 we learn that x <= y and both x and y are bigger than 0:"
print s.query(11(x,y))
print s.get_answer()

print "The state where x < y is not reachable"
print s.query(And(11(x,y), x < y))

The example uses the option
set_option(dl_compile_with_widening=True)

set_option(dl_engine=1, dl_pdr_use_farkas=True)
mc = Function('mc', IntSort(), IntSort(), BoolSort())
n, m, p = Ints('n m p')

fp = Fixedpoint()
fp.declare_var(n,m)
fp.register_relation(mc)

fp.rule(mc(m, m-10), m > 100)
fp.rule(mc(m, n), [m <= 100, mc(m+11,p), mc(p,n)])

print fp.query(And(mc(m,n),m < 100, n != 91))
print fp.get_answer()

print fp.query(And(mc(m,n),n < 91))
print fp.get_answer()
Deduction with SMT solvers is central to software analysis, testing and verification tools. Many problems can be reduced to an SMT problem.

Research on SMT:
- Efficient algorithms, data-structures, programming
- Raise the bar of supported theories
- Raise the bar of reduction (e.g., model checking as SMT)

Common algorithmic traits:
- Cooperation between model and proof search.