Model Checking Software, Solving Horn Clauses and IC3

Nikolaj Bjørner
Microsoft Research
SAT/SMT Summer School 2014, Semmering
Takeaways

Symbolic Software Model Checking =
SMT for Horn Clauses

Simplification and pre-processing

A Solver using clues from IC3

An Algorithmic Overview (separate slides)
Horn Clauses

mc(x) = x-10 \quad \text{if } x > 100
mc(x) = mc(mc(x+11)) \quad \text{if } x \leq 100

assert (x \leq 101 \rightarrow mc(x) = 91)

\forall X. \quad X > 100 \rightarrow mc(X, X - 10)
\forall X, Y, R. \quad X \leq 100 \land mc(X + 11, Y) \land mc(Y, R) \rightarrow mc(X, R)
\forall X, R. \quad mc(X, R) \land X \leq 101 \rightarrow R = 91

Solver finds solution for mc

[Hoder, B. SAT 2012]
Transition System

- \( V \) - program variables
- \( \text{init}(V) \) - initial states
- \( \text{step}(V, V') \) - transition relation
- \( \text{safe}(V) \) - safe states
Safe Transition System

∃Inv.

• ∀V. init(V) → Inv(V)
• ∀V, V'. Inv(V) ∧ step(V, V') → Inv(V')
• ∀V. Inv(V) → safe(V)

— [Rybalchenko et.al. PLDI 2012, POPL 2014]
Termination and reactivity are also handled in framework of solving systems of logical formulas.
Recursive Procedures

Formulate as Horn clauses:

\[ \forall X. \ X > 100 \rightarrow mc(X, X - 10) \]
\[ \forall X, Y, R. \ X \leq 100 \land mc(X + 11, Y) \land mc(Y, R) \rightarrow mc(X, R) \]
\[ \forall X, R. \ mc(X, R) \land X \geq 101 \rightarrow R = 91 \]

Solve for mc
Recursive Procedures

Formulate as Predicate Transformer:

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

Check: \( \mu \mathcal{F}(mc)(X,R) \land X \geq 101 \rightarrow R = 91 \)
Recursive Procedures

Instead of computing $\mu_F (\text{mc})(X,R)$, then checking $\mu_F (\text{mc})(X,R) \land X \leq 101 \rightarrow R = 91$

Suffices to find post-fixed point $\text{mc}_{post}$ satisfying:

$\forall X, R. \ F(\text{mc}_{post})(X, R) \rightarrow \text{mc}_{post}(X, R)$

$\forall X, R. \ \text{mc}_{post}(X, R) \land X \leq 101 \rightarrow R = 91$
Program Verification as SMT - aka
A Crusade for Hornish Satisfaction

Program Verification (Safety)

as Solving fixed-points

as Satisfiability of Horn clauses

[Bjørner, McMillan, Rybalchenko, SMT workshop 2012]
Hilbert Sausage Factory: [Grebenshchikov, Lopes, Popeea, Rybalchenko, PLDI 2012]
Procedures $\Rightarrow$ Horn Formulas

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

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

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

$R(X) \Rightarrow P(X)$
$(R(X_{old}) \wedge Q(X_{old}, X) \Rightarrow S(X))$
Modular Concurrency $\implies$ Horn Clauses

Init – Initial condition

Safe – Safety assertion

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

$R_i(X_i,Y)$ – Summary of process $i$

$E_i(Y,Y')$ – Summary of process $i$'s environment

$Init \Rightarrow R_i(X_i,Y)$

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

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

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

$R_1(X_1) \land \cdots \land R_N(X_N) \Rightarrow Safe$

[Predicate Abstraction and Refinement for Verifying Multi-Threaded Programs
Gupta, Popeea, Rybalchenko, POPL 2011]
Liquid Types $\Rightarrow$ Horn Clauses

\[
\Gamma \vdash \{ x: \tau \mid P(x) \} \rightarrow \{ y: \sigma \mid Q(x, y) \} < \{ x: \tau \mid P'(x) \} \rightarrow \{ y: \sigma \mid Q'(x, y) \}
\]

Extract sufficient Horn Conditions

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

\[
\Gamma \land P'(x) \land Q(x, y) \Rightarrow Q'(x, y)
\]
Generalized Horn Formulas

Handling background axioms:

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

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

eg. solve for negation of above formula:
\[ \exists \text{Ab}.\text{IC} (\text{Ab}) \wedge (\forall P . \text{Program}(\text{Ab}, P) \rightarrow \exists \text{ans} \cdot \text{Query(\text{ans}, \text{Ab}, P})) \]
sv-benchmarks — Revision 213
/trunk/clauses/ALIA/sdv

[Parent Directory]

sdv0.smt2
sdv1.smt2
sdv10.smt2
sdv100.smt2
sdv1000.smt2
sdv1001.smt2
sdv1002.smt2
sdv1003.smt2
sdv1004.smt2
sdv1005.smt2
sdv1006.smt2
sdv1007.smt2
sdv1008.smt2
sdv1009.smt2
sdv101.smt2
sdv1010.smt2
sdv1011.smt2
sdv1012.smt2
sdv1013.smt2
sdv1014.smt2
sdv1015.smt2
Simplifying Horn Clauses
A model checking Example

Program 1.4.1 Processing requests using locks.

do {
    lock();
    old_count = count;
    request = GetNextRequest();
    if (request != NULL) {
        ReleaseRequest(request);
        unlock();
        ProcessRequest(request);
        count = count + 1;
    }
} while (old_count != count);
unlock();
Abstraction as Boolean Program

Program 1.4.2 Processing requests using locks, abstracted.

1. do {
2.  lock();
3.  b = true;
4.  if (*) {
5.    unlock();
6.    if (b) {
7.      b = false;
8.    }
9.  } else {
10.   havoc b;
11.  }
12. }
13. while (!b);
14. unlock();

b := count == old_count

[SLAM, BLAST, Graf & Saidi, Uribe, ..]
(Predicate) Abstraction/Refinement

• SMT solver used to synthesize (strongest) abstract transition relation $F$:

$$
\rho(\tilde{x}, \tilde{x}') \Rightarrow F(b_1(\tilde{x}) \ldots, b_n(\tilde{x}), b_1(\tilde{x}') \ldots, b_n(\tilde{x}'))
$$
Control as Horn Clauses

(set-logic HORN)
(declare-fun Loop (Int Int Bool) Bool)
(declare-fun WhileTest (Int Int Bool) Bool)

; Loop is entered in arbitrary values of count, old_count
(assert (forall ((count Int) (old_count Int))
    (Loop count old_count false)));

; Loop without if test
(assert (forall ((count Int) (old_count Int)) (lock_state Bool))
    (=> (Loop count old_count lock_state) (WhileTest count count true))));

; Loop with if-test
(assert (forall ((count Int) (old_count Int) (lock_state Bool)))
    (=> (Loop count old_count lock_state) (WhileTest (+ 1 count) count false)));)

(assert (forall ((count Int) (old_count Int) (lock_state Bool))
    (=> (and (= old_count count) (= lock_state true)))
    (Loop count old_count lock_state)));

(assert (forall ((count Int) (old_count Int) (lock_state Bool))
    (=> (and (= old_count count) (= lock_state true))
        (= lock_state false)));

(while (old_count != count)
    unlock();
)

(check-sat)
(get-model)
Solving Horn Clauses

Pre-processing

\[ \text{HornClauses} \rightarrow \text{HornClauses}' \]

Search

- Find model \( M \) such that \( M \models \text{HornClauses} \)
- Find refutation proof \( \pi: \text{HornClauses} \vdash_\pi \bot \)
Pre-processing

- Cone of Influence
- Simplification
- Subsumption
- Inlining
- Slicing
- Unfolding
Cone of Influence – top down

\[ P(x) \land Q(y) \rightarrow \text{false} \]

\[ R(x) \land x > 0 \rightarrow P(x) \]
\[ R(x) \land x < 0 \rightarrow P(x) \]

\[ x = 2y \rightarrow R(x) \]

\[ Q(y) \land y \leq x \rightarrow Q(x) \]

\[ P(x) \rightarrow S(x) \]
\[ T(x) \rightarrow S(x) \]

\[ S \text{ is not used} \]
\[ S(x) \equiv \text{true} \]
Cone of Influence – bottom up

\[ P(x) \land Q(y, 0) \rightarrow false \]

\[ R(x) \land x > 0 \rightarrow P(x) \]
\[ R(x) \land x < 0 \rightarrow P(x) \]

\[ x = 2y \rightarrow R(x) \]

\[ Q(y, z) \land y \leq x \rightarrow Q(x, 1) \]

There is no “rule to produce \( Q(x, 1) \)”

\[ Q(x, y) := y = 1 \]
(set-logic HORN)
(declare-fun Loop (Int Int Bool) Bool)
(declare-fun WhileTest (Int Int Bool) Bool)

; Loop is entered in arbitrary values of count, old_count
(assert (forall ((count Int) (old_count Int))
   (Loop count old_count false)))

; Loop without if test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
   (=> (and (Loop count old_count lock_state) (not (= count count)))
        (Loop count count true)))

; Loop without if test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
   (=> (and (Loop count old_count lock_state) (= count count))
        (= true true)))

; Loop with if-test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
   (=> (and (not (= old_count count)) (WhileTest count old_count lock_state))
        (Loop count old_count lock_state))))

; Loop with if-test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
   (=> (and (Loop count old_count lock_state) (= (+ 1 count) count )
        (= lock_state true)))

(assump (forall ((count Int) (old_count Int) (lock_state Bool))
   (=> (Loop count old_count lock_state) (= lock_state false))))

(check-sat)
(get-model)
(set-logic HORN)
(declare-fun Loop (Int Int Bool) Bool)
(declare-fun WhileTest (Int Int Bool) Bool)

; Loop is entered in arbitrary values of count, old_count
(assert (forall ((count Int) (old_count Int))
    (Loop count old_count false)))

; Loop without if test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
    (=> (and (Loop count old_count lock_state) (not (= count count)))
        (Loop count count true)))

; Loop without if test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
    (=> (and (Loop count old_count lock_state) (= count count))
        (= true true)))

; Loop with if-test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
    (=> (and (Loop count old_count lock_state) (not (= (+ 1 count) count ))
        (Loop (+ 1 count) count false)))

; Loop with if-test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
    (=> (and (Loop count old_count lock_state) (= (+ 1 count) count )
        (= false true)))

(assert (forall ((count Int) (old_count Int) (lock_state Bool))
    (=> (Loop count old_count lock_state)
        (= lock_state false))))

(check-sat)
(get-model)
Simplification
(set-logic HORN)
declare-fun Loop (Int Int Bool) Bool
declare-fun WhileTest (Int Int Bool) Bool

; Loop is entered in arbitrary values of count, old_count
(assert (forall ((count Int) (old_count Int))
  (Loop count old_count false)))

; Loop without if test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (not (= count count))
    (Loop count count true))))

; Loop without if test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (= count count)
    (= true true))))

; Loop with if-test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (not (= (+ 1 count) count))
    (Loop (+ 1 count) count false))))

; Loop with if-test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (= (+ 1 count) count)
    (= false true))))

(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (Loop count old_count lock_state)
    (= lock_state false))))

(check-sat)
(get-model)
Cone of Influence

(set-logic HORN)
(declare-fun Loop (Int Int Bool) Bool)
(declare-fun WhileTest (Int Int Bool) Bool)

; Loop is entered in arbitrary values of count, old_count
(assert (forall ((count Int) (old_count Int))
  (Loop count old_count false)))

; Loop without if test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (not (= count count)))
   (Loop count count true)))

; Loop without if test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (= count count))
   (= truetrue)))

; Loop with if-test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (not (= (+ 1 count ) count ))
   (Loop (+ 1 count) count false)))

; Loop with if-test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (= (+ 1 count ) count )
   (= false true)))

(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (Loop count old_count lock_state)
   (= lock_state false))))

(check-sat)
(get-model)
(set-logic HORN)
(declare-fun Loop (Int Int Bool) Bool)
(declare-fun WhileTest (Int Int Bool) Bool)

; Loop is entered in arbitrary values of count, old_count
(declare-fun Loop (Int Int Bool) Bool)
(declare-fun WhileTest (Int Int Bool) Bool)

; Loop is entered in arbitrary values of count, old_count
(assert (forall ((count Int) (old_count Int))
  (Loop count old_count false)))

; Loop without if test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (not (= count count))
            (Loop count count true)))

; Loop without if test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (= count count)
            (= truetrue)))

; Loop with if-test + repeat loop
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (not (= (+ 1 count ) count ))
            (Loop (+ 1 count ) count false)))

; Loop with if-test + loop exit
(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (and (Loop count old_count lock_state) (= (+ 1 count ) count )
            (= false true)))

(assert (forall ((count Int) (old_count Int) (lock_state Bool))
  (=> (Loop count old_count lock_state)
       (= lock_state false)))

(check-sat)
(get-model)
Slicing

• Remove variables that have no effect
Unfolding

• = more aggressive in-lining

• Also applied to predicates that are not eliminated
A Horn Clause Solver
Algorithm

Objective is to solve for $R$ such that

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

Key elements of PDR algorithm:

Over-approximate reachable states

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

Propagate back from $\neg \text{Safe}$

Resolve conflicts

Strengthen/propagate using induction
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:

$R_0 := \mathcal{F}(false)$

Main invariant:
Algorithm

Main invariant:

\[ \mathcal{F}(R_i)(X) \rightarrow R_{i+1}(X) \]

\[ R_i(X) \rightarrow \text{Safe}(X) \]

\[ R_i(X) \rightarrow R_{i+1}(X) \]
Search: Mile-high perspective

Modern SMT solver

Decisions:
- Assignments

Conflict Resolution

Conflict Clauses

Fixedpoint solver

Bad $\rightarrow$ WP(Bad)

Conflict Resolution

(Init) $\leftrightarrow$ Init
Conflict resolution with arithmetic

initially $y_1 := y_2 := 0$;

$P_1 ::= \begin{align*}
\ell_0 & : y_1 := y_2 + 1; \\
\ell_1 & : \text{await } y_2 = 0 \lor y_1 \leq y_2; \\
\ell_2 & : \text{critical}; \\
\ell_3 & : y_1 := 0;
\end{align*}$

$P_2 ::= \begin{align*}
\ell_0 & : y_2 := y_1 + 1; \\
\ell_1 & : \text{await } y_1 = 0 \lor y_2 \leq y_1; \\
\ell_2 & : \text{critical}; \\
\ell_3 & : y_2 := 0;
\end{align*}$

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 \text{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).

\begin{align*}
\ell_0 & : y := \hat{y} + 1; \text{ goto } \ell_1 \\
\ell_1 & : \text{await } \hat{y} = 0 \lor y \leq \hat{y}; \text{ goto } \ell_2 \\
\ell_2 & : \text{critical}; \text{ goto } \ell_3 \\
\ell_3 & : y := 0; \text{ goto } \ell_0
\end{align*}
Search: Mile-high perspective
Interpolating Conflicts

Initially $y_1 := y_2 := 0$;

Loop forever do

$P_1 :=$
- $l_0 : y_1 := y_2 + 1$;
- $l_1 : \text{await } y_2 = 0 \lor y_1 \leq y_2$;
- $l_2 : \text{critical}$;
- $l_3 : y_1 := 0$;

$P_2 :=$
- $l_0 : y_2 := y_1 + 1$;
- $l_1 : \text{await } y_1 = 0 \lor y_2 \leq y_1$;
- $l_2 : \text{critical}$;
- $l_3 : y_2 := 0$;

Conflict Resolution

$L = 0$
$M = 0$
$Y_2 = 0$
$Y_1 = 0$

$L = 0$
$M = 1$
$Y_2 = 0$

$L = 1$
$M = 1$
$Y_1 = 1$
$Y_2 = 0$

$L = 1$
$M = 2$
$Y_1 = 1$
$Y_2 = 0$

$L = 2$
$M = 2$

Conflict

$Y_2 \geq Y_1 + 1 \land Y_1 \geq 0$

Resolution

$Y_2 \leq 0$

Get Generalization from Farkas Lemma

Eg., resolve away blue internal variables
Interpolating Conflicts

Initially $y_1 := y_2 := 0$;

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

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

Conflict Resolution

Conflict Propagation

$L = 0$
$M = 0$
$Y_2 = 0$
$Y_1 = 0$

$L = 1$
$M = 1$
$Y_2 \geq 1$

$L = 1$
$M = 1$
$Y_2 \geq 1$

$L = 2$
$M = 2$
**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**

$R(L, M, Y_1, Y_2) \rightarrow L = 2 \quad M = 2$.

**Unsafe state is unreachable**

$L = 0 \land M = 1 \land Y_2 = 0 \quad \overset{\mathcal{M}}{\land} \overset{\text{Pre}}{\mathcal{F}(R_0)}$ is unsatisfiable

E.g., there is unsat core of:  

$\overset{\mathcal{M}}{\land} c_j \leq x_j \leq c_j \quad \overset{\text{Pre}}{\mathcal{F}(R_i)}$

**Unsat proof uses T-lemmas**

$\left( \overset{\mathcal{M}}{5 > x_1} \lor \overset{\text{From} \neg \mathcal{M}}{3 < x_3} \lor \overset{\text{From} \neg \text{Pre}}{x_1 - x_2 > 2} \lor 2x_2 - x_3 > 1 \right)$
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

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

Unsafe state is unreachable

Unsat proof uses T-lemmas

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

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

satisfying this