

# Solving Quantified Horn Clauses

Andrey Rybalchenko  
Microsoft Research Cambridge and TUM

joint work with

T. Beyene, N. Bjørner, S. Chaudhuri,  
K. McMillan, and C. Popeea

May 28, 2013

# Program Verification as (Horn) Constraint Solving

- ▶ Universal temporal properties, e.g., LTL
- ▶ Quantifier free invariants/auxiliary assertions
- ▶ Proof rule formalization as Horn clauses

# Transition System

- ▶  $v$  - program variables
- ▶  $init(v)$  - initial states
- ▶  $step(v, v')$  - transition relation
- ▶  $safe(v)$  - safe states

# Safety and Termination of Transition System

$\exists inv(v) :$

$$init(v) \rightarrow inv(v)$$

$$inv(v) \wedge step(v, v') \rightarrow inv(v')$$

# Safety and Termination of Transition System

$\exists inv(v) :$

$$init(v) \rightarrow inv(v)$$

$$inv(v) \wedge step(v, v') \rightarrow inv(v')$$

$$inv(v) \rightarrow safe(v)$$

$$wf(inv(v) \wedge step(v, v'))$$

safety

termination

# Transition Invariant for Termination

$\exists ti(v, v') \exists w_1(v, v') \dots \exists w_n(v, v') :$

$inv(v) \wedge step(v, v') \rightarrow ti(v, v')$

$ti(v, v') \wedge step(v', v'') \rightarrow ti(v, v'')$

$ti(v, v') \rightarrow w_1(v, v'') \vee \dots \vee w_n(v, v'')$

# Transition Invariant for Termination

$\exists ti(v, v') \exists w_1(v, v') \dots \exists w_n(v, v') :$

$inv(v) \wedge step(v, v') \rightarrow ti(v, v')$

$ti(v, v') \wedge step(v', v'') \rightarrow ti(v, v'')$

$ti(v, v') \rightarrow w_1(v, v'') \vee \dots \vee w_n(v, v'')$

$wf(w_1(v, v'))$

$\dots$

$wf(w_n(v, v'))$

# Program with procedures

- ▶  $v$  - program variables
- ▶  $init(v)$  - initial states of main procedure
- ▶  $step(v, v')$  - intra-procedural transition relation
- ▶  $safe(v)$  - safe states



# Program with procedures

- ▶  $v$  - program variables
- ▶  $init(v)$  - initial states of main procedure
- ▶  $step(v, v')$  - intra-procedural transition relation
- ▶  $safe(v)$  - safe states
- ▶  $call(v, v')$  - parameter passing relation
- ▶  $ret(v, v')$  - return value passing

# Safety and Termination of Program with Procedures

$\exists \text{sum}(v, v') :$

$\text{init}(v_0) \rightarrow \text{sum}(v_0, v_0)$

$\text{sum}(v_0, v_1) \wedge \text{step}(v_1, v_2) \rightarrow \text{sum}(v_0, v_2)$

$\text{sum}(v_0, v_1) \wedge \text{call}(v_1, v_2) \rightarrow \text{sum}(v_2, v_2)$

$\text{sum}(v_0, v_1) \wedge \text{call}(v_1, v_2) \wedge \text{sum}(v_2, v_3) \wedge \text{ret}(v_3, v_4) \rightarrow \text{sum}(v_0, v_4)$

# Safety and Termination of Program with Procedures

$\exists \text{sum}(v, v') :$

$\text{init}(v_0) \rightarrow \text{sum}(v_0, v_0)$

$\text{sum}(v_0, v_1) \wedge \text{step}(v_1, v_2) \rightarrow \text{sum}(v_0, v_2)$

$\text{sum}(v_0, v_1) \wedge \text{call}(v_1, v_2) \rightarrow \text{sum}(v_2, v_2)$

$\text{sum}(v_0, v_1) \wedge \text{call}(v_1, v_2) \wedge \text{sum}(v_2, v_3) \wedge \text{ret}(v_3, v_4) \rightarrow \text{sum}(v_0, v_4)$

$\text{sum}(v_0, v_1) \rightarrow \text{safe}(v_1)$

$\text{wf}(\exists v_1 : \text{sum}(v_0, v_1) \wedge \text{call}(v_1, v_2))$

# Multi-Threaded Program

- ▶  $v = (g, l_1, l_2)$  - global and thread-local variables
- ▶  $init(v)$  - initial states
- ▶  $safe(v)$  - safe states

# Multi-Threaded Program

- ▶  $v = (g, l_1, l_2)$  - global and thread-local variables
- ▶  $init(v)$  - initial states
- ▶  $safe(v)$  - safe states
- ▶  $step_1(v, v')$  - transition relation of 1st thread, preserves  $l_2$
- ▶  $step_2(v, v')$  - transition relation of 2nd thread, preserves  $l_1$

# Owicki/Gries Rule for Safety

$\exists inv_1(v) \exists inv_2(v) :$

$$init(v) \rightarrow inv_1(v)$$

$$inv_1(v) \wedge step_1(v, v') \rightarrow inv_1(v')$$

$$inv_1(v) \wedge inv_2(v) \wedge step_2(v, v') \rightarrow inv_1(v')$$

...

$$inv_1(v) \wedge inv_2(v) \rightarrow safe(v)$$

Clauses for preservation of  $inv_2(v)$  are symmetric

## Rely/Guarantee Rule for Safety

$\exists inv_1(v) \exists inv_2(v) \exists env_1(v, v') \exists env_2(v, v') :$

$init(v) \rightarrow inv_1(v)$

$inv_1(v) \wedge step_1(v, v') \rightarrow inv_1(v') \wedge env_2(v, v')$

$inv_1(v) \wedge env_1(v, v') \rightarrow inv_1(v')$

...

$inv_1(v) \wedge inv_2(v) \rightarrow safe(v)$

Clauses for preservation of  $inv_2(v)$  are symmetric

# Thread-Modular Rule for Safety

$\exists inv_1(g, l_1) \exists inv_2(g, l_2) \exists env(g, g') :$

$init(v) \rightarrow inv_1(g, l_1)$

$inv_1(g, l_1) \wedge step_1(v, v') \rightarrow inv_1(g', l'_1) \wedge env(g, g')$

...

$inv_1(g, l_1) \wedge inv_2(g, l_2) \rightarrow safe(v)$

Clauses for preservation of  $inv_2(v)$  are symmetric



## Quantifier Free Horn Clauses

$$\forall v \forall w : \textit{body}(v, w) \rightarrow \textit{head}(v)$$

*body*(*v*, *w*) and *head*(*v*) are quantifier free

# Quantified Horn Clauses

- ▶ Existential temporal properties, e.g., CTL
- ▶ Program synthesis and infinite-state game solving

$$\forall v \forall w : \mathit{body}(v, w) \rightarrow \exists x : \mathit{head}(v, x)$$

- ▶ Quantified invariants/auxiliary assertions

$$\forall v \forall w : (\forall y : \mathit{body}(v, w, y)) \rightarrow \mathit{head}(v)$$

## Existentially Quantified Horn Clauses

$$\forall v \forall w : \mathit{body}(v, w) \rightarrow \exists x : \mathit{head}(v, x)$$

*body*(*v*, *w*) and *head*(*v*, *x*) are quantifier free

## Proving CTL Properties

$$(init(v), step(v, v')) \models EF(q(v))$$

$$(init(v), step(v, v')) \models EG(EU(p(v), q(v)))$$

Based on proof system for CTL\* by Kesten and Pnueli [TCS'05]

## Proving $EF(q(v))$

$\exists inv(v) \exists round(v, v') :$

$init(v) \rightarrow inv(v)$

$inv(v) \wedge \neg q(v) \rightarrow \exists v' : step(v, v')$

$\wedge inv(v')$

$\wedge round(v, v')$

$wf(round(v, v'))$

## Decomposing $EG(EU(p(v), q(v)))$

$$(init(v), step(v, v')) \models EG(EU(p(v), q(v)))$$

iff

$\exists mid(v) :$

$$(init(v), step(v, v')) \models EG(mid(v))$$

$$(mid(v), step(v, v')) \models EU(p(v), q(v))$$

Proving  $(init(v), step(v, v')) \models EG(mid(v))$  and  
 $(mid(v), step(v, v')) \models EU(p(v), q(v))$

$\exists mid(v) \exists inv_1(v) \exists inv_2(v) \exists round(v, v') :$

$init(v) \rightarrow inv_1(v)$

$inv_1(v) \rightarrow mid(v) \wedge \exists v' : step(v, v') \wedge inv_1(v')$

$mid(v) \rightarrow inv_2(v)$

$inv_2(v) \wedge \neg q(v) \rightarrow p(v) \wedge \exists v' : step(v, v') \wedge inv_2(v') \wedge round(v, v')$

$wf(round(v, v'))$

# Solving Infinite-State Game

Given five empty bottles arranged in circle and jar full of water

- ▶ Stepmother pours all water from jar into some bottles
- ▶ Cinderella empties pair of adjacent bottles
- ▶ Jar is refilled for next round

Stepmother wins if some bottle overflows



# Formalization of Game Arena

- ▶  $v = (v_1, \dots, v_5)$
- ▶  $B$  - bottle volume
- ▶  $J$  - jar volume

$$\mathit{init}(v) = (v_1 = \dots = v_5 = 0)$$

$$\mathit{cindy}(v, v') = (v'_1 = v'_2 = 0 \wedge \mathit{same}(v_3, v_4, v_5) \vee$$

...

$$\vee v'_5 = v'_1 = 0 \wedge \mathit{same}(v_2, v_3, v_4))$$

$$\mathit{step}(v, v') = (v'_1 \geq v_1 \wedge \dots \wedge v'_5 \geq v_5 \wedge$$

$$v'_1 + \dots + v'_5 - (v_1 + \dots + v_5) = J)$$

$$\mathit{over}(v) = (v_1 > B \vee \dots \vee v_5 > B)$$

# Stepmother's Victory as Constraint Satisfaction

$\exists win(v) \exists round(v, v') :$

$init(v) \rightarrow win(v)$

$win(v) \wedge \neg over(v) \wedge cindy(v, v') \rightarrow \exists v'' : step(v', v'')$

$\wedge win(v'')$

$\wedge round(v, v'')$

$wf(round(v, v'))$

## Universally Quantified Horn Clauses

$$\forall v \forall w : (\forall y : \mathit{body}(v, w, y)) \rightarrow \mathit{head}(v)$$

$\mathit{body}(v, w, y)$  and  $\mathit{head}(v)$  are quantifier free

# Verification with Universally Quantified Invariants

```
for(i = 0; i < n; i++) { a[i] = i; }  
assert("forall p: i <= p && p < n -> a[p] == p");
```

State-of-the-art recipe (e.g., Gopan et al. POPL'05, Gulwani et al. POPL'08, Halbwachs et al. PLDI'08, Dillig et al. ESOP'10, Logozzo et al. POPL'11, Alberti et al. CAV'12, Larraz et al. VMCAI'13):

- ▶ Quantification template
- ▶ Instantiation template
- ▶ Shape template with abstract domains

# Templates for Universally Quantified Invariants

- ▶ Quantification template

$$\forall p : \text{inv}(i, n, p, a(p))$$

- ▶ Instantiation template

$$\text{inv}(i, n, e_1, a(e_1)) \wedge \cdots \wedge \text{inv}(i, n, e_k, a(e_k))$$

- ▶ Shape template with abstract domains

$$\text{inv}(i, n, p, a(p)) = (\text{guard}(i, n, p) \rightarrow \text{property}(i, n, p, a(p)))$$

\*under-approximation for  $\text{guard}(i, n, p)$

# Universally Quantified Clauses

```
for(i = 0; i < n; i++) { a[i] = i; }  
assert("forall p: i <= p && p < n -> a[p] == p");
```

- Quantification template  $\forall p : \text{inv}(i, n, p, a(p))$

$\exists \text{inv}(i, n, p, \underbrace{a(p)}_v) :$

$$i = 0 \rightarrow (\forall p : \text{inv}(i, n, p, a(p)))$$

$$(\forall p : \text{inv}(i, n, p, a(p))) \wedge i < n \wedge i' = i + 1 \wedge a' = a\{i := i'\} \rightarrow$$
$$(\forall q : \text{inv}(i', n, q, a'(q)))$$

$$(\forall p : \text{inv}(i, n, p, a(p))) \rightarrow (\forall q : 0 \leq q < n \rightarrow a(q) = q)$$

# Universally Quantified Clauses

```
for(i = 0; i < n; i++) { a[i] = i; }  
assert("forall p: i <= p && p < n -> a[p] == p");
```

- ▶ Quantification template  $\forall p : inv(i, n, p, a(p))$

$\exists inv(i, n, p, \underbrace{a(p)}_v) :$

$i = 0 \rightarrow inv(i, n, p, a(p))$

$(\forall p : inv(i, n, p, a(p))) \wedge i < n \wedge a' = a\{i := i\} \rightarrow$   
 $inv(i + 1, n, q, a'(q))$

$(\forall p : inv(i, n, p, a(p))) \wedge 0 \leq q < n \rightarrow a(q) = q$

# Quantifier Instantiation Heuristic

## Instantiation Constraint Generation

- ▶ If  $a(p)$  occurs in clause then  $p \in inst(a)$
- ▶ If  $a' = a\{\cdot := \cdot\}$  occurs in clause then  $inst(a') \subseteq inst(a)$

## Example

- ▶ Clause

$$\begin{aligned} & (\forall p : inv(i, n, p, a(p))) \wedge i < n \wedge a' = a\{i := i\} \\ & \rightarrow inv(i + 1, n, q, a'(q)) \end{aligned}$$

- ▶ Instantiation constraints  $q \in inst(a')$  and  $inst(a') \subseteq inst(a)$
- ▶ Instantiation solution  $inst(a) = inst(a') = \{q\}$



# Quantifier Instantiation Validation

$$\text{inv}(i, n, p, v) = (0 \leq p < i \rightarrow v = p) \text{ and } \text{inst}(a) = \text{inst}(a') = \{q\}$$

$$(0 \leq q < i \rightarrow q = a(q))$$

$$\wedge i < n$$

$$\wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q))$$

$$\rightarrow (0 \leq q < i + 1 \rightarrow a'(q) = q)$$

## Instantiation Based Proof

$$(0 \leq q < i \rightarrow a(q) = q) \wedge i < n \wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q))$$

$$(0 \leq q < i + 1 \rightarrow a'(q) = q)$$

## Instantiation Based Proof

$$(0 \leq q < i \rightarrow a(q) = q) \wedge i < n \wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q)) \\ 0 \leq q < i + 1$$

$$a'(q) = q$$

$$(0 \leq q < i + 1 \rightarrow a'(q) = q)$$

# Instantiation Based Proof

$$(0 \leq q < i \rightarrow a(q) = q) \wedge i < n \wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q))$$

$$0 \leq q < i + 1$$

$$q \neq i$$

$$q = i$$

$$a'(q) = q$$

$$(0 \leq q < i + 1 \rightarrow a'(q) = q)$$

# Instantiation Based Proof

$$(0 \leq q < i \rightarrow a(q) = q) \wedge i < n \wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q))$$

$$0 \leq q < i + 1$$

$$q \neq i$$

$$q = i$$

$$a(q) = q$$

$$a'(q) = q$$

$$(0 \leq q < i + 1 \rightarrow a'(q) = q)$$

# Instantiation Based Proof

$$(0 \leq q < i \rightarrow a(q) = q) \wedge i < n \wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q))$$

$$0 \leq q < i + 1$$

$$q \neq i$$

$$q = i$$

$$a(q) = q$$

$$a'(q) = a(q)$$

$$a'(q) = q$$

$$(0 \leq q < i + 1 \rightarrow a'(q) = q)$$

# Instantiation Based Proof

$$(0 \leq q < i \rightarrow a(q) = q) \wedge i < n \wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q))$$

$$0 \leq q < i + 1$$

$$q \neq i$$

$$q = i$$

$$a(q) = q$$

$$a'(q) = a(q)$$

$$a'(q) = q$$

$$a'(q) = q$$

$$(0 \leq q < i + 1 \rightarrow a'(q) = q)$$

# Instantiation Based Proof

$$(0 \leq q < i \rightarrow a(q) = q) \wedge i < n \wedge a'(i) = i \wedge (q \neq i \rightarrow a'(q) = a(q))$$

$$0 \leq q < i + 1$$

$$q \neq i$$

$$q = i$$

$$a(q) = q$$

$$a'(q) = q$$

$$a'(q) = a(q)$$

$$a'(q) = q$$

$$a'(q) = q$$

$$(0 \leq q < i + 1 \rightarrow a'(q) = q)$$



## Universally Quantified Invariant for Termination

```
for(i = 0; i < n; i++) {  
    a[i] = 1;  
}  
while (x > 0) {  
    for(i = 0; i < n; i++) {  
        x = x-a[i];  
    }  
}
```

$$inv_1(i, n, x, p, v) = (0 \leq p < i \rightarrow v \geq 1)$$

$$inv_2(i, n, x, p, v) = (0 \leq x \wedge 0 \leq i < n \wedge (0 \leq p < n \rightarrow v \geq 1))$$
$$= inv_3(i, n, x, p, v)$$

## Further Pointers

- ▶ Solving recursion-free clauses over LI+UIF, [APLAS'11]
- ▶ Solving quantifier free clauses and well-foundedness, [PLDI'12]
- ▶ Solving existentially quantified clauses: [CAV'13]
- ▶ Solving universally quantified clauses: [SAS'13]
- ▶ Proof rules for multi-threaded programs [POPL'11, CAV'11, TACAS'12]
- ▶ Proof rules for functional programs [CAV'11, SAS'12]
- ▶ Software verification competition [SV-COMP'12, SV-COMP'13]