Solving (Quantified) Horn Constraints for Program Verification and Synthesis

Andrey Rybalchenko (Microsoft Research)

September 30, 2015

Programs vs/as Equations

- Execution of rule-based programs
- Solving of equations in form of implication constraints

Quiz

$$F1 := \qquad \forall x : (\exists y : p(x, y)) \to q(x)$$

vs.

$$F2 := \qquad \forall x \, \forall y : p(x, y) \to q(x)$$

Transition System

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

Safety and Termination (WF) of Transition System

$\exists inv \ \exists round :$

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

Safety and Termination (WF) of Transition System

$\exists inv\ \exists round :$

$$
\begin{array}{ll}
init(v) \rightarrow inv(v) & \\
inv(v) \wedge step(v, v') \rightarrow inv(v') & \\
inv(v) \rightarrow safe(v) & \text{safety} \\
inv(v) \wedge step(v, v') \rightarrow round(v, v') & \\
wf(round) & \text{well-foundedness}
\end{array}
$$

From WF to DWF

$wf(rel)$

iff

$\exists ti$:

$rel(v, v') \rightarrow ti(v, v')$

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

$dwf(ti)$

disjunctive

well-foundedness

From WF to DWF

$wf(rel)$

iff

$\exists ti$:

$rel(v, v') \rightarrow ti(v, v')$

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

$dwf(ti)$ disjunctive well-foundedness

dwf - finite union of well-founded relations

Backward Safety of Transition System

$$\exists inv :$$

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

Backward Safety of Transition System

$$\exists inv :$$

$$\neg safe(v) \rightarrow inv(v)$$
$$inv(v') \wedge step(v, v') \rightarrow inv(v)$$
$$inv(v) \wedge init(v) \rightarrow false$$

Forward and Backward Safety of Transition System

$$\exists \mathit{finv}\ \exists \mathit{binv} :$$

$$
\begin{aligned}
&\mathit{init}(v) \to \mathit{finv}(v) \\
&\mathit{finv}(v) \land \mathit{step}(v, v') \to \mathit{finv}(v') \\
&\neg \mathit{safe}(v) \to \mathit{binv}(v) \\
&\mathit{binv}(v') \land \mathit{step}(v, v') \to \mathit{binv}(v)
\end{aligned}
$$

Forward and Backward Safety of Transition System

$$\exists \mathit{finv}\ \exists \mathit{binv}:$$

$$\begin{aligned}
&\mathit{init}(v) \rightarrow \mathit{finv}(v)\\
&\mathit{finv}(v) \wedge \mathit{step}(v, v') \rightarrow \mathit{finv}(v')\\
&\neg \mathit{safe}(v) \rightarrow \mathit{binv}(v)\\
&\mathit{binv}(v') \wedge \mathit{step}(v, v') \rightarrow \mathit{binv}(v)\\
&\mathit{finv}(v) \wedge \mathit{binv}(v) \rightarrow \mathit{false}
\end{aligned}$$

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 of Program with Procedures

$\exists sum :$

$$
\begin{aligned}
& init(v_0) \rightarrow sum(v_0, v_0) \\
& sum(v_0, v_1) \wedge step(v_1, v_2) \rightarrow sum(v_0, v_2) \\
& sum(v_0, v_1) \wedge call(v_1, v_2) \rightarrow sum(v_2, v_2) \\
& sum(v_0, v_1) \wedge call(v_1, v_2) \wedge sum(v_2, v_3) \wedge ret(v_3, v_4) \rightarrow sum(v_0, v_4)
\end{aligned}
$$

Safety of Program with Procedures

$\exists sum :$

$$init(v_0) \to sum(v_0, v_0)$$
$$sum(v_0, v_1) \land step(v_1, v_2) \to sum(v_0, v_2)$$
$$sum(v_0, v_1) \land call(v_1, v_2) \to sum(v_2, v_2)$$
$$sum(v_0, v_1) \land call(v_1, v_2) \land sum(v_2, v_3) \land ret(v_3, v_4) \to sum(v_0, v_4)$$
$$sum(v_0, v_1) \to safe(v_1)$$

Termination of Program with Procedures

$$\exists round\ \exists descent :$$

$$\ldots$$
$$sum(v_0, v_1) \wedge step(v_1, v_2) \rightarrow round(v_1, v_2)$$
$$sum(v_0, v_1) \wedge call(v_1, v_2) \wedge sum(v_2, v_3) \wedge ret(v_3, v_4) \rightarrow round(v_1, v_4)$$

Termination of Program with Procedures

$\exists round\ \exists descent :$

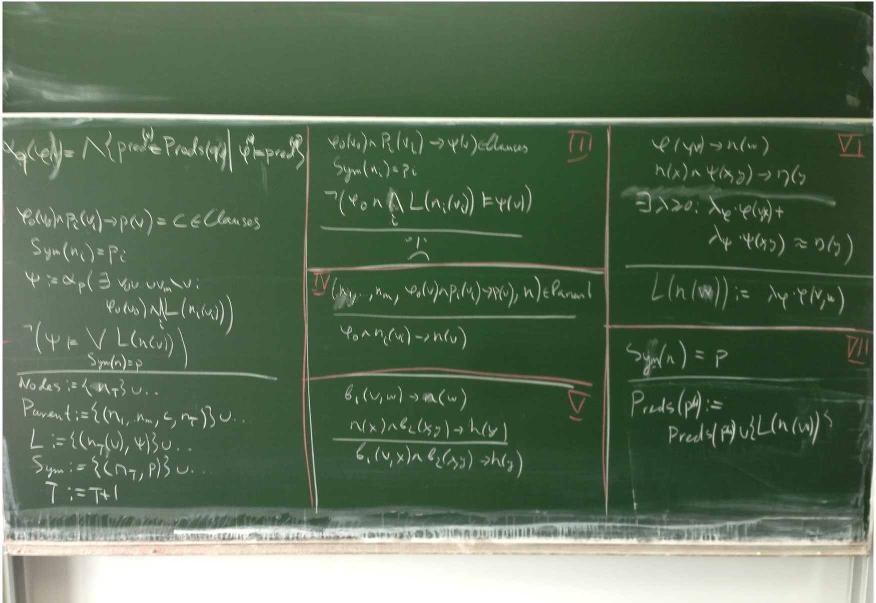
$$\begin{aligned}
&\ldots \\
&sum(v_0, v_1) \land step(v_1, v_2) \rightarrow round(v_1, v_2) \\
&sum(v_0, v_1) \land call(v_1, v_2) \land sum(v_2, v_3) \land ret(v_3, v_4) \rightarrow round(v_1, v_4) \\
&sum(v_0, v_1) \land call(v_1, v_2) \rightarrow descent(v_0, v_2)
\end{aligned}$$

Termination of Program with Procedures

$\exists round\ \exists descent:$

$$\begin{aligned}
&\ldots\\
&sum(v_0, v_1) \land step(v_1, v_2) \to round(v_1, v_2)\\
&sum(v_0, v_1) \land call(v_1, v_2) \land sum(v_2, v_3) \land ret(v_3, v_4) \to round(v_1, v_4)\\
&sum(v_0, v_1) \land call(v_1, v_2) \to descent(v_0, v_2)\\
&wf(round)\\
&wf(descent)
\end{aligned}$$

Solving Horn Constraints



Symbolic self-composition (for non-interference)

$$\exists sum :$$

$$\ldots$$

$$v_0 \neq w_0 \wedge sum(v_0, v_1) \wedge sum(w_0, w_1) \rightarrow v_1 = w_1$$

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

Rely/Guarantee Rule for Safety

$$\exists inv_1 \ \exists inv_2 \ \exists env_1 \ \exists env_2 :$$

$$init(v) \rightarrow inv_1(v)$$
$$inv_1(v) \wedge step_1(v, v') \rightarrow inv_1(v') \wedge env_2(v, v')$$
$$inv_1(v) \wedge env_1(v, v') \rightarrow inv_1(v')$$
$$\dots$$
$$inv_1(v) \wedge inv_2(v) \rightarrow safe(v)$$

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

Resolving Rely/Guarantee Rule

$\exists env_2 :$

$$
\begin{aligned}
&\ldots \\
&inv_1(v) \land step_1(v, v') \to env_2(v, v') \\
&\ldots \\
&inv_2(v) \land env_2(v, v') \to inv_2(v') \\
&\ldots
\end{aligned}
$$

Into Owicki/Gries Rule

$$\ldots$$

$$env_2(v, v') := inv_1(v) \wedge step_1(v, v')$$

$$\ldots$$

$$inv_2(v) \wedge inv_1(v) \wedge step_1(v, v') \rightarrow inv_2(v')$$

$$\ldots$$

Owicki/Gries Rule for Safety

$$\exists inv_1 \ \exists inv_2 :$$

$$init(v) \rightarrow inv_1(v)$$
$$inv_1(v) \wedge step_1(v, v') \rightarrow inv_1(v')$$
$$inv_1(v) \wedge inv_2(v) \wedge step_2(v, v') \rightarrow inv_1(v')$$
$$\ldots$$
$$inv_1(v) \wedge inv_2(v) \rightarrow safe(v)$$

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

Thread-Modular Rule for Safety

$$\exists \mathit{inv_1}\ \exists \mathit{inv_2}\ \exists \mathit{env}:$$

$$\begin{aligned}
&\mathit{init}(v) \rightarrow \mathit{inv_1}(g, l_1)\\
&\mathit{inv_1}(g, l_1) \wedge \mathit{step_1}(v, v') \rightarrow \mathit{inv_1}(g', l_1') \wedge \mathit{env}(g, g')\\
&\ldots\\
&\mathit{inv_1}(g, l_1) \wedge \mathit{inv_2}(g, l_2) \rightarrow \mathit{safe}(v)
\end{aligned}$$

Clauses for preservation of $\mathit{inv_2}(v)$ are symmetric

Quantifier Free Horn Clauses

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

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

Quantified Horn Clauses

- Existential temporal properties, e.g., CTL
- Program synthesis and infinite-state game solving
- Inference of transactions for concurrent programs

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

- Quantified invariants/auxiliary assertions

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

Existentially Quantified Horn Clauses

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

$\mathit{body}(v, w)$ and $\mathit{head}(v, x)$ 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 $EF(q(v))$

$$\exists inv \; \exists round :$$

$$\begin{aligned}
& init(v) \rightarrow inv(v) \\
& inv(v) \wedge \neg q(v) \rightarrow \exists v' : step(v, v') \\
& \qquad\qquad\qquad\qquad\qquad \wedge\ inv(v') \\
& \qquad\qquad\qquad\qquad\qquad \wedge\ round(v, v') \\
& wf(round)
\end{aligned}$$

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

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

iff

$\exists mid$:

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

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

$$\exists mid\ \exists inv_1\ \exists inv_2\ \exists round :$$

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

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

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 adjucent bottles
- Jar is refilled for next round

Stepmother wins if some bottle overflows

Formalization of Game Arena

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

$$
\begin{aligned}
init(v) \quad &= (v_1 = \cdots = v_5 = 0) \\
cindy(v, v') &= (v'_1 = v'_2 = 0 \wedge same(v_3, v_4, v_5) \vee \\
&\quad \ldots \\
&\quad \vee\ v'_5 = v'_1 = 0 \wedge same(v_2, v_3, v_4)) \\
step(v, v') \quad &= (v'_1 \geq v_1 \wedge \cdots \wedge v'_5 \geq v_5 \wedge \\
&\quad v'_1 + \cdots + v'_5 - (v_1 + \cdots + v_5) = J) \\
over(v) \quad &= (v_1 > B \vee \cdots \vee v_5 > B)
\end{aligned}
$$

Stepmother's Victory as Constraint Satisfaction

$$\exists win\ \exists round :$$

$$
\begin{aligned}
& init(v) \rightarrow win(v) \\
& win(v) \land \neg over(v) \land cindy(v, v') \rightarrow \exists v'' : step(v', v'') \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ win(v'') \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ round(v, v'') \\
& wf(round)
\end{aligned}
$$

Example: instantiation of universal quantifiers

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

Example: instantiation of universal quantifiers

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

$$\exists inv_{v1} :$$

$$\forall i, n\ \forall a\colon i = 0 \rightarrow inv_{v1}(i, n, a)$$

$$\forall i, n\ \forall a, a'\colon inv_{v1}(i, n, a) \wedge i < n \wedge a' = a\{i := i\} \rightarrow inv_{v1}(i + 1, n, a')$$

$$\forall i, n\ \forall a\colon inv_{v1}(i, n, a) \rightarrow (\forall q : 0 \leq q < n \rightarrow a(q) = q)$$

Example: instantiation of universal quantifiers

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

$$\exists inv_{v2}:$$

$$\forall i, n\, \forall a \colon i = 0 \rightarrow (\forall q : inv_{v2}(i, n, q, a(q)))$$

$$\forall i, n\, \forall a, a' \colon (\forall p : inv_{v2}(i, n, p, a(p))) \wedge i < n \wedge a' = a\{i := i\} \rightarrow$$

$$(\forall q : inv_{v2}(i, n, q, a'(q)))$$

$$\forall i, n\, \forall a \colon (\forall p : inv_{v2}(i, n, p, a(p))) \rightarrow (\forall q : 0 \leq q < n \rightarrow a(q) = q)$$

Example: instantiation of universal quantifiers

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

$$\exists inv_{v2}\ \exists t_1, t_2 :$$

$$i = 0 \rightarrow inv_{v2}(i, n, q, a(q)))$$

$$t_1(i, n, q, p) \wedge inv_{v2}(i, n, p, a(p)) \wedge i < n \wedge a' = a\{i := i\} \rightarrow$$

$$inv_{v2}(i, n, q, a'(q))$$

$$t_2(i, n, q, p) \wedge inv_{v2}(i, n, p, a(p)) \wedge 0 \leq q < n \rightarrow a(q) = q$$

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];
```

$$\exists \mathit{inv}\ \exists \mathit{round} :$$

$$\ldots$$

$$\mathit{inv}(i, n, x, a) \wedge \mathit{next}(i, n, x, a, i', n', x', a') \rightarrow \mathit{round}(i, n, x, a, i', n', x', a')$$

$$\mathit{wf}(\mathit{round})$$

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]
- Separation logic modulo theories [APLAS'13]
- A constraint-based approach to solving games on infinite graphs [POPL'2014]
- Compositional repair of programs with procedures
- Solving Horn clauses with cardinality constraints
- Probabilistic relational reasoning