Automating Well-Founded Induction for Horn Clause Solving

Sho Torii　Hiroshi Unno

Verification problems of programs in various paradigms (e.g., imperative, functional, and concurrent) can be reduced to constraint solving problems of Horn clauses over predicate variables that represent unknown inductive invariants to be inferred. This paper presents a novel constraint solving method based on inductive theorem proving. The main advantage of the proposed method over existing methods is that it can verify relational specifications (e.g., the equivalence, associativity, commutativity, distributivity, monotonicity, idempotency, and non-interference) where multiple function calls need to be analyzed simultaneously. Our novel combination with Horn-clause-based verification extends the reach of induction-based verification from pure total functions to impure partial programs in various paradigms with different evaluation strategies. The proposed method automates well-founded induction for integers by finding well-founded orders with a ranking function synthesizer and discharging proof obligations with an SMT solver.

Summary

Verification problems of programs in various paradigms including imperative, functional, and concurrent can be reduced to constraint solving problems of Horn clauses over predicate variables that represent unknown inductive invariants to be inferred [1, 2, 9, 13, 14]. An input program is guaranteed to satisfy a given specification if the constraints generated from the program have a solution.

This paper presents a novel Horn constraint solving method based on inductive theorem proving. The main advantage of the proposed method over previous constraint solving methods based on interpolation [1, 3, 4, 6, 10–12, 14, 15] is that it can verify *relational specifications* where multiple function calls need to be analyzed simultaneously. The class of specifications includes practically important ones such as the equivalence, associativity, commutativity, distributivity, monotonicity, idempotency, and non-interference.

For example, consider the following functional program D_{mult} (in OCaml syntax).

```
let rec mult x y =
  if y<=0 then 0 else x + mult x (y-1)
let rec mult_acc x y a =
  if y<=0 then a else mult_acc x (y-1) (a+x)
let main x y =
  assert (mult x y + a = mult_acc x y a)
```

Here, the function `mult` takes two arguments `x`, `y` and recursively computes `x` × `y`. `mult_acc` is a tail-recursive version of `mult` with an accumulator `a`. The function `main` contains an assertion with the condition `mult x y + a = mult_acc x y a`, which represents a relational specification, namely, the equivalence of `mult` and `mult_acc`. By using existing Horn constraint generation methods for call-by-value functional programs [9, 13, 14], the relational verification problem is reduced to the con-

ホーン節制約解消のための整礎帰納法自動化

鳥居 翔　海野 広志, 筑波大学, University of Tsukuba.

straint solving problem of the following Horn clause constraint set $\mathcal{H}_{mult}$.

$$\left\{\begin{array}{l} P(x,y,0) \Leftarrow y \leq 0, \\ P(x,y,x+r) \Leftarrow P(x,y-1,r) \wedge y > 0, \\ Q(x,y,a,a) \Leftarrow y \leq 0, \\ Q(x,y,a,r) \Leftarrow Q(x,y-1,a+x,r) \wedge y > 0, \\ r_1 + a = r_2 \Leftarrow P(x,y,r_1) \wedge Q(x,y,a,r_2) \end{array}\right\}$$

Here, the predicate variable P (resp. Q) represents an inductive invariant among the arguments and the return value of the function `mult` (resp. `mult_acc`). The first Horn clause $P(x,y,0) \Leftarrow y \leq 0 \in \mathcal{H}_{mult}$ is generated from the then-branch of the definition of `mult` and expresses that `mult` returns 0 if 0 is given as the second argument. The second clause $P(x,y,x+r) \Leftarrow P(x,y-1,r) \wedge y > 0 \in \mathcal{H}_{mult}$ is generated from the else-branch and represents that `mult` returns $x+r$ if the second argument y is non-zero and r is returned by the recursive call `mult x (y-1)`. The other Horn clauses are similarly generated from the then- and else-branches of `mult_acc` and the assertion in `main`. Because $\mathcal{H}_{mult}$ has a satisfying substitution (i.e., solution) $\theta_{mult} = \{P \mapsto \lambda(x,y,r).x \times y = r, Q \mapsto \lambda(x,y,a,r).x \times y + a = r\}$ for the predicate variables P and Q, it is guaranteed that the evaluation of `main` x y never causes an assertion failure for any input x and y.

The previous Horn constraint solving methods [1, 3, 4, 6, 10–12, 14, 15], however, cannot solve this kind of constraints that require a relational analysis of multiple predicates. To see why, recall the constraint $r_1 + a = r_2 \Leftarrow P(x,y,r_1) \wedge Q(x,y,a,r_2) \in \mathcal{H}_{mult}$ that asserts the equivalence of `mult` and `mult_acc`, where a relational analysis of the two predicates P and Q is required. The previous methods, however, analyze each predicate P and Q separately, and therefore must infer nonlinear invariants $r_1 = x \times y$ and $r_2 = x \times y + a$ respectively for the predicate applications $P(x,y,r_1)$ and $Q(x,y,a,r_2)$ to conclude $r_1 + a = r_2$ by canceling $x \times y$, because x and y are the only shared arguments between $P(x,y,r_1)$ and $Q(x,y,a,r_2)$. Furthermore, the previous methods can only find solutions that are expressible by some efficiently decidable theories such as the quantifier-free linear real (QF_LRA) and integer (QF_LIA) arithmetic (see `http://smt-lib.org/` for the definition of the theories), which are not powerful enough to express the above nonlinear invariants and the solution θ_{mult} of $\mathcal{H}_{mult}$.

By contrast, our new Horn constraint solving method based on inductive theorem proving can directly infer the mutual invariant $r_1 + a = r_2$ to show that $\mathcal{H}_{mult}$ is solvable by simultaneously unfolding and analyzing the predicate applications $P(x,y,r_1)$ and $Q(x,y,a,r_2)$. More precisely, our method applies well-founded induction to prove the goal clause $\forall x,y,r_1,a,r_2.\ P(x,y,r_1) \wedge Q(x,y,a,r_2) \Rightarrow r_1 + a = r_2$ in $\mathcal{H}_{mult}$ where P, Q are interpreted as the predicates inductively defined as the least models of the definite clauses in $\mathcal{H}_{mult}$. If we are given a well-founded relation $\prec_{mult} = \{((x',y',r_1',a',r_2'),(x,y,r_1,a,r_2)) \mid 0 \leq y' < y\}$ for the universally-quantified integer variables, the proof proceeds as follows.

$P(x,y,r_1) \wedge Q(x,y,a,r_2) \Rightarrow r_1 + a = r_2$

Case1. $P(x,y,0) \Leftarrow y \leq 0$

$\iff r_1 = 0 \wedge y \leq 0 \wedge Q(x,y,a,r_2) \Rightarrow r_1 + a = r_2$

Case1-1. $Q(x,y,a,a) \Leftarrow y \leq 0$

$\iff r_1 = 0 \wedge y \leq 0 \wedge r_2 = a \Rightarrow r_1 + a = r_2$

$\iff \top$

Case1-2. $Q(x,y,a,r) \Leftarrow Q(x,y-1,a+x,r) \wedge y > 0$

$\iff r_1 = 0 \wedge y \leq 0 \wedge Q(x,y-1,a+x,r_2) \wedge y > 0$
$\quad \Rightarrow r_1 + a = r_2$

$\iff \top$

Case2. $P(x,y,x+r) \Leftarrow P(x,y-1,r) \wedge y > 0$

$\iff P(x,y-1,r_1 - x) \wedge y > 0 \wedge Q(x,y,a,r_2)$
$\quad \Rightarrow r_1 + a = r_2$

Case2-1. $Q(x,y,a,a) \Leftarrow y \leq 0$

$\iff P(x, y-1, r_1 - x) \wedge y > 0 \wedge r_2 = a \wedge y \leq 0$
$\quad \Rightarrow r_1 + a = r_2$
$\iff \top$

Case2-2. $Q(x, y, a, r) \Leftarrow Q(x, y-1, a+x, r) \wedge y > 0$

$\iff P(x, y-1, r_1 - x) \wedge y > 0 \wedge Q(x, y-1, a+x, r_2)$
$\quad \Rightarrow r_1 + a = r_2$
(*induction hypothesis*)
$\left[\begin{array}{l} \forall (x', y', r_1', a', r_2') \prec_{mult} (x, y, r_1, a, r_2). \\ \quad P(x', y', r_1') \wedge Q(x', y', a', r_2') \Rightarrow r_1' + a' = r_2' \end{array}\right]$
$\iff y > 0 \wedge r_1 - x + a + x = r_2 \Rightarrow r_1 + a = r_2$
$\iff \top$

The reasoning above is automated by the state-of-the-art SMT provers.

The remaining question is how to get an appropriate well-founded relation for induction. To this end, we propose a technique to synthesize a well-founded relation for induction from the definition of predicates using existing ranking function synthesizers [5, 7, 8], which are originally developed for termination verification of programs. For example, consider the following McCarthy 91 function D_{mc91}.

```
let rec mc91 n =
  if n > 100 then n - 10
  else mc91 (mc91 (n + 11))
let main n =
  if n <= 101 then assert (mc91 n = 91)
```

The assertion expresses the well-known property of the function `mc91`: for any input $n \leq 101$, `mc91` n always returns 91. From the program, we obtain the following Horn clause constraint set $\mathcal{H}_{mc91}$.

$$\left\{\begin{array}{l} R(n, n-10) \Leftarrow n > 100, \\ R(n, r) \Leftarrow R(n+11, s) \wedge R(s, r) \wedge n \leq 100, \\ r = 91 \Leftarrow R(n, r) \wedge n \leq 101 \end{array}\right\}$$

Our method then synthesizes a well-founded relation for induction from the definition of the predicate R as follows. First, we introduce the transition relation T_R that relates the argument (n, r) of R in the head of the definite clause $R(n, r) \Leftarrow R(n+11, s) \wedge R(s, r) \wedge n \leq 100 \in \mathcal{H}_{mc91}$ with those (i.e., $(n+11, s)$ and (s, r)) of R that recursively occurs in the body. Formally, T_R is defined by the following constraint set.

$$\left\{\begin{array}{l} T_R((n, r), (n+11, s)) \\ \quad \Leftarrow R(n+11, s) \wedge R(s, r) \wedge n \leq 100, \\ T_R((n, r), (s, r)) \\ \quad \Leftarrow R(n+11, s) \wedge R(s, r) \wedge n \leq 100 \end{array}\right\}$$

We then apply existing ranking function synthesizers [5, 7, 8] to find a ranking function f that witnesses the well-foundedness of T_R. Formally, we find f such that $T_R \subseteq \{((n', r'), (n, r)) \mid 0 \leq f(n', r') < f(n, r)\}$ and obtain, for example, $f_{mc91}(n, r) = 111 - n$. By using the synthesized well-founded relation $\prec_{mc91} = \{(n', r'), (n, r) \mid 0 \leq 111 - n' < 111 - n\}$, the proof of the goal clause $\forall n, r.\ R(n, r) \wedge n \leq 101 \Rightarrow r = 91$ in $\mathcal{H}_{mc91}$ proceeds as follows.

$R(n, r) \wedge n \leq 101 \Rightarrow r = 91$

Case1. $R(n, n-10) \Leftarrow n > 100$

$\iff r = n - 10 \wedge n \leq 101 \wedge n > 100 \Rightarrow r = 91$
$\iff \top$

Case2. $R(n, r) \Leftarrow R(n+11, s) \wedge R(s, r) \wedge n \leq 100$

$\iff R(n+11, s) \wedge R(s, r) \wedge n \leq 100 \wedge n \leq 101$
$\quad \Rightarrow r = 91$
(*induction hypothesis*)
$\left[\forall (n', r') \prec_{mc91} (n, r).\ R(n', r') \wedge n' \leq 101 \Rightarrow r' = 91\right]$
$\iff ((s, r) \prec_{mc91} (n, r) \Rightarrow s \leq 101 \Rightarrow r = 91)$
$\quad \wedge R(n+11, s) \wedge n \leq 100$
$\quad \Rightarrow r = 91$
$\iff (s > n \Rightarrow s \leq 101 \Rightarrow r = 91)$
$\quad \wedge ((n+11, s) \prec_{mc91} (n, r) \Rightarrow n \leq 90 \Rightarrow s = 91)$
$\quad \wedge R(n+11, s) \wedge n \leq 100$
$\quad \Rightarrow r = 91$

Case2-1. $R(n, n-10) \Leftarrow n > 100$

$\iff (s > n \Rightarrow s \leq 101 \Rightarrow r = 91)$
$\quad \wedge (n \leq 90 \Rightarrow s = 91)$
$\quad \wedge s = n + 1 \wedge n > 89 \wedge n \leq 100$
$\quad \Rightarrow r = 91$
$\iff \top$

Case2-2. $R(n,r) \Leftarrow R(n+11,s) \wedge R(s,r) \wedge n \leq 100$

$$\begin{aligned}
&\Longleftrightarrow (s > n \Rightarrow s \leq 101 \Rightarrow r = 91) \\
&\qquad \wedge (n \leq 90 \Rightarrow s = 91) \\
&\qquad \wedge R(n+22, s') \wedge R(s', s) \wedge n \leq 89 \wedge n \leq 100 \\
&\qquad\quad \Rightarrow r = 91 \\
&\Longleftrightarrow \top
\end{aligned}$$

References

[1] Grebenshchikov, S., Lopes, N. P., Popeea, C., and Rybalchenko, A.: Synthesizing Software Verifiers from Proof Rules, *PLDI '12*, ACM, 2012, pp. 405–416.

[2] Gupta, A., Popeea, C., and Rybalchenko, A.: Predicate Abstraction and Refinement for Verifying Multi-threaded Programs, *POPL '11*, ACM, 2011, pp. 331–344.

[3] Gupta, A., Popeea, C., and Rybalchenko, A.: Solving recursion-free horn clauses over LI+UIF, *APLAS '11*, LNCS, Vol. 7078, Springer, 2011, pp. 188–203.

[4] Hoder, K., Bjørner, N., and Moura, L. D.: μZ: An Efficient Engine for Fixed Points with Constraints, *CAV '11*, LNCS, Vol. 6806, Springer, 2011, pp. 457–462.

[5] Kuwahara, T., Terauchi, T., Unno, H., and Kobayashi, N.: Automatic Termination Verification for Higher-Order Functional Programs, *ESOP '14*, LNCS, Vol. 8410, Springer, 2014, pp. 392–411.

[6] McMillan, K. and Rybalchenko, A.: Computing Relational Fixed Points using Interpolation, Technical Report MSR-TR-2013-6, 2013.

[7] Podelski, A. and Rybalchenko, A.: A Complete Method for the Synthesis of Linear Ranking Functions, *VMCAI '04*, LNCS, Vol. 2937, Springer, 2004, pp. 239–251.

[8] Popeea, C. and Rybalchenko, A.: Compositional Termination Proofs for Multi-threaded Programs, *TACAS '12*, LNCS, Vol. 7214, Springer, 2012, pp. 237–251.

[9] Rondon, P., Kawaguchi, M., and Jhala, R.: Liquid Types, *PLDI '08*, ACM, 2008, pp. 159–169.

[10] Rümmer, P., Hojjat, H., and Kuncak, V.: Disjunctive Interpolants for Horn-Clause Verification, *CAV '13*, LNCS, Vol. 8044, Springer, 2013, pp. 347–363.

[11] Terauchi, T.: Dependent types from counterexamples, *POPL '10*, ACM, 2010, pp. 119–130.

[12] Terauchi, T. and Unno, H.: Relaxed Stratification: A New Approach to Practical Complete Predicate Refinement, *ESOP '15*, LNCS, Vol. 9032, Springer, 2015, pp. 610–633.

[13] Unno, H. and Kobayashi, N.: On-Demand Refinement of Dependent Types, *FLOPS '08*, LNCS, Vol. 4989, Springer, 2008, pp. 81–96.

[14] Unno, H. and Kobayashi, N.: Dependent Type Inference with Interpolants, *PPDP '09*, ACM, 2009, pp. 277–288.

[15] Unno, H. and Terauchi, T.: Inferring Simple Solutions to Recursion-free Horn Clauses via Sampling, *TACAS '15*, LNCS, Vol. 9035, Springer, 2015, pp. 149–163.