A Horn Constraint Solver based on Inductive Theorem Proving

Hiroki Sakamoto Sho Torii Shu Nakao Hiroshi Unno (University of Tsukuba)

Background

Verification problems of programs in various paradigms

imperative, logic, concurrent, functional, ...

can be reduced to Horn constraint solving problems

Verification Problem

```
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)

∀x,y,a.(mult x y)+a = (mult_acc x y a)
```

reduce

Horn Constraint Solving Problem

$\mathcal{H}_{mult}$

```
P(x,0,0).
P(x,y,x+r) :- P(x,y-1,r) ∧ y≠0.

Q(x,0,a,a).
Q(x,y,a,r) :- Q(x,y-1,a+x,r) ∧ y≠0.
```

Unknown invariant of mult

Unknown invariant of mult_acc

```
P(x,y,r1) ∧ Q(x,y,a,r2) |= r1+a=r2
```

Is there a substitution for P and Q that satisfies all the constraints?

Applications

1. Verif. of higher-order programs

```
type list = Nil | Cons of int * list
let rec sum_list l = match l with
  | Nil -> 0 | Cons(x,xs) -> x + sum_list xs
let rec foldl f s l = match l with
  | Nil -> s | Cons(x,xs) -> foldl f (f s x) xs
let plus x y = x + y
∀l.sum_list l = foldl plus 0 l
```

reduce

```
P(Nil,0). P(Cons(x,l'),x+r) :- P(…).
R(…). R(…) :- …
P(l,r1) ∧ R(0,l,r2) |= r1=r2
```

prove

Spec. Satisfied!

2. Verif. of user-defined ADTs

```
type list =
  Nil | Cons of int * list
let rec append l ys = …
let rec drop n l = …
let rec take n l = …
∀n,l.
append (take n l) (drop n l) = l
```

reduce

```
P(Nil,l2,l2). P(…) :- ….
Q(…). Q(…) :- ….
R(…). R(…) :- ….
∀n,l. P(…) ∧ Q(…) ∧ R(…) |= r=l
```

prove

Spec. Satisfied!

3. Verif. of programs that may raise exceptions

```
let rec find p l = match l with
  | []->raise Not_found
  | x::xs->if p x then x else find p xs
let rec find_opt p l = match l with
  | [] -> None
  | x::xs -> if p x then Some x else
find_opt p xs
∀p,l.
try find_opt p l = Some (find p l)
with Not_found -> find_opt p l = None
```

reduce

```
…
Pex(…,Not_found) ∧ Q(…,r2) |= None=r2
```

prove

Spec. Satisfied!

4. Verif. of heap-manipulating programs (with Nam Mai and Tachio Terauchi)

```
let rec zero_list x = match !x with
  | Nil->true
  | Cons(d,l)->if d=0 then zero_list l else false
let rec init_list x = match !x with
  | Nil->() | Cons(d,l)->init_list l;x:=Cons(0,l)
∀x. (init_list x) => (zero_list x)
```

reduce

```
Zero_list(h,x) :- h=(x |-> Nil) * h2.
Zero_list(h,x) :-
  Zero_list(h',l), h=(x|->Cons(0,l))*h'.
Init_list(h,x,h) :- h=(x|->Nil)*h2.
Init_list(h,x,h') :- Init_list(h2,l,h3),
 h=(x|->Cons(d,l))*h2, h'=(x|->Cons(0,l))*h3.
∀h,x,h'. Init_list(h,x,h') |= Zero_list(h',x)
```

prove

Spec. Satisfied!

5. Verif. of nonlinear spec.

```
let rec sum n =
  if n<0 then n + sum (n+1)
  else if n=0 then 0
       else n + sum (n-1)
∀n. n>=0 => 2*sum n = n*(n+1)
```

reduce

```
P(0,0). P(…) :- P(…) ∧ x<0.
p(x,r+x) :- P(x-1,r) ∧ x>0.
∀n.P(x,r)∧x>=0 |= 2*r=x*(x+1)
```

prove

Spec. Satisfied!

Our Horn Constraint Solver (Demo Available!)

Approach : Inductive theorem proving

1. Reduce Horn constaint solving problems into validity checking of first-order formulas with inductively defined predicates, then
2. Check by induction on the derivation of the predicates

Automation of inductive proofs

(1) A new proof system tailored to Horn constraint solving

(2) Discharge proof obligations arising in the proof search using
- PDR-based Horn constraint solver, and
- SMT solver

(3) Exploit lemmas, which are
- User-supplied,
- Heuristically obtained from the given constraints, or
- Automatically generated by an abstract interpreter

Advantages

Our method can

(1) deal with **constraints over any background theories** supported by the underlying SMT solver (nonlinear arithmetics, algebraic data structures, and heaps)

(2) enable verification of **relational specifications** across programs in various paradigms (functional equivalence, associativity, commutativity, monotonicity, …)

(3) enable verification of recursive functions that are possibly **non-deterministic, higher-order, exception-raising, and over non-inductively defined data types**

Example : Inductive Proof for $\mathcal{H}_{mult}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\models y \neq 0 \wedge (r_1 - x) + (a + x) = r_2 \Rightarrow r_1 + a = r_2}\,\text{Valid}}{\phi; C \wedge y \neq 0 \wedge (r_1 - x) + (a + x) = r_2 \vdash r_1 + a = r_2}\,\text{Apply}}{\vdots \quad \phi; C \wedge y \neq 0 \vdash r_1 + a = r_2}\,\text{Unfold}}{\vdots \quad \phi; B \wedge y \neq 0 \vdash r_1 + a = r_2}\,\text{Unfold}}{\dfrac{\phi; A \vdash r_1 + a = r_2}{A \vdash r_1 + a = r_2}\,\text{Induct}}$$

$\phi = \forall x', y', r_1', r_2'.\big(P(x', y', r_1') \prec P(x, y, r_1)\big) \wedge P(x', y', r_1') \wedge Q(x', y', a', r_2') \Rightarrow r_1' + a' = r_2'$

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

$B = A \wedge P(x, y - 1, r_1 - x)$

$C = B \wedge Q(x, y - 1, a + x, r_2)$

Extensions

1. Coinduction on copredicates (with Seiich Shinomiya)

```
merge(Cons(a,as),bs,Cons(a,cs)) :- merge(bs,as,cs).
even(Cons(x1,Cons(x2,xs)),Cons(x1,ys))
  :- even(xs,ys).
equal(Cons(l,ls),Cons(r,rs)) :- l=r,equal(ls,rs).
∀u,v,r1,r2.merge(u,v,r1),even(r1,r2) |= equal(r2,u)
```

prove

Solvable!

2. Relational program synthesis

```
let rec mult x y = …
let rec mult_acc x y a = ?
∀x,y,a.(mult x y) + a
          = (mut_acc x y a)
```

reduce

Horn Constraint Abduction Problem

solve

Definition of mult_acc