Proving Horn Clause Specifications of Imperative Programs

Emanuele De Angelis (University “d’Annunzio”, Pescara, Italy)
Fabio Fioravanti (University “d’Annunzio”, Pescara, Italy),
Alberto Pettorossi (University “Tor Vergata”, Rome, Italy),
Maurizio Proietti (IASI-CNR, Rome, Italy)

Specifying Program Correctness

- Imperative, sequential programs (integer variables, assignments, conditionals, while-loops, jumps)

- Partial correctness specification[Hoare triple]:

$$\{\varphi\}\ \mathit{prog}\ \{\psi\}$$

  If the initial values of the program variables satisfy the **precondition** φ **and** *prog* terminates **then** the final values of the program variables satisfy the **postcondition** ψ.

- The assertions φ, ψ are any first order formulas

Proving Partial Correctness

- Hoare proof rules for proving partial correctness:

$$\frac{\{\mathrm{Inv} \wedge b\}\ \mathit{comm}\ \{\mathrm{Inv}\}}{\{\mathrm{Inv}\}\ \mathit{while}\ b\ \mathit{do}\ \mathit{comm}\ \{\mathrm{Inv} \wedge \neg b\}} \qquad (\textit{while-do}\ \text{rule})$$

- Two difficulties:
  1. We have to find suitable auxiliary assertions (invariant Inv)
  2. First order logic is undecidable

Proving Partial Correctness with Horn Clauses and Linear Arithmetic Assertions

- Verification difficulties can be mitigated by restricting to linear arithmetic assertions: logical validity is decidable
- Partial correctness can be translated into Horn clauses with linear arithmetic constraints (aka Constraint Logic Programs)

Translating Partial Correctness to CLP

- The specification:
  $\{n \geq 1\}$ x=0; y=0; while (x<n) {x=x+1; y=y+2} {y>x}

  is translated into the Horn clauses (Verification Conditions, in CLP syntax):

  p(X, Y, N) :- N≥1, X=0, Y=0. %Initialization
  p(X+1, Y+2, N) :- X<N, p(X, Y, N). %Loop
  false :- Y≤X, X≥ N, p(X, Y, N). %Loop exit

- The program is correct if the Horn clauses have a solution (linear arithmetic predicate that satisfies all clauses):
  p(X, Y, N) ≡ (X=0, Y=0, N≥1) ∨ Y>X %Loop invariant

Horn Clause Solvers

- Effective solvers for computing (linear arithmetic) solutions of constrained Horn clauses/CLP are available:

  QARMC [Rybalchenko et al.],
  Z3 [deMoura-Bjorner],
  VeriMAP [DeAngelis et al.],
  CHA [Gallagher et al.],
  MSATIC3 [Griggio et al.],
  SeaHorn [Gurfinkel,Navas],
  TRACER [Jaffar et al.]

  (and others)

- Some extensions with more complex constraints are being developed (nonlinear arithmetic, arrays, …)

Limitations of Linear Arithmetic Assertions

- Not very expressive; relative completeness is lost.
- Example: computing Fibonacci numbers

fibonacci:
```
while (n>0) {
           t=u;
           u=u+v;
           v=t;
           n=n-1 }
```

{n=N, N>=0, u=1, v=0, t=0} *fibonacci* {fib(N,u)}

- The postcondition of *fibonacci* cannot be specified by using linear arithmetic assertions.

Horn Clause Specifications

- Functional Horn specifications:

  {z1= P1, . . . , zs= Ps, **pre**(P1, . . . , Ps)} *prog* {**f**(P1, . . . , Ps,z)}

  where :
  - z1, . . . , zs are global variables of *prog*
  - P1, . . . , Ps are parameters
  - z is a variable in {z1, . . . , zs}
  - **pre** and **f** are defined by a set of Horn clauses with linear arithmetic constraints
  - **f** is a functional relation : z=F(P1, . . . , Ps) for some function F defined for all P1, . . . , Ps that satisfy **pre**

- All recursive functions on the integers can be specified by sets of Horn clauses with linear arithmetic constraints

Fibonacci Specification

- Fibonacci specification:

  {n=N, N>=0, u=1, v=0, t=0} *fibonacci* {fib(N,u)}

  where:

```
fib(0,1).
fib(1,1).
fib(N3,F3) :- N1>=0, N2=N1+1, N3=N2+1, F3=F1+F2,
              fib(N1,F1), fib(N2,F2).
```

Translating Partial Correctness into Horn Clauses

- A partial correctness specification can be translated into CLP in two steps:

  Step1. Translating the operational semantics into CLP

  Step 2. Generating verification conditions as a set of CLP clauses

Translating the Operational Semantics

- Define a relation **r_prog**(P1, . . . , Ps,Z) such that for all integers P1, . . . , Ps,

  if the initial values of the program variables satisfy the precondition **pre**(P1, . . . , Ps)

  then the final value of z computed by **prog** is Z

- Fibonacci Example: Define a relation **r_fibonacci**(N,U) such that for all integers N, if the program variables satisfy the precondition

  {n=N, N>=0, u=1, v=0, t=0}

  then the final value of u computed by program **fibonacci** is U

Translating the Operational Semantics

- Define a set *OpSem* of clauses that encode the operational semantics*:*

```
r_fibonacci(N,U) :- initConf(Cf0,N), reach(Cf0,Cfh), finalConf(Cfh,U).
initConf(cf(LC,E),N) :- N>=0, U=1, V=0, T=0,
                        firstComm(LC),
                        env((n,N),E), env((u,U),E), env((v,V),E), env((t,T),E).
finalConf(cf(LC,E),U) :- haltComm(LC), env((u,U),E).
reach(Cf,Cf).
reach(Cf0,Cf2) :- tr(Cf0,Cf1), reach(Cf1,Cf2).
```

tr(Cf0,Cf1) is the transition relation that defines the operational semantics

Generating Verification Conditions

- For each clause **f**(P1, . . . ,Ps,Z) :- B defining the postcondition,

  (1) Replace **f** by **r_prog** (we want to prove that **r_prog** satisfies **f**):

  r_prog(P1, . . . ,Ps,Z) :- B′

  (2) Negate the clause (exploiting functionality of **r_prog**):

  $Y \neq Z$, **r_prog**((P1, . . . ,Ps,Y), B′ where Y is a new variable

  (3) Case split:

  p1 :- Y >Z, **r_prog**(X1, . . . ,Xs,Z), B′

  p2 :- Y <Z, **r_prog**(X1, . . . ,Xs,Z), B′

  where p1 and p2 are two new predicate symbols

- Theorem (Partial Correctness). *If* for all generated clauses p :- G, $p \notin M(\mathit{OpSem} \cup \{p \text{ :- } G\})$, *then* the specification is valid.

Verification Conditions for Fibonacci

- Generating verification conditions for Fibonacci

```
p1 :- F>1, r_fibonacci(0,F).
p2 :- F<1, r_fibonacci(0,F).
p3 :- F>1, r_fibonacci(1,F).
p4 :- F<1, r_fibonacci(1,F).
p5 :- N1>=0, N2=N1+1, N3=N2+1, F3>F1+F2,
      r_fibonacci(N1,F1), r_fibonacci(N2,F2), r_fibonacci(N3,F3).
p6 :- N1>=0, N2=N1+1, N3=N2+1, F3<F1+F2,
      r_fibonacci(N1,F1), r_fibonacci(N2,F2), r_fibonacci(N3,F3).
```

A limitation of Horn clause solvers

- Horn clause solvers with linear arithmetic could not prove the satisfiability of the verification conditions for Fibonacci (we checked with QARMC, Z3, MSATIC3)
- Claim (not proved in the paper): The verification conditions for Fibonacci have no atomic solutions (e.g., $p(X) \equiv c$)

Verification by Transformation

1. Removal of the Interpreter (IR).
   Specialize *OpSem* w.r.t. *prog*

2. Unfold/Fold transformation.
   Apply the unfold/fold rules and transform program
   OpSem ∪ {*p* :- *G*} into program *T* such that:

   (i) $p \in M(OpSem \cup \{p \text{ :- } G\})$ iff $p \in M(T)$

   (ii) *either* $p \in T$ (p is true in M(*T*), specification not valid)

   or no clauses for p in *T* (p is false in M(*T*), specification valid)

- Step 2 (ii) may fail! (due to incompleteness)

Verifying Fibonacci (IR)

1. Removal of the Interpreter (IR).
   Specialize *OpSem* w.r.t. *fibonacci*

```
r_fibonacci(N,F) :-                              %Initialization
      N>=0, U=1, V=0, T=0,
      r(N,U,V,T,  N1,F,V1,T1).
r(N,U,V,T, N2,U2,V2,T2) :-                       %Loop
      N>=1, N1=N-1, U1=U+V, V1=U, T1=U,
      r(N1,U1,V1,T1, N2,U2,V2,T2).
r(N,U,V,T,  N,U,V,T) :- N=<0.                    %Loop exit
```

Verifying Fibonacci (U/F)

2. Unfold/Fold transformation.

```
p5:- N1>=0, N2=N1+1, N3=N2+1, F3>F1+F2,
      r_fibonacci(N1,F1), r_fibonacci(N2,F2), r_fibonacci(N3,F3).
```

UNFOLD (Replace **r_fibonacci**(N,F) by the body of its definition)

```
p5 :- N1>=0, U=1, V=0, T=0, N2=N1+1, N3=N2+1, F3>F1+F2,
       r(N1,U,V,T, Na,F1,Va,Ta), r(N2,U,V,T, Nb,F2,Vb,Tb), r(N3,U,V,T, Nc,F3,Vc,Tc).
```

DEFINITION (Conjunctive definition + Generalization):

```
gen(N1,U,V,T) :- N1>=0, U>=1, V>=0, T>=0, N2=N1+1, N3=N2+1, F3>F1+F2,
              r(N1,U,V,T, Na,F1,Va,Ta), r(N2,U,V,T, Nb,F2,Vb,Tb), r(N3,U,V,T, Nc,F3,Vc,Tc).
```

FOLD (Replace an instance of the body of the definition of **gen** by its head)

```
p5 :- N1>=0, U>=1, V=0, T=0, gen(N1,U,V,T).
```

Verifying Fibonacci (U/F) Cont.

UNFOLD the definition of **gen**

```
gen(N1,U,V,T) :-
    N1=0, U>=1, V>=0, T>=0, F3>U+F2,
    r(1,U,V,T, Nb,F2,Vb,Tb), r(2,U,V,T, Nc,F3,Vc,Tc).

gen(N1,U,V,T) :-
    N1>=1, U>=1, V>=0, T>=0, U1=U+V, N=N1-1, N2=N1+1, F3>F1+F2,
    r(N,U1,U,U,Na,F1,Va,Ta), r(N1,U1,U,U,Nb,F2,Vb,Tb), r(N2,U1,U,U,Nc,F3,Vc,Tc).
```

Verifying Fibonacci (U/F) Cont.

UNFOLD the definition of gen

~~gen(N1,U,V,T) :-~~
~~N1=0, U>=1, V>=0, T>=0, F3>U+F2,~~
~~r(1,U,V,T, Nb,F2,Vb,Tb), r(2,U,V,T, Nc,F3,Vc,Tc).~~

DELETE: body fails after some unfoldings

gen(N1,U,V,T) :-
N1>=1, U>=1, V>=0, T>=0, U1=U+V, N=N1-1, N2=N1+1, F3>F1+F2,
r(N,U1,U,U,Na,F1,Va,Ta), r(N1,U1,U,U,Nb,F2,Vb,Tb), r(N2,U1,U,U,Nc,F3,Vc,Tc).

Verifying Fibonacci (U/F) Cont.

UNFOLD the definition of **gen**

~~gen(N1,U,V,T) :-~~
~~N1=0, U>=1, V>=0, T>=0, F3>U+F2,~~
~~r(1,U,V,T, Nb,F2,Vb,Tb), r(2,U,V,T, Nc,F3,Vc,Tc).~~

DELETE: body fails after some unfoldings

gen(N1,U,V,T) :-
N1>=1, U>=1, V>=0, T>=0, U1=U+V, N=N1-1, N2=N1+1, F3>F1+F2,
r(N,U1,U,U,Na,F1,Va,Ta), r(N1,U1,U,U,Nb,F2,Vb,Tb), r(N2,U1,U,U,Nc,F3,Vc,Tc).

FOLD (Replace an instance of the body of the definition of **gen** by its head)

gen(N1,U,V,T) :-
N1>=1, U>=1, V>=0, T>=0, U1=U+V, N=N1-1,
gen(N,U1,U,U).

Verifying Fibonacci: Final Program

p5 :- N1>=0, U>=1, V=0, T=0, **gen**(N1,U,V,T).

gen(N1,U,V,T) :- N1>=1, U>=1, V>=0, T>=0, N=N1-1, U1=U+V, **gen**(N,U1,U,U).

NO FACTS for gen and p5.

$\Rightarrow$ The least model of the program is empty

$\Rightarrow$ All clauses can be deleted

$\Rightarrow$ **p5** is false in M(*T*)

Similar proofs for p1, p2, p3, p4, p6.

The specification is *valid*.

Automating the Unfold/Fold Transformation

- Transformation strategy in two steps:

  (2.1) Linearization: Transforms clauses to linear recursive form by using conjunctive definitions

  (2.2) Iterated Specialization: Transforms the definition of the predicate p into a set T of clauses such that either

  (i) $p \in M(\mathit{OpSem} \cup \{p \text{ :- } G\})$ iff $p \in M(T)$

  (ii) either $p \in T$ (p is true in *T*, specification *not valid*)

  or no clauses for p in *T* (p is false in *T*, specification *valid*)

The Linearization Strategy

$T = \emptyset$;   Defs = {p :- G};

while $\exists$ cl $\in$ Defs **do**

  Cls = Unfold(cl);

  Defs = (Defs − {cl}) $\cup$ Define(Cls);

  $T = T \cup$ Fold(Cls, Defs);

od

cl:	H :- c, p1(X), p2(Y).	%nonlinear
Define:	lin(X,Y) :- p1(X), p2(Y).	%conjunctive def
Fold:	H :- c, lin(X,Y).	%linear

Linearized Fibonacci

```
p5 :- A=B+2, C=B+1, D=1, E=0, F=0, G=1, H=0, I=0, J=1, K=0, L=0, B>=0, M>N+N1,
        lin1(B,G,H,I,P,N1,Q,R,C,D,E,F,S,N,T,U,A,J,K,L,V,M,W,X).
lin1(A,B,C,D,A,B,C,D,E,F,G,H,E,F,G,H,I,J,K,L,M,N,N1,P) :- I=Q+1, K=R-J, A=<0, E=<0, Q>=0, lin2(Q,R,J,J,M,N,N1,P).
lin1(A,B,C,D,A,B,C,D,E,F,G,H,I,J,K,L,M,N,N1,P,M,N,N1,P) :- Q=E-1, R=F+G, M=<0, A=<0, E>=1, lin2(Q,R,F,F,I,J,K,L).
lin1(A,B,C,D,A,B,C,D,E,F,G,H,I,J,K,L,M,N,N1,P,Q,R,S,T) :- M=U+1, N1=V-N, W=E-1, X=F+G, A=<0, E>=1, U>=0,
        lin3(W,X,F,F,I,J,K,L,U,V,N,N,Q,R,S,T).
lin1(A,B,C,D,E,F,G,H,I,J,K,L,I,J,K,L,M,N,N1,P,M,N,N1,P) :- Q=A-1, R=B+C, M=<0, I=<0, A>=1, lin2(Q,R,B,B,E,F,G,H).
lin1(A,B,C,D,E,F,G,H,I,J,K,L,I,J,K,L,M,N,N1,P,Q,R,S,T) :- M=U+1, N1=V-N, W=A-1, X=B+C, I=<0, A>=1, U>=0,
        lin3(W,X,B,B,E,F,G,H,U,V,N,N,Q,R,S,T).
lin1(A,B,C,D,E,F,G,H,I,J,K,L,M,N,N1,P,Q,R,S,T,Q,R,S,T) :- U=I-1, V=J+K, W=A-1, X=B+C, Q=<0, A>=1, I>=1,
        lin3(W,X,B,B,E,F,G,H,U,V,J,J,M,N,N1,P).
lin1(A,B,C,D,E,F,G,H,I,J,K,L,M,N,N1,P,Q,R,S,T,U,V,W,X) :- Q=Y+1, S=Z-R, A1=I-1, B1=J+K, C1=A-1, D1=B+C, A>=1, I>=1, Y>=0,
        lin1(C1,D1,B,B,E,F,G,H,A1,B1,J,J,M,N,N1,P,Y,Z,R,R,U,V,W,X).
lin1(A,B,C,D,A,B,C,D,E,F,G,H,E,F,G,H,I,J,K,L,I,J,K,L) :- I=<0, A=<0, E=<0.
lin2(A,B,C,D,A,B,C,D) :- A=<0.
lin2(A,B,C,D,E,F,G,H) :- A=I+1, C=J-B, I>=0, lin2(I,J,B,B,E,F,G,H).
lin3(A,B,C,D,A,B,C,D,E,F,G,H,E,F,G,H) :- E=<0, A=<0.
lin3(A,B,C,D,A,B,C,D,E,F,G,H,I,J,K,L) :- E=M+1, G=N-F, A=<0, M>=0, lin2(M,N,F,F,I,J,K,L).
lin3(A,B,C,D,E,F,G,H,I,J,K,L,I,J,K,L) :- M=A-1, N=B+C, I=<0, A>=1, lin2(M,N,B,B,E,F,G,H).
lin3(A,B,C,D,E,F,G,H,I,J,K,L,M,N,N1,P) :- I=Q+1, K=R-J, S=A-1, T=B+C, A>=1, Q>=0, lin3(S,T,B,B,E,F,G,H,Q,R,J,J,M,N,N1,P).
```

The Iterated Specialization Strategy

$T = \varnothing$;   Defs = {p :- G};

while $\exists$ cl $\in$ Defs **do**

Cls = Unfold(cl);

Cls = ClauseRemoval(Cls);   %Delete clauses with unsat body

Defs = (Defs − {cl}) $\cup$ Define(Cls);

$T = T \cup$ Fold(Cls, Defs);

od;

Delete from T all predicates that "do not depend on facts"

cl:	H :- c, A.	
Define:	New :- g, A.	%atomic def where constraint g is a
Fold:	H :- c, New.	%generalization of c (c $\to$ g)

Specialized Fibonacci

p5 :- A=B+2, C=B+1, D=1, E=0, F=0, G=1, H=0, I=0, J=1, K=0, L=0, B>=0, M>N+Z,
    new1(B,G,H,I,P,Z,Q,R,C,D,E,F,S,N,T,U,A,J,K,L,V,M,W,X).

new1(A,B,C,C,D,E,F,G,H,B,C,C,I,J,K,L,M,B,C,C,N,Z,P,Q) :-
    A=M-2, H=M-1, Z>E+J, B>=1, M>=2, M=R+1, S=B+C, T=H-1, U=B+C, V=A-1,
    W=B+C, A>=1, H>=1, R>=0,
    new1(V,W,B,B,D,E,F,G,T,U,B,B,I,J,K,L,R,S,B,B,N,Z,P,Q).

No facts

- ⇒ the least model is empty
- ⇒ all clauses can be deleted
- ⇒ p5 is false
- ⇒ the verification condition is satisfied

Experiments

Program	*Specified Function*	*Proof Time*			
		RI	*LN*	*IS*	*Total*
fibonacci	`fib(0,1).` `fib(1,1).` `fib(N3,F3) :- N1>=0, N2=N1+1, N3=N2+1, F3=F1+F2, fib(N1,F1), fib(N2,F2).`	50	40	340	430
remainder of integer division	`rem(X,Y,X) :- X<Y.` `rem(X,Y,0) :- X=Y.` `rem(X,Y,Z) :- X>Y, X1=X-Y, rem(X1,Y,Z).`	20	10	20	50
greatest common divisor	`gcd(X,Y,Z) :- X<Y, Y1=Y-X, gcd(X,Y1,Z).` `gcd(X,Y,X) :- X=Y.` `gcd(X,Y,Z) :- X>Y, X1=X-Y, gcd(X1,Y,Z).`	30	20	80	130
McCarthy's 91 function	`mc91(X,Z) :- X =< 100, Z=91.` `mc91(X,Z) :- X>=101, Z=X-10.`	40	–	40	80
McCarthy's 91 function	`mc91r(X,Z) :- X>=101, Z=X-10.` `mc91r(X,Z) :- X =< 100, X1=X+11, mc91r(X1,K), mc91r(K,Z).`	40	40	142611	142691
integer division	`idiv(M,K,0) :- M+1=<K.` `idiv(M,K,Q1) :- M>=K, M1=M-K, Q1=Q+1, idiv(M1,K,Q).`	30	10	120	160
even-odd multiplication	`mult(J,0,0).` `mult(J,N1,Y1) :- N1=N+1, Y1=Y+J, N>0, mult(J,N,Y).`	50	60	880	990

Table 1: Experimental results. The columns named *RI*, *LN*, *IS*, and *Total* show the times in milliseconds taken for the Removal of the Interpreter, the Linearization, the Iterated Specialization, and the total proof time (that is, $RI + LN + IS$), respectively. '–' means that the Linearization step for the program *McCarthy's 91 function* was not needed.

Conclusions

- Horn clause solving < Unfold/Fold transformation
- Key factor: use of conjunctive definitions and folding (thus, a similar effect can be achieved by conjunctive partial deduction or supercompilation)
- Current work: Unfold/Fold transformation (in particular, linearization) as a preprocessor for Horn clause solvers