Synthesizing Software Verifiers from Proof Rules

Corneliu Popeea
Technical University Munich

Joint work with Sergey Grebenshchikov,
Nuno Lopes and
Andrey Rybalchenko

Developing verifiers today

Program Model

transition system, program with procedures,
multi-threaded program, functional program, ...

+

Proof Rule

invariance, summarization, rely/guarantee,
transition invariance, refinement typing, ...

+

Complex verification effort

=

Verification Tool

Developing verifiers tomorrow

Verification Tool = Synthesizer (Program Model, Proof Rule)

Programs as transition systems

```
int sum (int i) {
A:  int s = 0;

B:  while (i > 0) {
        s = s + i;
        i = i – 1;
    }

C: assert (s >= 0);
}
```

V = (pc, s, i)
V' = (pc', s', i')

Init(V) = (pc = A)

Step(V, V') =
  (pc=A ∧ pc'=B ∧ s'=0 ∧ i'=i) ∨
  (pc=B ∧ pc'=B ∧ i>0 ∧ s'=s+i ∧ i'=i-1) ∨
  (pc=B ∧ pc'=C ∧ i≤0 ∧ s'=s ∧ i'=i)

Error(V) = (pc=C ∧ s<0)

Invariance proof rule

- Inv(V) - describes reachable states

Init(V) -> Inv(V)
Inv(V) ∧ Step(V, V') -> Inv(V')
Inv(V) ∧ Error(V) -> false

Transition system is safe

Example

```
int sum (int i) {
A:  int s = 0;

B:  while (i > 0) {
        s = s + i;
        i = i – 1;
    }

C: assert (s >= 0);
}
```

Solution:
Inv(V) = (pc=A ∨ s ≥ 0)

Find Inv(V) such that:

1) pc = A -> Inv(V)

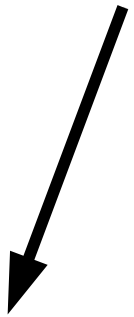
2) Inv(V) ∧
((pc=A ∧ pc'=B ∧ s'=0 ∧ i'=i) ∨
(pc=B ∧ pc'=B ∧ i>0 ∧ s'=s+i ∧ i'=i-1) ∨
(pc=B ∧ pc'=C ∧ i≤0 ∧ s'=s ∧ i'=i))
-> Inv(V')

3) Inv(V) ∧ pc=C ∧ s<0 -> false

Transition invariance proof rule

- Inv(V) - describes reachable states
- TransInv(V,V') – describes reachable computations

Inv(V) ∧ Step(V, V') -> TransInv(V, V')
TransInv(V, V') ∧ Step(V', V'') -> TransInv(V, V'')
dwf(TransInv(V, V'))



Transition system terminates

exists $WF_1(V,V')$, …, $WF_N(V,V')$:
TransInv(V,V') -> $WF_1(V,V')$ ∨ .. ∨ $W_F(V,V')$

Example

```
int sum (int i) {
A:  int s = 0;

B:  while (i > 0) {
        s = s + i;
        i = i – 1;
    }

C: assert (s >= 0);
}
```

Solution:

Inv(V) = (pc=A ∨ s ≥ 0)

TransInv(V, V') = (pc=A ∧ pc'=B) ∨
(pc=A ∧ pc'=C) ∨
(pc=B ∧ pc'=C) ∨
(i' < i ∧ i > 0)

Outline

- Programs, properties, and proof rules
  - Transition systems
  - Reachability, termination
- Proof rules as Horn Clauses + DWF
- Experience with software verifiers

Horn clause representation

- Symbols in a clause
  - queries: $q_1(v_1)$, $q_2(v_2)$, ...
  - formulas in some theory: c(v), d(v)
  - dwf-predicate

- Clauses
  - inference clauses: $c(v_0) \wedge q_1(v_1) \wedge .. \wedge q_n(v_n) \rightarrow q(v)$
  - property clauses
    - safety: $c(v_0) \wedge q_1(v_1) \wedge .. \wedge q_n(v_n) \rightarrow d(v)$
    - termination: dwf(q(v,v'))

HSF - Horn clause solving

- Find solutions for queries, e.g., Inv, TransInv
- Counterexample guided abstraction refinement
  - abstract inference
  - are property clauses satisfied?
    - counterexample: recursion-free Horn clauses
  - abstraction refinement
    - safety: solving rec.-free Horn clauses
      [Gupta, Popeea, Rybalchenko - POPL 2011]
    - termination: solving rec.-free Horn clauses with wf
      [Popeea, Rybalchenko - TACAS 2012]

Proof rules

Init(V) -> Inv(V)
Inv(V) ∧ Step(V, V') -> Inv(V')
Inv(V) ∧ Error(V) -> false

Transition system is safe

true -> Pre(n)
Pre(n) ∧ n>0 -> Pre(n-1)
Pre(n) ∧ n>0 ∧ Post(n-1,s) -> Post(n,s+n)
Pre(n) ∧ n<=0 -> Post(n,0)
Post(n,s) -> s>=0

Functional program is safe

Inv(V) ∧ Step(V, V') -> TransInv(V, V')
TransInv(V, V') ∧ Step(V', V'') ->
TransInv(V, V'')
dwf(TransInv(V, V'))

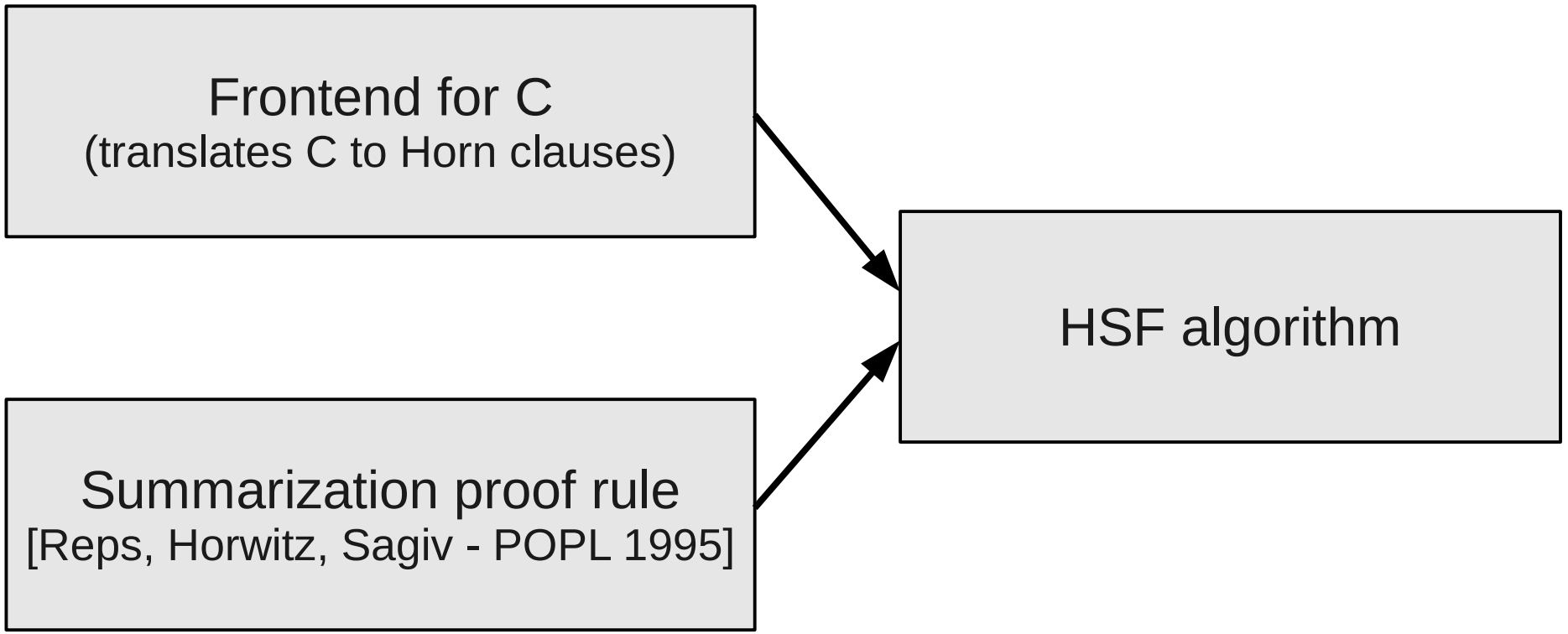
$Init(V) \wedge Step_i(V,V') \rightarrow T_i(V,V')$
$T_i(V,V') \wedge Step_i(V',V'') \rightarrow T_i(V,V'')$
$T_i(V,V') \wedge Step_i(V', V'') \rightarrow T_i(V',V'')$
$(\vee_{j \neq i}\ Init(V) \wedge Step_j(V,V')) \rightarrow E_i(V,V')$
$(\vee_{j \neq i}\ T_j(V,V') \wedge Step_j(V',V'')) \rightarrow E_i(V',V'')$
$Init(V) \wedge E_i(V,V') \rightarrow T_i(V,V')$
$T_i(V,V') \wedge E_i(V',V'') \rightarrow T_i(V,V'')$
$T_i(V,V') \wedge E_i(V', V'') \rightarrow T_i(V',V'')$
$dwf(T_1(V,V') \wedge .. \wedge T_N(V,V'))$

Multi-threaded program terminates

Outline

- Programs, properties, and proof rules
  - Transition systems
  - Reachability, termination
- Proof rules as Horn Clauses + DWF
- Experience with software verifiers

HSF(C)


Frontend for C
(translates C to Horn clauses)
Summarization proof rule
[Reps, Horwitz, Sagiv - POPL 1995]
HSF algorithm

HSF(C) competition candidate

[TACAS 2012]

ControlFlowInteger category:
- 96 benchmarks
- 207.2 kloc

Place	Tool	Points (144 max)
1st	CPAChecker-ABE	141
2nd	CPAChecker-Memo	140
3rd	**HSF(C)**	**140**
4th	ESBMC	102
…	…	

94 correct results in 80 minutes
2 time/outs

More software verifiers

- HSF with different proof rules
  - Safety for procedural programs
  - Termination for procedural programs
  - Safety for multi-threaded programs
  - Safety for OCaml programs

Safety for procedural programs

- Numerical benchmarks, safety from bound overflows
- **Blast**, **CPAchecker**

Program	BLAST	CPAchecker	HSF
Numerical Recipes			
amebsa	FAIL	T/O	**0.2s**
amotsa	FAIL	2.3s	**0.2s**
bandec	T/O	T/O	T/O
choldc	14.1s	2.7s	**2.6s**
crank	6.0s	1.9s	**0.9s**
cyclic	FAIL	T/O	**2.2s**
four1	FAIL	T/O	T/O
lop	FAIL	FAIL	**0.7s**
pzextr	11.2s	FAIL	**0.1s**
qrdcmp	75.2s	T/O	**12.2s**
qrsolv	FAIL	3.3s	**0.1s**
rsolv	T/O	33.6s	**0.2s**
spline	FAIL	1.6s	**0.3s**
tridag	4.1s	1.5s	**0.3s**

Termination for procedural programs

- Numerical benchmarks

Program	HSF
Terminating loops	
broydn (33 cutpoints)	591s
elmhes (9 cutpoints)	23.0s
jacobi (15 cutpoints)	16.0s
ludcmp (11 cutpoints)	4.2s
qrdcmp (9 cutpoints)	189s
rlft3 (7 cutpoints)	16.1s
spctrm (14 cutpoints)	86s

Safety for multi-threaded programs

- Mutual exclusion protocols, models for device drivers
- **Threader**

Program	Threader	HSF
Multi-threaded programs		
Fig2-cex-BUG	0.2s	**0.1s**
Fig2-fixed	0.8s	**0.7s**
Fig4-cex-BUG	4.5s	**0.5s**
Fig4-fixed	1.5s	**0.5s**
Bluetooth2	29.1s	**12.9s**
Bluetooth2-fixed	3.7s	**0.2s**
Bluetooth3-fixed	135s	**18.6s**
Scull	129s	**11.6s**
Dekker	11.1s	**4.0s**
Peterson	4.7s	**3.7s**
Readers-writer-lock	0.2s	**0.1s**
Time varying mutex	11.8s	**9.8s**
Szymanski	32s	**8.7s**
NaiveBakery	**2.5s**	2.6s
Bakery	105s	**32.4s**
Lamport	121s	**30.5s**
QRCU	34.5s	**15.4s**

Safety for OCaml Programs

- Array manipulating programs, safety from bound overflows
- **HMC** based on refinement typing + abstraction refinement

Program	HMC	HSF
OCaml programs (correct / buggy)		
na_dotprod-m	**0.04s / 0.04s**	0.11s / 0.06s
na_arraymax-m	0.32s / **0.05s**	**0.05s** / 0.07s
na_bcopy-m	0.09s / 5.94s	**0.06s / 0.07s**
na_bsearch-m	0.91s / 0.10s	**0.03s / 0.02s**
na_insertsort-m	**0.03s / 0.03s**	1.78s / 0.04s
mult-cps-m	**0.03s / 0.03s**	**0.03s / 0.03s**
mult-all-m	0.03s / 0.03s	**0.01s / 0.01s**
sum-all-m	0.03s / 0.03s	**0.01s / 0.01s**
sum-acm-m	0.04s / 0.03s	**0.01s / 0.01s**

HSF and related work

- Software verification tools
  - Slam, Blast, Terminator, CPAchecker, DSolve, ...
- Verifiers - target for automated synthesis
  - XSB: generates model checkers for CCS programs
  - Getafix: generates model checkers for boolean programs

HSF: generates model checkers for C and OCaml programs competitive with mature software verification tools

Future work

- Add atomicity and reduction to multi-threaded proof rules
  [Elmas, Qadeer, Tasiran - POPL 2009]
- More efficient transition invariant check
  [Kroening, Sharygina, Tsitovich, Wintersteiger - CAV 2010]
- Fairness assumptions for rely-guarantee reasoning
  [Cohen, Namjoshi, Sa'ar - CAV 2010]
- Combine symmetry reduction and rely-guarantee reasoning
  [Donaldson, Kaiser, Kroening, Wahl - CAV 2011]
- Conditional termination for multi-threaded programs
  [Iosif, Bozga, Konečný - TACAS 2012]
- Dynamic creation of threads using counter abstraction
  [Henzinger, Jhala, Majumdar - PLDI 2004]

Conclusion

- Verification task representation
  Horn clauses + disjunctive well-foundedness

- Solving algorithm
  predicate abstraction and refinement

Synthesizing software verifiers from proof rules
[Grebenshchikov, Lopes, Popeea, Rybalchenko - PLDI 2012]

Additional Slides

Proof Rules

- Termination via transition invariants
  [Podelski, Rybalchenko - LICS'04]
- CFL reachability
  [Reps, Horwitz, Sagiv - POPL'95]
- Refinement typing for OCaml
  [Rondon, Kawaguchi, Jhala - PLDI'08]
- Rely/guarantee + safety properties
  [Gupta, Popeea, Rybalchenko - POPL'11]
- Rely/guarantee + termination
  [Popeea, Rybalchenko - TACAS'12]

Preprocessing Horn Clauses

- Remove trivially valid clauses
- Clause inlining
- Trim set of variables in heads
- Houdini (for projection and/or for initial abstraction)
- Simple projection
- Dataflow projection (forward and backwards)
- Remove duplicated queries (on the left)
- Remove subsumed clauses
- ...

Safety for Procedural Programs

ntdrivers			
cdaudio_simpl1	64.8s	**24.9s**	393s
diskperf_simpl1	41.9s	**21.1s**	213s
floppy_simpl3	30.9s	**10.3s**	67s
floppy_simpl4	50.1s	**16.3s**	139s
kbfiltr_simpl1	3.7s	**2.7s**	3.2s
kbfiltr_simpl2	5.5s	**4.0s**	7.2s
cdaudio_simpl1_BUG	29.3s	**12.3s**	351s
floppy_simpl3_BUG	**1.5s**	7.5s	96s
floppy_simpl4_BUG	**1.5s**	12.9s	135s
kbfiltr_simpl2_BUG	**3.1s**	3.2s	14.7s

ssh-simplified			
s3_clnt_1	103s	8.0s	**7.4s**
s3_clnt_2	147s	59s	**4.2s**
s3_clnt_3	FAIL	7.4s	**7.3s**
s3_clnt_4	80s	9.7s	**6.6s**
s3_srvr_1	FAIL	23.2s	**9.8s**
s3_srvr_2	FAIL	40.0s	**10.1s**
s3_srvr_3	FAIL	**9.9s**	36.1s
s3_srvr_4	FAIL	11.3s	**8.9s**
s3_srvr_6	110s	**41.9s**	49.5s
s3_srvr_7	FAIL	**14.0s**	133s
s3_srvr_8	41.5s	**11.1s**	23.1s
s3_clnt_1_BUG	4.5s	3.0s	**1.3s**
s3_clnt_2_BUG	4.9s	2.6s	**1.4s**
s3_clnt_3_BUG	4.9s	2.9s	**1.3s**
s3_clnt_4_BUG	4.6s	3.0s	**1.4s**
s3_srvr_1_BUG	FAIL	**2.6s**	3.1s
s3_srvr_2_BUG	66s	2.5s	**2.2s**

Solving rec.-free Horn clauses with well-foundedness conditions

Stem

Lasso

1. HC = { Init(V) ∧ $move_1$(a0,a1) ∧ ... -> T_{11}(V,V'),
   T_{11}(V,V') ∧ $move_1$(a1,a2) ∧ ... -> T_{12}(V',V''),
   T_{12}(V,V') ∧ $move_1$(a2,a1) ∧ ... -> T_{13}(V,V''),
   T_{13}(V,V') -> WF(V,V') }

2. SOL^{least}(T_{11}(V,V')) = (l=0 ∧ l'=1 ∧ x'=x ∧ $move_1$(a0,a1) ∧ pc_2=pc_2'=b0)
   SOL^{least}(T_{12}(V,V')) = (l=1 ∧ l'=1 ∧ x>0 ∧ x'=x ∧ $move_1$(a1,a2) ∧ pc_2=pc_2'=b0)
   SOL^{least}(T_{13}(V,V')) = (l=1 ∧ l'=1 ∧ x>0 ∧ x'=x-1 ∧ $move_1$(a1,a1) ∧ pc_2=pc_2'=b0)

3. WF(V,V') = (x>0 ∧ x'<x)

4. HC_1 = { Init(V) ∧ $move_1$(a0,a1) ∧ ... -> T11(V,V'),
   T11(V,V') ∧ $move_1$(a1,a2) ∧ ... -> T12(V',V''),
   T12(V,V') ∧ $move_1$(a2,a1) ∧ ... -> T13(V,V''),
   T13(V,V') -> x>0 ∧ x'<x }

5. SOL(T_{11}(V,V')) = true
   SOL(T_{12}(V,V')) = (x>0 ∧ x'=x)
   SOL(T_{13}(V,V')) = (x>0 ∧ x'<x)
   SOL(WF(V,V'))= (x>0 ∧ x'<x)

3 new predicates:
x>0, x'=x, x'<x

Example

Find Inv(V) such that:

1) pc = A → Inv(V)

2) Inv(V) ∧
((pc=A ∧ pc'=B ∧ s'=0 ∧ i'=i) ∨
(pc=B ∧ pc'=B ∧ i>0 ∧ s'=s+i ∧ i'=i-1) ∨
(pc=B ∧ pc'=C ∧ i≤0 ∧ s'=s ∧ i'=i))
→ Inv(V')

3) Inv(V) ∧ pc=C ∧ s<0 -> ⊥

- Inference (empty abstraction):

  a1) Inv(v) = true via ((), 1)

  a1) fails 3

- Counterexample analysis:

  $pc_0 = A \rightarrow Inv_0(pc_0,s_0,i_0)$

  $Inv_0(pc_0,s_0,i_0) \wedge pc_0=c \wedge s_0<0 \rightarrow \bot$

- Interpolants:

  $Inv_0(pc_0,s_0,i_0) = pc_0=A$

- Refine abstraction: { pc=A }

Example

Find Inv(V) such that:

1) pc = A → Inv(V)

2) Inv(V) ∧
((pc=A ∧ pc'=B ∧ s'=0 ∧ i'=i) ∨
(pc=B ∧ pc'=B ∧ i>0 ∧ s'=s+i ∧ i'=i-1) ∨
(pc=B ∧ pc'=C ∧ i≤0 ∧ s'=s ∧ i'=i))
→ Inv(V')

3) Inv(V) ∧ pc=C ∧ s<0 -> ⊥

- Inference with { pc=A, s≥0 }

  a1) Inv(v) = pc=A via ((), 1)

  a2) Inv(v) = s≥0 via (a1, 2)

- Counterexample analysis:

  both a1) and a2) satisfy clause 3

Solution:
Inv(V) = (pc=A ∨ s ≥ 0)