Synthesizing Software Verifiers from Proof Rules

Sergey Grebenshchikov, Nuno Lopes,
Corneliu Popeea, Andrey Rybalchenko

INESC-ID / TU München

Developing Verifiers Today

Developing Verifiers Today

Program Model

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

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, ...

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

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, ...

+

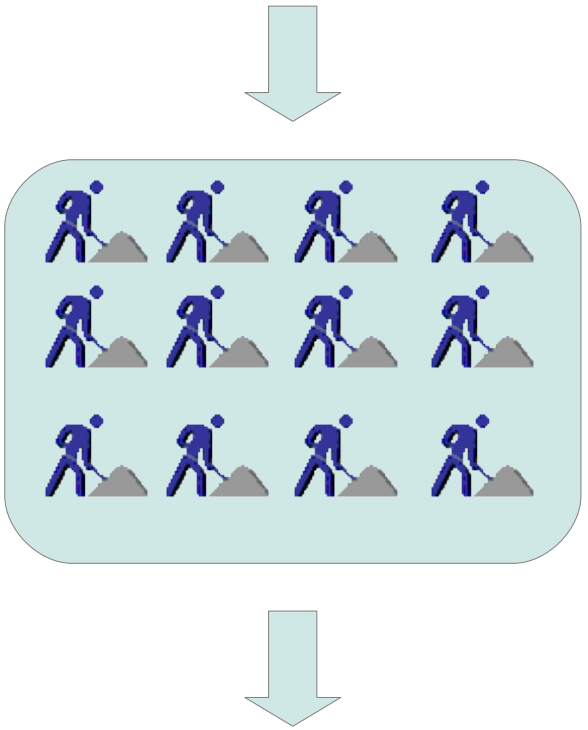
Complex verification effort

=

Verification Tool

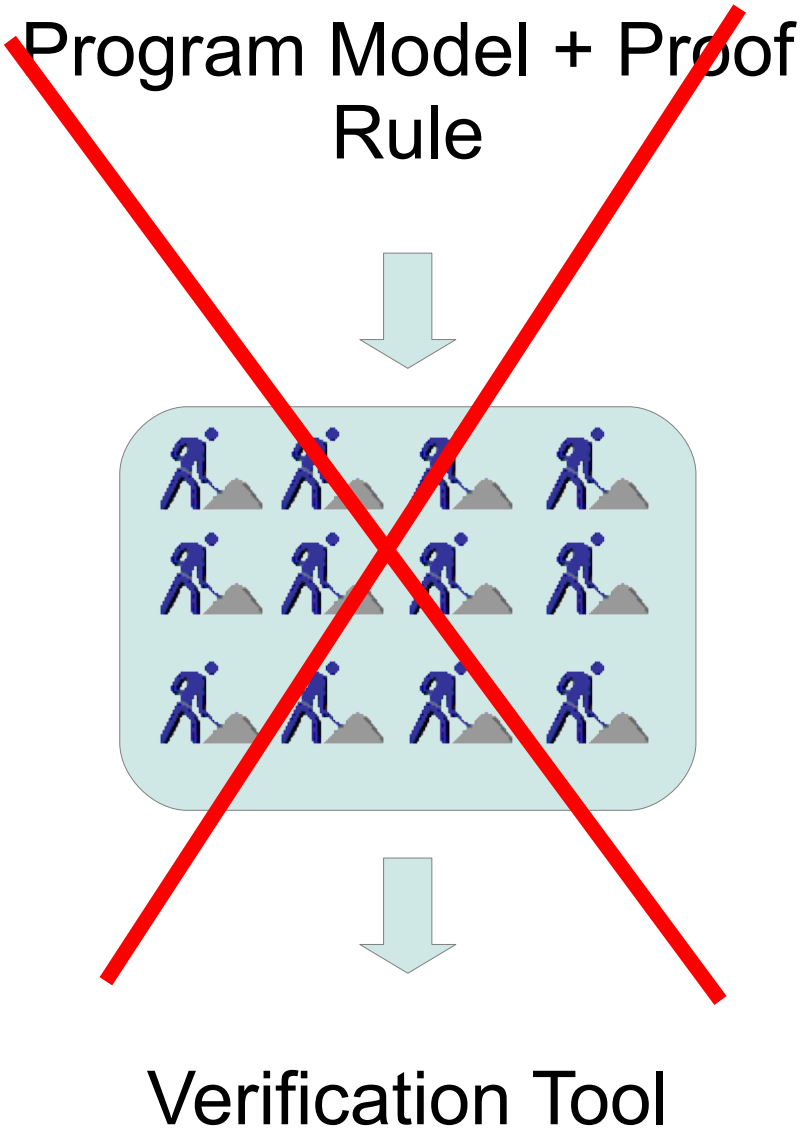
Developing Verifiers Today

Program Model + Proof Rule

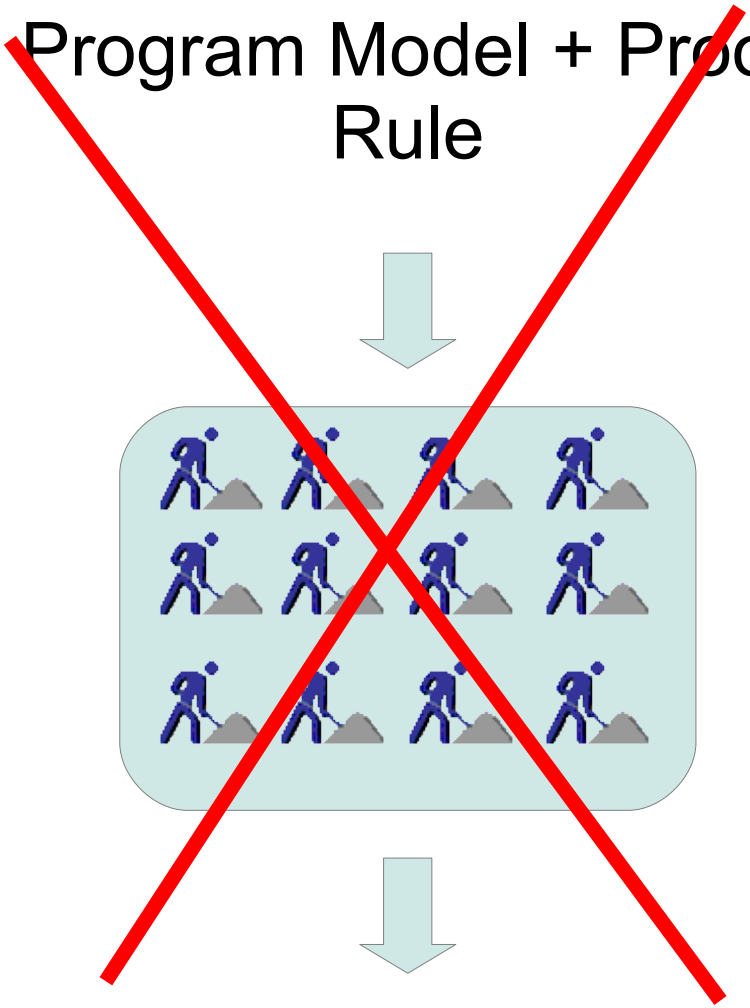


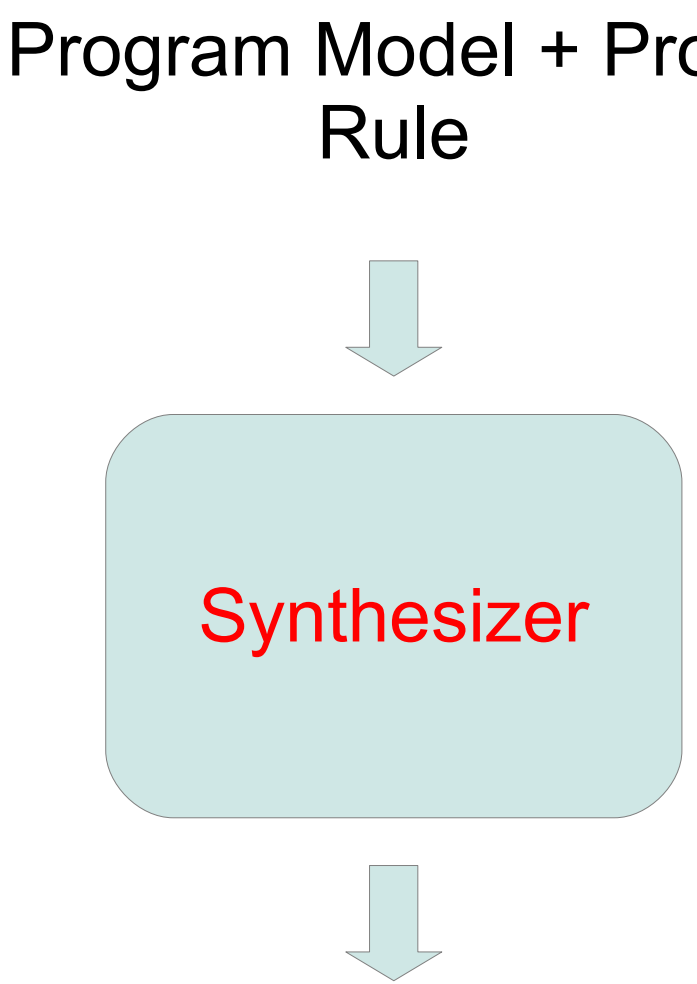
Verification Tool

Our Goal



Our Goal


Program Model + Proof Rule
Verification Tool


Program Model + Proof Rule
Synthesizer
Verification Tool

Outline

- Programs, properties, and proof rules
- Representation as Horn clauses
- HSF: Solving Horn Clauses
- Evaluation

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);
}
```

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)

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')

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)

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)

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)

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)

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)

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

Invariance Proof Rule

- Describes set of reachable states

$$\frac{\begin{array}{l} Init(V) \rightarrow Inv(V) \\ Inv(V) \wedge Step(V, V') \rightarrow Inv(V') \\ Inv(V) \wedge Error(V) \rightarrow false \end{array}}{\text{Transition system is safe}}$$

Verification

```
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)

Init(V) → Inv(V)
Inv(V) ∧ Step(V, V') → Inv(V')
Inv(V) ∧ Error(V) → false

Transition system is safe

Verification

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

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

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

Find Inv(V) such that:

1) $pc = A \rightarrow Inv(V)$

2) $Inv(V) \wedge$
$((pc=A \wedge pc'=B \wedge s'=0 \wedge i'=i) \vee$
$(pc=B \wedge pc'=B \wedge i>0 \wedge s'=s+i \wedge i'=i-1) \vee$
$(pc=B \wedge pc'=C \wedge i\leq 0 \wedge s'=s \wedge i'=i))$
$\rightarrow Inv(V')$

3) $Inv(V) \wedge pc=C \wedge s<0 \rightarrow false$

Verification

Solution:
Inv(V) = (pc=A ∨ s ≥ 0)

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

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

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

Find Inv(V) such that:

1) $pc = A \rightarrow Inv(V)$

2) $Inv(V) \wedge$
$((pc=A \wedge pc'=B \wedge s'=0 \wedge i'=i) \vee$
$(pc=B \wedge pc'=B \wedge i>0 \wedge s'=s+i \wedge i'=i-1) \vee$
$(pc=B \wedge pc'=C \wedge i\leq 0 \wedge s'=s \wedge i'=i))$
$\rightarrow Inv(V')$

3) $Inv(V) \wedge pc=C \wedge s<0 \rightarrow$ false

Termination of Transition Systems

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

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

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

Termination of Transition Systems

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

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

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

$Inv(V) \wedge Step(V, V') \rightarrow TransInv(V, V')$

$TransInv(V, V') \wedge Step(V', V'') \rightarrow TransInv(V, V'')$

$dwf(TransInv(V, V'))$

Transition system terminates

Termination of Transition Systems

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

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

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

$Inv(V) \wedge Step(V, V') \rightarrow TransInv(V, V')$

$TransInv(V, V') \wedge Step(V', V'') \rightarrow TransInv(V, V'')$

$dwf(TransInv(V, V'))$

Transition system terminates

exists $WF_1(V,V')$, …, $WF_N(V,V')$:

$TransInv(V,V') \rightarrow WF_1(V,V') \vee .. \vee W_F(V,V')$

Termination of Transition Systems

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

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

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

$Inv(V) \wedge Step(V, V') \rightarrow TransInv(V, V')$
$TransInv(V, V') \wedge Step(V', V'') \rightarrow$
$\quad TransInv(V, V'')$
$dwf(TransInv(V, V'))$

Transition system terminates

Termination of Transition Systems

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

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

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

$Inv(V) \wedge Step(V, V') \rightarrow TransInv(V, V')$
$TransInv(V, V') \wedge Step(V', V'') \rightarrow TransInv(V, V'')$
$dwf(TransInv(V, V'))$

Transition system terminates

Solution:

$Inv(V) = (pc=A \vee s \geq 0)$
$TransInv(V, V') = (pc=A \wedge pc'=B) \vee$
$(pc=A \wedge pc'=C) \vee$
$(pc=B \wedge pc'=C) \vee$
$(i' < i \wedge i > 0)$

Representation as Horn Clauses

- Proof Rules = Horn Clauses + DWF

Representation as Horn Clauses

- Proof Rules = Horn Clauses + DWF

$$\text{Init}(V) \rightarrow \text{Inv}(V)$$
$$\text{Inv}(V) \wedge \text{Step}(V, V') \rightarrow \text{Inv}(V')$$
$$\text{Inv}(V) \wedge \text{Error}(V) \rightarrow \text{false}$$

Transition system is safe

Representation as Horn Clauses

- Proof Rules = Horn Clauses + DWF

$$\text{Init}(V) \rightarrow \text{Inv}(V)$$
$$\text{Inv}(V) \wedge \text{Step}(V, V') \rightarrow \text{Inv}(V')$$
$$\text{Inv}(V) \wedge \text{Error}(V) \rightarrow \text{false}$$

Transition system is safe

$$\text{Inv}(V) \wedge \text{Step}(V, V') \rightarrow \text{TransInv}(V, V')$$
$$\text{TransInv}(V, V') \wedge \text{Step}(V', V'') \rightarrow \text{TransInv}(V, V'')$$
$$\text{dwf}(\text{TransInv}(V, V'))$$

Transition system terminates

Representation as Horn Clauses

- Proof Rules = Horn Clauses + DWF

$$\text{Init}(V) \rightarrow \text{Inv}(V)$$
$$\text{Inv}(V) \wedge \text{Step}(V, V') \rightarrow \text{Inv}(V')$$
$$\text{Inv}(V) \wedge \text{Error}(V) \rightarrow \text{false}$$

Transition system is safe

$$\text{Inv}(V) \wedge \text{Step}(V, V') \rightarrow \text{TransInv}(V, V')$$
$$\text{TransInv}(V, V') \wedge \text{Step}(V', V'') \rightarrow \text{TransInv}(V, V'')$$
$$\text{dwf}(\text{TransInv}(V, V'))$$

Transition system terminates

$$\text{Init}(V) \wedge V'=V \rightarrow \text{Summ}(V,V')$$
$$\text{Summ}(V,V') \wedge \text{Step}(V', V'') \rightarrow \text{Summ}(V,V'')$$
$$\text{Summ}(V,V') \wedge \text{Call}(V', V'') \wedge V'''=V'' \rightarrow \text{Summ}(V'',V''')$$
$$\text{Summ}(V,V') \wedge \text{Call}(V', V'') \wedge \text{Summ}(V'', V''') \wedge \text{Return}(V''', V'''') \wedge \text{Local}(V', V'''') \rightarrow \text{Summ}(V,V'''')$$
$$\text{Summ}(V,V') \wedge \text{Error}(V') \rightarrow \text{false}$$

Procedural program is safe

Representation as Horn Clauses

- Proof Rules = Horn Clauses + DWF

$\text{Init}(V) \rightarrow \text{Inv}_i(V)$

$\text{Inv}_i(V) \wedge \text{Step}_i(V, V') \rightarrow \text{Inv}_i(V')$

$(\bigvee_{j \neq i} \text{Inv}_j(V) \wedge \text{Step}_j(V,V')) \rightarrow \text{Env}_i(V,V')$

$\text{Inv}_i(V) \wedge \text{Env}_i(V,V') \rightarrow \text{Inv}_i(V')$

$\text{Inv}_1(V) \wedge .. \wedge \text{Inv}_N(V) \wedge \text{Error}(V) \rightarrow \text{false}$

Multi-threaded program is safe

$\text{Inv}(V) \wedge \text{Step}(V, V') \rightarrow \text{TransInv}(V, V')$

$\text{TransInv}(V, V') \wedge \text{Step}(V', V'') \rightarrow \text{TransInv}(V, V'')$

$\text{dwf}(\text{TransInv}(V, V'))$

Transition system terminates

$\text{Summ}(V,V'')$

$\text{Summ}(V,V') \wedge \text{Call}(V', V'') \wedge V'''=V'' \rightarrow \text{Summ}(V'',V''')$

$\text{Summ}(V,V') \wedge \text{Call}(V', V'') \wedge \text{Summ}(V'', V''') \wedge \text{Return}(V''', V'''') \wedge \text{Local}(V', V'''') \rightarrow \text{Summ}(V,V'''')$

$\text{Summ}(V,V') \wedge \text{Error}(V') \rightarrow \text{false}$

Procedural program is safe

Representation as Horn Clauses

- Proof Rules = Horn Clauses + DWF

$$\text{Init}(V) \rightarrow \text{Inv}_i(V)$$
$$\text{Inv}_i(V) \wedge \text{Step}_i(V, V') \rightarrow \text{Inv}_i(V')$$
$$(\bigvee_{j \neq i} \text{Inv}_j(V) \wedge \text{Step}_j(V,V')) \rightarrow \text{Env}_i(V,V')$$
$$\text{Inv}_i(V) \wedge \text{Env}_i(V,V') \rightarrow \text{Inv}_i(V')$$
$$\text{Inv}_1(V) \wedge .. \wedge \text{Inv}_N(V) \wedge \text{Error}(V) \rightarrow \text{false}$$

Multi-threaded program is safe

$$\text{Summ}(V,V') \wedge \text{Call}(V', V'') \wedge \text{Su}$$
$$\text{Return}(V''', V'''') \wedge \text{Local}(V$$
$$\text{Summ}(V,V') \wedge \text{Error}(V') \rightarrow \text{false}$$

Procedural program is safe

$$\text{Init}(V) \wedge \text{Step}_i(V,V') \rightarrow T_i(V,V')$$
$$T_i(V,V') \wedge \text{Step}_i(V',V'') \rightarrow T_i(V,V'')$$
$$T_i(V,V') \wedge \text{Step}_i(V', V'') \rightarrow T_i(V',V'')$$
$$(\bigvee_{j \neq i} \text{Init}(V) \wedge \text{Step}_j(V,V')) \rightarrow E_i(V,V')$$
$$(\bigvee_{j \neq i} T_j(V,V') \wedge \text{Step}_j(V',V'')) \rightarrow E_i(V',V'')$$
$$\text{Init}(V) \wedge E_i(V,V') \rightarrow T_i(V,V')$$
$$T_i(V,V') \wedge E_i(V',V'') \rightarrow T_i(V,V'')$$
$$T_i(V,V') \wedge E_i(V', V'') \rightarrow T_i(V',V'')$$
$$\text{dwf}(T_1(V,V') \wedge ... \wedge T_N(V,V'))$$

Multi-threaded program terminates

Challenges

- Generic solving algorithm
- Infinite state systems

Outline

- Programs, properties, and proof rules
- Representation as Horn clauses
- HSF: Solving Horn Clauses
- Evaluation

HSF: Solving Horn Clauses

1) Finite unfolding of clauses
2) Solve recursion-free clauses
3) Generalize solution to unbounded unfolding

Example

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

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

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

$$\text{Init}(V) \rightarrow \text{Inv}(V)$$

$$\text{Inv}(V) \wedge \text{Step}(V, V') \rightarrow \text{Inv}(V')$$

$$\text{Inv}(V) \wedge \text{Error}(V) \rightarrow \text{false}$$

Step 1: Unfolding (n = 0)

$\mathrm{Init}(V) \rightarrow \mathrm{Inv}_1(V)$

$\mathrm{Inv}_1(V) \wedge \mathrm{Error}(V) \rightarrow \mathrm{false}$

Step 1: Unfolding (n = 1)

$\text{Init}(V) \rightarrow \text{Inv}_1(V)$

$\text{Inv}_1(V) \wedge \text{Step}(V, V') \rightarrow \text{Inv}_2(V')$

$\text{Inv}_2(V) \wedge \text{Error}(V) \rightarrow \text{false}$

Step 1: Unfolding (n = 2)

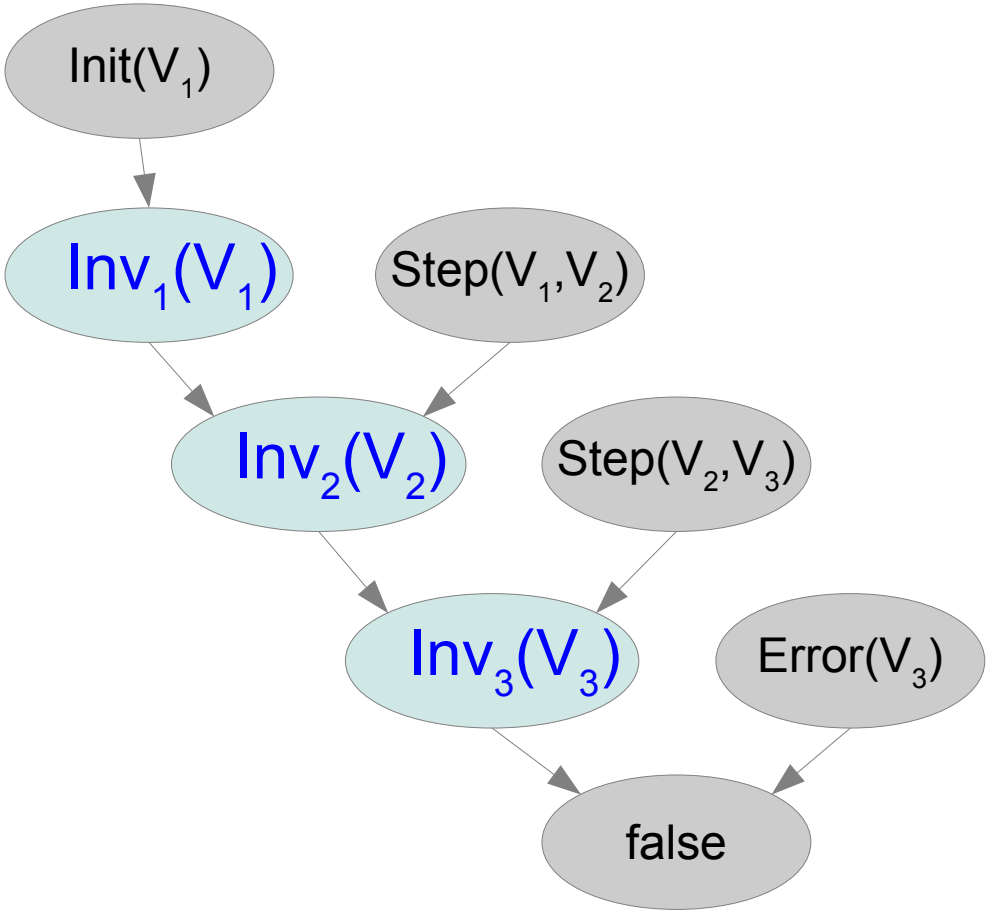
$\text{Init}(V) \rightarrow \text{Inv}_1(V)$

$\text{Inv}_1(V) \wedge \text{Step}(V, V') \rightarrow \text{Inv}_2(V')$

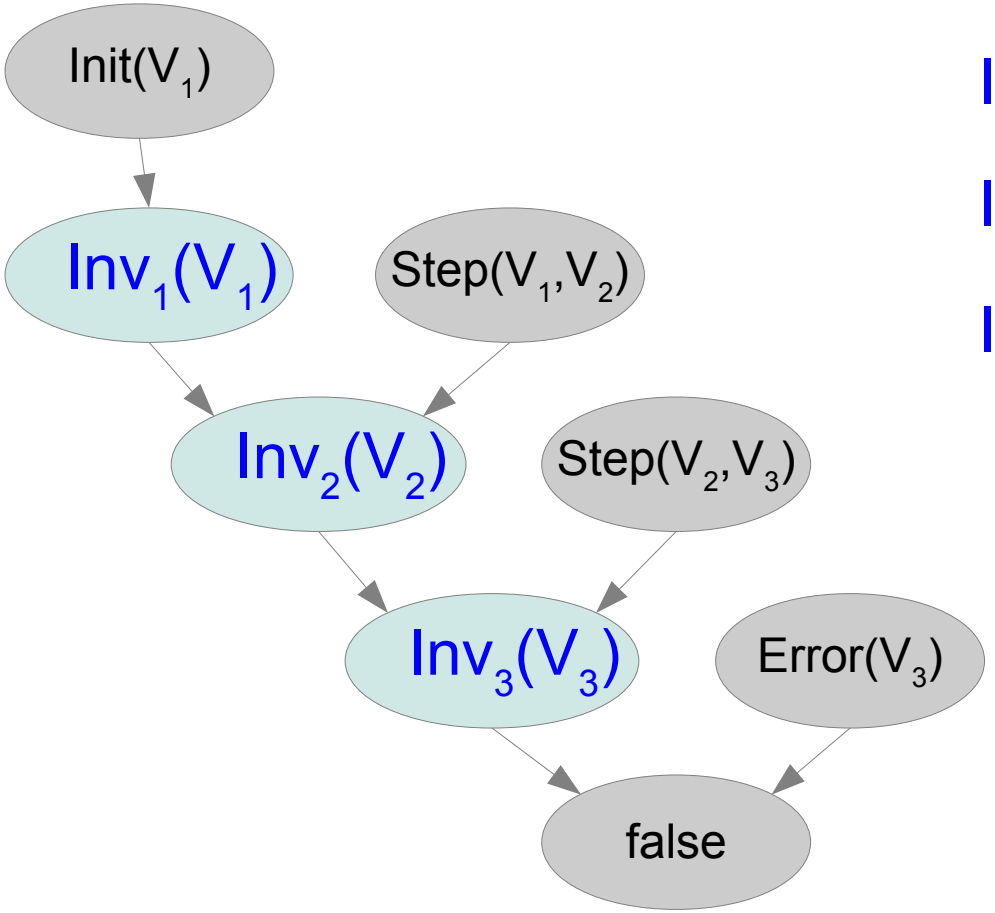
$\text{Inv}_2(V) \wedge \text{Step}(V, V') \rightarrow \text{Inv}_3(V')$

$\text{Inv}_3(V) \wedge \text{Error}(V) \rightarrow \text{false}$

Step 2: Solve Rec.-free Clauses



Step 2: Solve Rec.-free Clauses



- Solution:

  $Inv_1(V)$ = (pc=A)

  $Inv_2(V)$ = (pc=B ∧ s=0)

  $Inv_3(V)$ = (pc=B ∧ s≥0 ∨ pc=C ∧ s≥0)

Step 3: Generalize Solution

- Extrapolate solution to the recursive case
- Obtain boolean combinations of the predicates

HSF: Our Implementation

1) Unfolding:
   - BFS
2) Rec.-free clause solver:
   - LI+UF [Gupta et al, POPL 2011, APLAS 2011]
   - Termination for LI [Popeea, Rybalchenko, TACAS 2012]
3) Generalization:
   - Predicate (cartesian) abstraction

Outline

- Programs, properties, and proof rules
- Representation as Horn clauses
- HSF: Solving Horn Clauses
- Evaluation

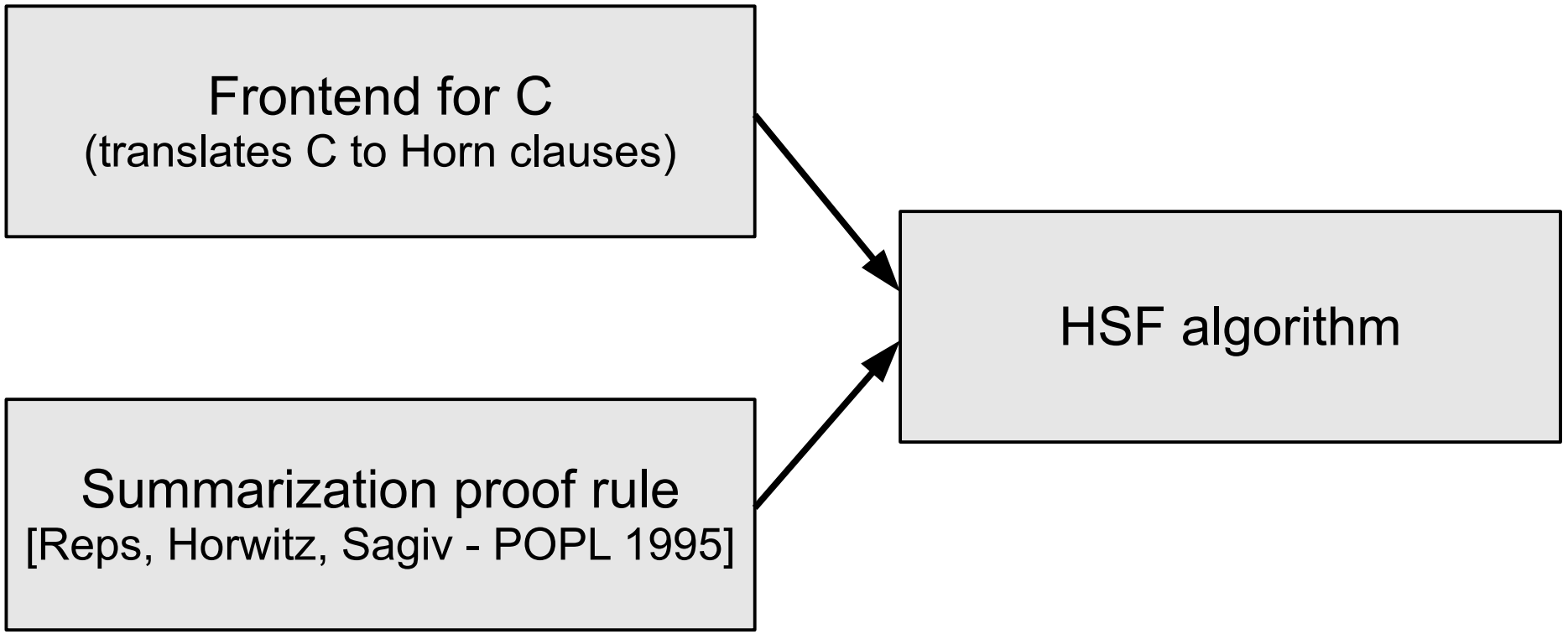
Evaluation

Evaluation

- HSF(C) and Software Verification competition results
- Comparison with specialized tools for different proof rules

HSF(C)

HSF(C)


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

Software Verification Competition (TACAS'12)

Software Verification Competition (TACAS'12)

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

(ControlFlowInteger category, 15 min timeout, 10 participants)
> 200k LOC

Experiment #1

- Frontend for C
- Termination proof rule using transition invariants [Podelski, Rybalchenko, LICS 2004]
- Benchmarks: Numerical Recipes book

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

Experiment #2

- Frontend for C with Pthread model
- Rely-guarantee proof rule [Jones - TOPLAS 1983]
- Benchmarks: device driver snippets and mutual exclusion protocols
- Comparison with Threader [Gupta et al - CAV 2011]

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**

Experiment #3

- Frontend for OCaml
- Proof rule based on liquid type system
  [Rondon, Kawaguchi, Jhala - PLDI 2008]
- Benchmarks: array programs
- Compare with HMC
  [Jhala et al - CAV 2011]

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**

Related work

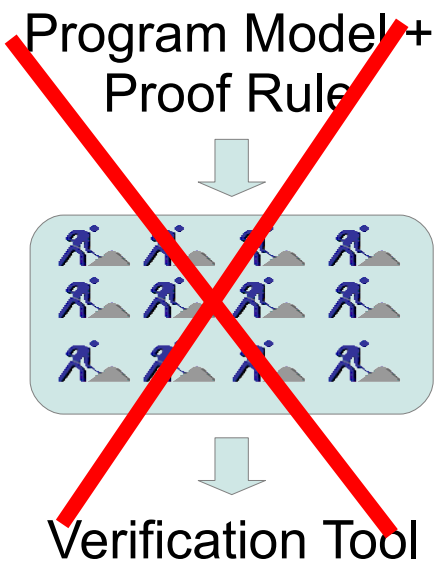
Related work

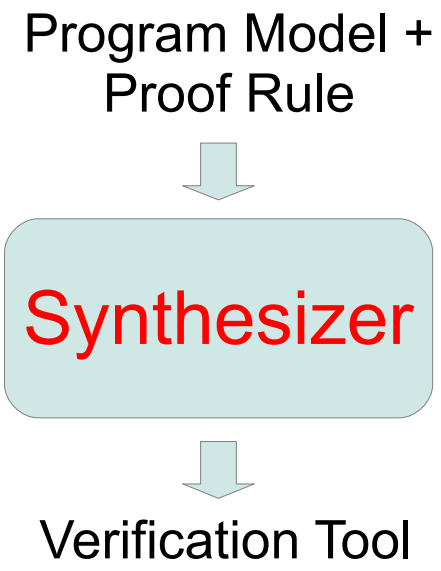
- Software verification tools
  - Slam, Blast, Terminator, CPAchecker, DSolve, ...
- Verifiers - target for automated synthesis
  - XSB: generates model checkers for CCS programs
  - Getafix: generates model checkers for boolean programs

Conclusion

Conclusion

- Programs and proof rules represented as Horn clauses + disjunctive well-foundedness
- Presented solving algorithm for Horn clauses





Additional Slides

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: $dwf(q(v,v'))$

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 (in the body)
- Remove subsumed clauses
- ...

Proof Rules

- CFL reachability
  [Reps, Horwitz, Sagiv - POPL 1995]
- Termination via transition invariants
  [Podelski, Rybalchenko - LICS 2004]
- Refinement typing for OCaml
  [Rondon, Kawaguchi, Jhala - PLDI 2008]
- Rely/guarantee + safety properties
  [Gupta, Popeea, Rybalchenko - POPL 2011]
- Rely/guarantee + termination
  [Popeea, Rybalchenko - TACAS 2012]

Inference Rules

$$c(v_0) \wedge q_1(v_1) \wedge \cdots \wedge q_n(v_n) \rightarrow h(v)$$

$$\textsc{RInit}\ \frac{c(v_0) \rightarrow q(v) \in \mathcal{I}}{\alpha(c(v_0), \mathit{Preds}(q)) \in \mathit{Inferred}(q)}$$

$$\textsc{RStep}\ \frac{\varphi_1(v_1) \in \mathit{Inferred}(q_1) \quad \ldots \quad \varphi_n(v_n) \in \mathit{Inferred}(q_n) \qquad c(v_0) \wedge \bigwedge_{i=1}^{n} q_i(v_i) \rightarrow q(v) \in \mathcal{I}}{\alpha(c(v_0) \wedge \bigwedge_{i=1}^{n} \varphi_i(v_i), \mathit{Preds}(q)) \in \mathit{Inferred}(q)}$$