# Solving Existentially Quantified Horn Clauses

Corneliu Popeea

- joint work with Tewodros Beyene and Andrey Rybalchenko -

Technische Universität München

Microsoft Research

# Universal properties ... success story

- Temporal verification of universal properties of various kind of programs
  - Slam, Blast, Astrée, SatAbs, Terminator, Clousot, CPAChecker, AProVE, UFO
- Infer auxiliary assertions
- Reason about infinite-state, complex data domains

# Universal properties ... a recipe

- First ingredient: proof rules

exists **inv** such that

$\quad$ init(v) $\rightarrow$ **inv(v)**

$\quad$ **inv(v)** $\wedge$ next(v,v') $\rightarrow$ **inv(v')**

$\quad$ inv(v) $\rightarrow$ safe(v)

-------------------------------------------

( init(v), next(v,v') ) |= AG safe(v)

exists **inv** and **segm** such that

$\quad$ init(v) $\rightarrow$ **inv(v)**

$\quad$ **inv(v)** $\wedge$ $\neg$dst(v) $\wedge$ next(v,v') $\rightarrow$ **inv(v')**

$\quad$ **inv(v)** $\wedge$ $\neg$dst(v) $\wedge$ next(v,v') $\rightarrow$ **segm(v,v')**

$\quad$ wf(**segm**)

-----------------------------------------------------------

( init(v), next(v,v') ) |= AF dst(v)

- Second ingredient: inference of auxiliary assertions via HSF algorithm [Grebenschikov, Lopes, P, R – PLDI'12]

# What about existential properties?

- One example

exists **inv** such that

$\text{init}(v) \rightarrow \textbf{inv}(v)$

$\textbf{inv}(v) \rightarrow \exists \textbf{v'}: \text{next}(v,v') \land \textbf{inv}(v')$

$\textbf{inv}(v) \rightarrow \text{safe}(v)$

-------------------------------------------

$(\ \text{init}(v),\ \text{next}(v,v')\ ) \models \text{EG safe}(v)$

Our GOAL

solve "existentially quantified Horn clauses"

# Overview

- Solving algorithm E-HSF
- Evaluation: verification for CTL properties of programs
- Other applications / Future directions

# SOLVING ALGORITHM

# Obligations for EG

- Implicit in proof rule notation
  - conjunction between clauses
  - clauses are universally quantified

exists **inv** such that

$\quad (\forall v\colon init(v) \rightarrow inv(v))$

$\quad \wedge (\forall v\colon inv(v) \rightarrow \exists v'\colon next(v,v') \wedge inv(v'))$

$\quad \wedge (\forall v\colon inv(v) \rightarrow safe(v))$

-----------------------------------------------------------------

$\qquad (init(v), next(v,v')) \models EG\ safe(v)$

$\forall\exists$ Horn clauses

# $\forall\exists$ Horn clauses

$\phi(v) \in P$ (background predicates, e.g., QF_LRA)
**q(v)** $\in Q$ (queries)

body ::= **q(v)** | $\phi(v)$ | body $\wedge$ body
head ::= **q(v)** | $\phi(v)$ | wf(**q**)
cl ::= $\forall$v,w: body(v,w) $\rightarrow$ **$\exists$x:** head(w,x)
cls ::= cl $\wedge$ cls | cl

Abbreviations: $\forall\exists$H-clauses
$\forall$H-clauses

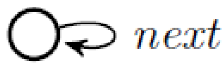
# Steps of E-HSF algorithm

- Skolemization for ∀∃H-clauses
- Start with “true” as witness candidate
  - Solve ∀H-clauses (e.g., use HSF)
  - In case there is a solution for ∀H-clauses, return “sat”
  - Otherwise
    - Replace the candidate witness by a template constraint
    - Look for an instantiation of template parameters (solve recursion-free ∀H-clauses)
    - In case there is no solutio
    - Repeat with the ∀H-solution as a new witness

Solution for ∀∃H-clauses

No solution for ∀∃H-clauses

# Example




```
while (1) {
  x = x+y;
  y = nondet();
}
```

$$v = (x, y)$$
$$init(v) = (y \geq 1)$$
$$next(v, v') = (x' = x + y)$$

**EF (x$\geq$0)**

exists inv(v) and segm(v,v') such that

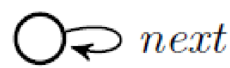
init(v) $\rightarrow$ inv(v)

inv(v) $\wedge$ $\neg$(x$\geq$0) $\rightarrow$ $\exists$v': next(v,v') $\wedge$ inv(v') $\wedge$ segm(v,v')

wf(segm)

# Example

```
while (1) {
  x = x+y;
  y = nondet();
}
```




$$v = (x, y)$$
$$init(v) = (y \geq 1)$$
$$next(v, v') = (x' = x + y)$$

- Witness for existential quantifier

  wit(v,v') = $(x'=x+1 \wedge y'=1)$

- Solutions for other assertions

  inv(v) = $(y \geq 1)$

  segm(v,v') = $(x \leq -1 \wedge x' \geq x+1)$

**Program satisfies CTL specification**

# E-HSF EVALUATION

# E-HSF implementation

- Built in SICStus Prolog
- Input: transition system + CTL property
  - generate ∀∃H-clauses from a given CTL property
  - use HSF for solving ∀H-clauses over linear arithmetic domain, i.e., QF_LRA
  - use Z3 / Barcelogic for solving non-linear constraints

# Experiments

- CTL benchmarks [Cook, Koskinen – PLDI'13]

- For each case we attempt two proofs:
  - P $\models \phi$
  - P $\models \neg\phi$

# Proofs for all correct programs except 2 cases

| Program | Property $\phi$ | $\models_{CTL} \phi$ | | | $\models_{CTL} \neg\phi$ | | |
|---|---|---|---|---|---|---|---|
| | | Result | Time | Name | Result | Time | Name |
| P1 | $AG(a = 1 \rightarrow AF(r = 1))$ | ✓ | 1.2s | 1 | × | 2.7s | 29 |
| P2 | $EF(a = 1 \wedge EG(r \neq 5))$ | ✓ | 0.6s | 30 | × | 5.2s | 2 |
| P3 | $AG(a = 1 \rightarrow EF(r = 1))$ | ✓ | 4.8s | 3 | × | 0.1s | 31 |
| P4 | $EF(a = 1 \wedge AG(r \neq 1))$ | ✓ | 0.6s | 32 | × | 0.4s | 4 |
| P5 | $AG(s = 1 \rightarrow AF(u = 1))$ | ✓ | 6.1s | 5 | × | 0.2s | 33 |
| P6 | $EF(s = 1 \wedge EG(u \neq 1))$ | ✓ | 1.4s | 34 | × | 3.6s | 6 |
| P7 | $AG(s = 1 \rightarrow EF(u = 1))$ | ✓ | 12.9s | 7 | × | 0.2s | 35 |
| P8 | $EF(s = 1 \wedge AG(u \neq 1))$ | ✓ | 44.7s | 36 | × | 3.8s | 8 |
| P9 | $AG(a = 1 \rightarrow AF(r = 1))$ | ✓ | 51.3s | 9 | × | 120.0s | 37 |
| P10 | $EF(a = 1 \wedge EG(r \neq 1))$ | ✓ | 132.0s | 38 | × | 45.9s | 10 |
| P11 | $AG(a = 1 \rightarrow EF(r = 1))$ | ✓ | 67.6s | 11 | × | 3.9s | 39 |
| P12 | $EF(a = 1 \wedge AG(r \neq 1))$ | ✓ | 67.9s | 12 | × | 3.8s | 40 |
| P13 | $AF(io = 1) \vee AF(ret = 1)$ | ✓ | 37m54s | 13 | T/O | - | 41 |
| P14 | $EG(io \neq 1) \wedge EG(ret \neq 1)$ | T/O | - | 42 | × | 136.6s | 14 |
| P15 | $EF(io = 1) \wedge EF(ret = 1)$ | T/O | - | 15 | × | 1.4s | 43 |
| P16 | $AG(io \neq 1) \vee AG(ret \neq 1)$ | ✓ | 0.1s | 44 | × | 874.5s | 16 |
| P17 | $AG(AF(w \geq 1))$ | ✓ | 3.0s | 17 | × | 0.1s | 45 |
| P18 | $EF(EG(w < 1)$ | ✓ | 0.5s | 46 | × | 3.5s | 18 |
| P19 | $AG(EF(w \geq 1))$ | ✓ | 3.3s | 19 | × | 0.1s | 47 |
| P20 | $EF(AG(w < 1)$ | ✓ | 0.7s | 48 | × | 0.1s | 20 |
| P21 | $AG(AF(w = 1)$ | ✓ | 2.8s | 21 | × | 0.1s | 49 |
| P22 | $EF(EG(w \neq 1)$ | ✓ | 2.2s | 50 | × | 5.0s | 22 |
| P23 | $AG(EF(w = 1)$ | ✓ | 4.5s | 23 | × | 0.1s | 51 |
| P24 | $EF(AG(w \neq 1)$ | ✓ | 3.4s | 52 | × | 0.7s | 24 |
| P25 | $c > 5 \rightarrow AF(r > 5)$ | ✓ | 3.2s | 25 | × | 0.1s | 53 |
| P26 | $c > 5 \wedge EG(r \leq 5)$ | × | 0.1s | 54 | × | 1.3s | 26 |
| P27 | $c > 5 \rightarrow EF(r > 5)$ | × | 0.2s | 27 | × | 0.1s | 55 |
| P28 | $c > 5 \wedge AG(r \leq 5)$ | × | 0.1s | 56 | × | 0.3s | 28 |

Windows fragment 1 (P1–P4)

Windows fragment 2 (P5–P8)

Windows fragment 3 (P9–P12)

Windows fragment 4 (P13–P16)

Windows fragment 5 (P17–P20)

PostgreSQL pgarch (P21–P24)

Software updates (P25–P28)

# In practice (a.k.a. T/O to 0.5s)

- Templates can be used to constrain the search space for witnesses
  - for CTL verification, automatic templates can be derived
  - E-HSF uses "mark-and-resolve nondeterminism" methodology [Cook, Koskinen – PLDI'13]

- No skolemization/witnesses required for some ∀∃H-clauses

  **inv(v)** $\wedge$ ¬dst(v) $\rightarrow$ ∃v': next(v,v') use projection

- Use template structure for expensive ∀H-clauses

  **inv(v)** $\wedge$ ¬dst(v) $\wedge$ **wit(v,v')** $\rightarrow$ next(v,v') $\wedge$ **inv(v,v')** reduces to

  **inv(v)** $\wedge$ ¬dst(v) $\wedge$ **wit(v,v')** $\rightarrow$ **inv(v,v')**

- Split queries over variables with finite-domains, e.g., pc

# Related work

- Compositional proof system for CTL*
[Kesten, Pnueli, TCS'05]

- Inference of auxiliary assertions for CTL properties of programs [Cook, Koskinen – PLDI'13]
  - monotonic choice of witnesses, give up on wrong choices
  - E-HSF "backtracks" from wrong choices

- Solving Horn clauses
  - mu-Z [Hoder, Bjørner, de Moura – CAV'11]
  - HSF [Grebenschikov, Lopes, P, R – PLDI'12]

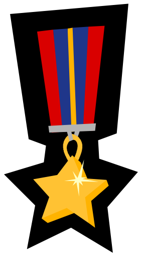
# Conclusion

- Algorithm to solve $\forall\exists$ Horn clauses
- Many applications
  - CTL properties
  - synthesis of programs from temporal specifications
  - solving games on infinite graphs with parity conditions

# Applying for jobs

- Solving recursion-free clauses over QF_LRA [POPL'11]
- Solving recursion-free clauses over QF_UFLRA [APLAS'11]
- Solving recursion-free clauses with WF [TACAS'12]

- Proof rules for multi-threaded programs [CAV'11]
- Solving recursive ∀H-clauses [PLDI'12]
- Solving recursive ∀∃H-clauses [CAV'13]
- Verification competitions [SV-COMP'12] [SV-COMP'13]



www.model.in.tum.de/~popeea

# EXTRA MATERIAL

# Steps of rec.-free solving algorithm

- Resolution
  - remove clausal structure
- Farkas' lemma
  - introduce weights for linear inequalities
- Call SMT-solve
- Obtain solution for rec.-free clauses
  - use weights and SMT solution

# Farkas' lemma

Constants:

- A – matrix
- b, 0 – vectors
- d - number

Unknowns:

- $\lambda$,t - vectors

$\neg(\exists v\text{: } Av \leq b) \wedge \forall v\text{: } Av \leq b \rightarrow 0v \leq -1$

iff

$\exists\lambda\text{: } \lambda \geq 0 \wedge \lambda A = 0 \wedge \lambda b \leq -1$

For rec.-free clauses with WF

$\exists t\text{: } (\exists v\text{: } Av \leq b \wedge \forall v\text{: } Av \leq b \rightarrow tv \leq d)$

iff

$\exists t\text{: } (\exists\lambda\text{: } \lambda \geq 0 \wedge \lambda A = t \wedge \lambda b \leq d)$

# EXAMPLE WITH CTL PROPERTY

# The behavior of software is often nondeterministic

- Interesting properties may not hold on all execution paths
  - but a property may still hold only on some path
- "For each reachable state, is that the case that on some path eventually wakend is 1?"

$\phi$ = AG (EF wakend)

- ( init, next) $\models \phi$ reduces to $\forall\exists$H-clauses

# Example PostgreSQL

```
/*
 * Main loop for archiver
 */
int wakend, last_copy_time = 0, curtime, got_SIGHUP;
#define PGC_SIGHUP 1
#define PGARCH_AUTOWAKE_INTERVAL 1000

void ProcessConfigFile(int a) { /* process the file */ }
void pgarch_ArchiverCopyLoop() { /* loop of the archiver */ }
int XLogArchivingActive() { return nondet(); }
int PostmasterIsAlive() { return nondet(); }
int time(int a) { return nondet(); }

int pgarch_MainLoop(void) {

  wakend = true;

  /*
   * There shouldn't be anything for the archiver to do except to
   * wait for a signal, ... however, the archiver exists to
   * protect our data, so she wakes up occasionally to allow
   * herself to be proactive. In particular this avoids getting
   * stuck if a signal arrives just before we sleep.
   */
while(1)
  {
    /* Check for config update */
    if (got_SIGHUP)
        {
          got_SIGHUP = false;
          ProcessConfigFile(PGC_SIGHUP);
          if (!XLogArchivingActive())
            break;              /* user wants us to shut down */
        }
    /* Do what we're here for */
    if (wakend)
        {
          wakend = false;
          pgarch_ArchiverCopyLoop();
          last_copy_time = time(NULL);
        }
    if (!wakend)
        {
          curtime = time(NULL);
          if ((curtime - last_copy_time) >= PGARCH_AUTOWAKE_INTERVAL)
            wakend = true;
        }
    if (!PostmasterIsAlive()) { break; }
  }

}
```

Are there any sources of nondeterminism in this model?

$\phi$ = AG (AF wakend)

# ALGORITHM

# E-HSF

```
algorithm E-HSF(Clauses)
1   Skolemized, Parent, Rels, Grds := SKOLEMIZE(Clauses)
2   Constraint := true
3   Defs := {true → rel(v, w) | rel ∈ Rels} ∪ {grd(v) → true | grd ∈ Grds}
4   match HSF(Skolemized ∪ Defs) with
5   | solution ClauseSol -> return "solution ClauseSol"
6   | error derivation Cex and symbol map SYM ->
7     CexDefs := {(body → q(...)) ∈ Cex | SYM(q) ∈ Rels ∪ Grds}
8     if CexDefs = ∅ then return "error derivation Cex and symbol map SYM"
9     else
10      (body ∧ ⋀_{i=1}^{n} q_i(v_i, w_i) → head) := RESOLVE(Cex \ CexDefs)
11      body := body ∧ ⋀_{i=1}^{n} RELT(SYM(q_i))(v_i, w_i)
12      match head with
13       | q(v, w) when dwf(SYM(q)) ∈ Clauses ->
14         head := BOUNDT(SYM(q))(v) ∧ DECREASET(SYM(q))(v, w)
15       | q(v) when SYM(q) ∈ Grds ->
16         head := GRDT(SYM(q))(v)
17       | _ -> skip
18      Constraint := ENCODEVALIDITY(body → head) ∧ Constraint
19      match SMTSOLVE(Constraint) with
20       | solution CexSol ->
31         Defs := {RELT(rel)(v, w)CexSol → rel(v, w) | rel ∈ Rels} ∪
32                 {grd(v) → GRDT(grd)(v)CexSol | grd ∈ Grds}
33         goto line 4
34       | _ -> return "error derivation Cex and symbol map SYM"
```

Solution for ∀∃H clauses

No solution for ∀∃H clauses