Efficient CTL Verification via Horn Constraints Solving

Tewodros A. Beyene [1]
fortiss GmbH, Munich, Germany

joint work with **C. Popeea** [2] and **A. Rybalchenko** [3]

fortiss CQSE Microsoft Research

Temporal logics

- Important class of **specification languages**
  - applications: **program verification, synthesis, security analysis**, etc.
  - when **assertions** do not suffice
- Deductive temporal reasoning:

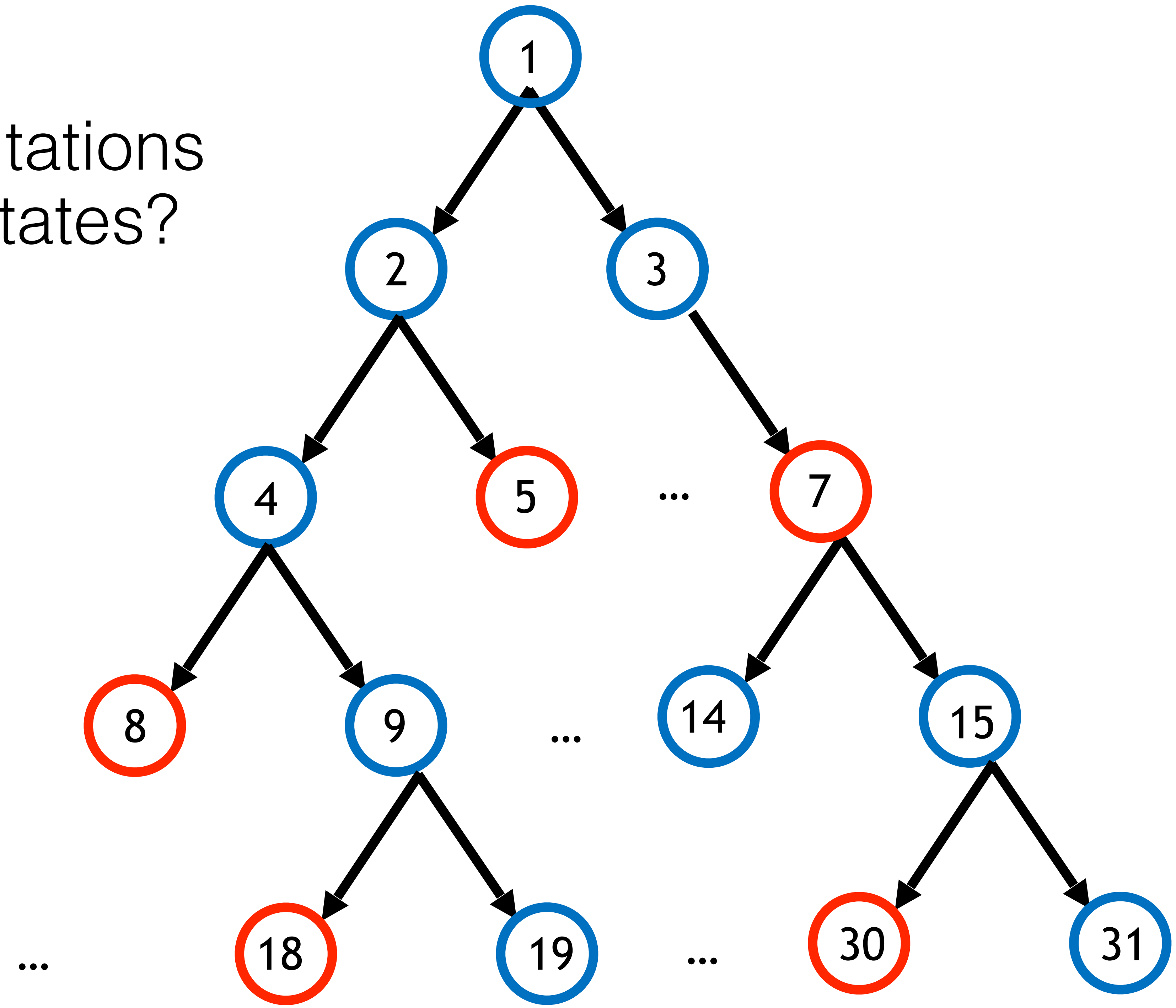
  (1) **Proof rules** to generate proof sub-goals

  (2) **Solvers** for **auxiliary predicates**
- **Universal** and **existential** fragments

Temporal universal properties

Do all computations avoid ***error*** states?

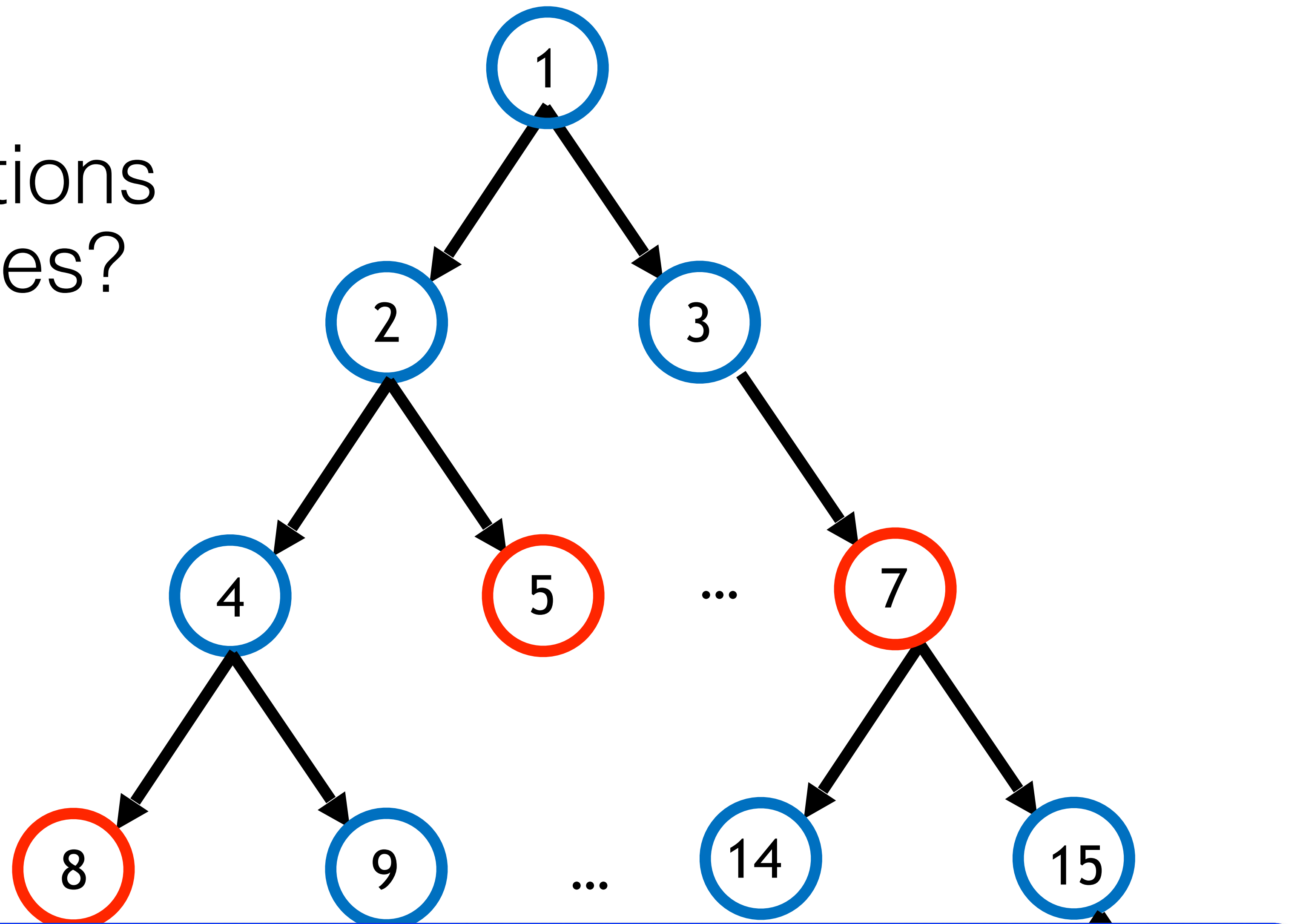
AG ¬ ○



Solving universal properties

Do all computations avoid ***error*** states?

$AG\ \neg$ ○

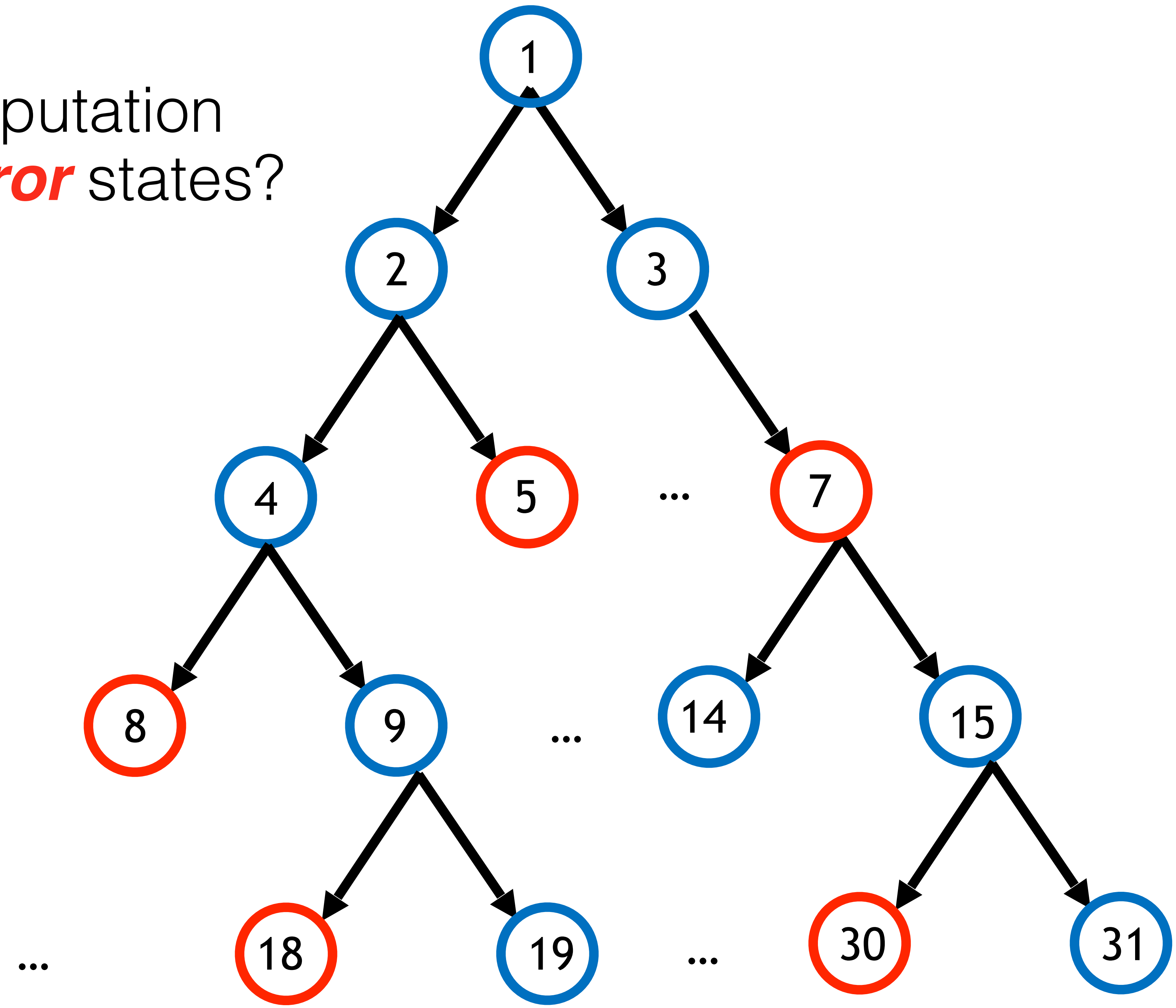


... a success story!

Temporal existential properties

Is there a computation that avoids ***error*** states?

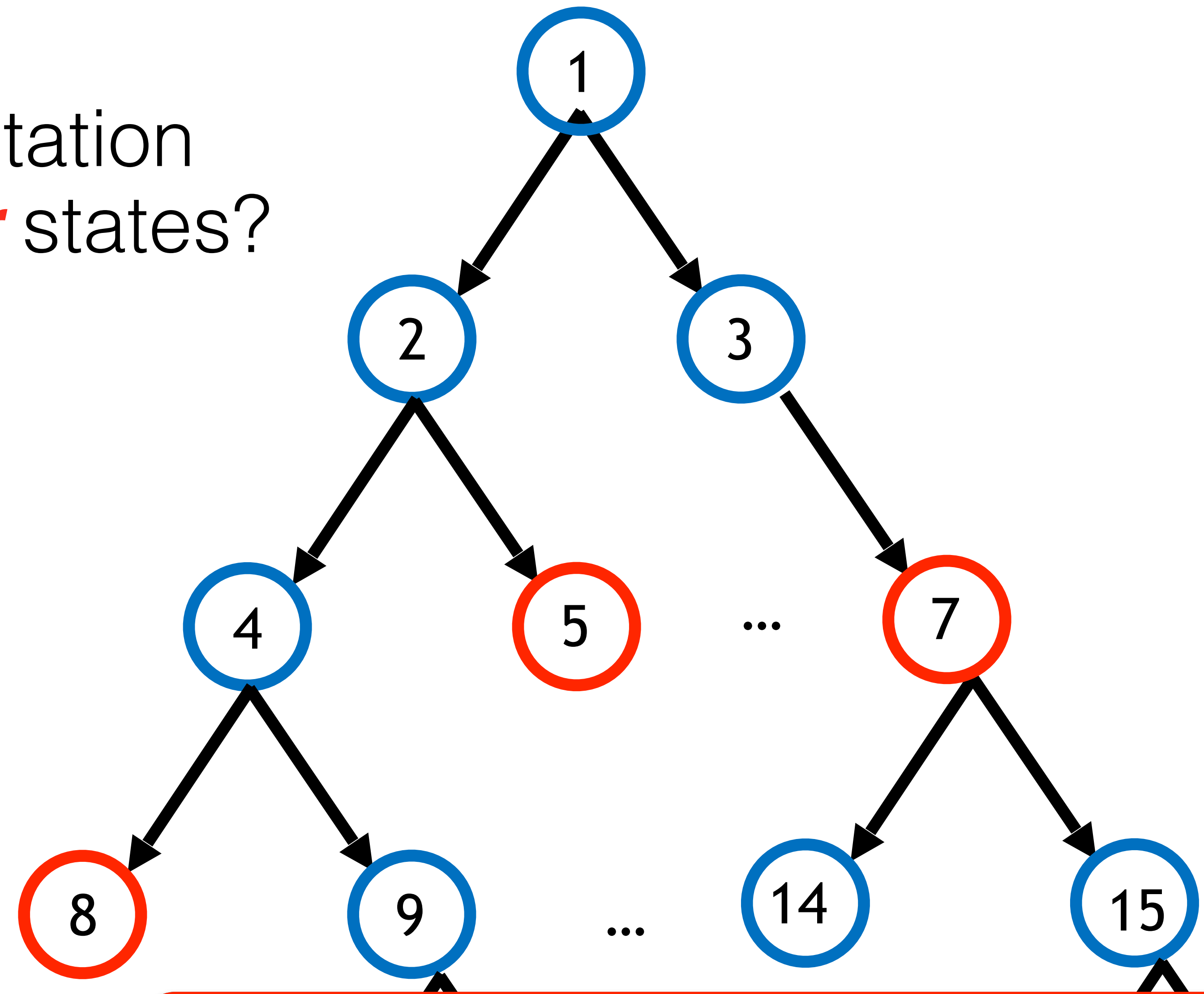
$EG \neg$ ○



Solving existential properties

Is there a computation
that avoids ***error*** states?

EG ¬ ○



... not as good as universal solvers!

State of the art

- Specialised efforts, e.g., non-termination = ***EG true***
- Direct efforts: e.g., ***Cook-Koskinen PLDI2013***, ***Cook-Khlaaf-Piterman FMCAD2014***.
    - program transformation specific to CTL

In this talk …

CTL Verification

↓

Proof rules

- *forall-exists quantified Horn constraints*
- *well-foundedness constraints*

E-HSF engine

Context

- Existentially quantified Horn constraints (CAV13)

Solving Existentially Quantified Horn Clauses

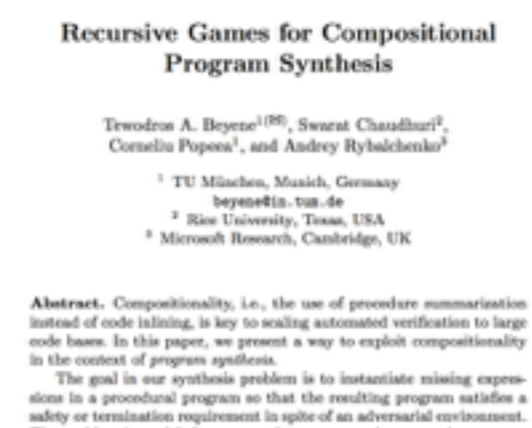
Tewodros A. Beyene[1], Corneliu Popeea[1], and Andrey Rybalchenko[1,2]

[1] Technische Universität München
[2] Microsoft Research Cambridge

- Proof rules for 2-player games and automation (POPL14, VSTTE15)

A Constraint-Based Approach to Solving Games on Infinite Graphs

Tewodros A. Beyene, Technische Universität München
Swarat Chaudhuri, Rice University
Corneliu Popeea, Technische Universität München
Andrey Rybalchenko, Microsoft Research Cambridge and Technische Universität München



Recursive Games for Compositional Program Synthesis

Tewodros A. Beyene[1], Swarat Chaudhuri[2], Corneliu Popeea[1], and Andrey Rybalchenko[3]

[1] TU München, Munich, Germany
beyene@in.tum.de
[2] Rice University, Texas, USA
[3] Microsoft Research, Cambridge, UK

- CTL+FO verification (SPIN14)

CTL+FO Verification as Constraint Solving

Tewodros A. Beyene, Technische Universität München, Germany
Marc Brockschmidt, Microsoft Research Cambridge, UK
Andrey Rybalchenko, Microsoft Research Cambridge, UK

Outline

- Illustration
- Solving engine: E-HSF
- Proof rules
- Experimental evaluation

Outline

- Illustration
- Solving engine: E-HSF
- Proof rules
- Experimental evaluation

Example

```
assume(y=0)
while(1) {
    x = x+y
    y = nondet()
  }
```

EF (x ≥ 0) ?

Example

```
assume(y=0)
while(1) {
    x = x+y
    y = nondet()
  }
```

EF (x ≥ 0) ?

v = (x, y)

init(v) = (y=0)

next(v, v') = (x'=x+y ∧ y'=?)

(init(v), next(v, v')) ⊨ EF (x ≥ 0) ?

Constraint Generation

1. $\forall v$: init(v) → inv(v)
2. $\forall v$: inv(v) ∧¬ (x ≥ 0) → ∃v':next(v, v') ∧ inv(v') ∧ round(v, v')
3. well-founded(round(v, v'))

(init(v), next(v, v')) ⊨ EF (x ≥ 0)

- solve for inv(v') and round(v, v')

Solving challenges

1. $\forall v$: init(v) → inv(v)
2. $\forall v$: inv(v) ∧¬ (x ≥ 0) → ∃v': next(v, v') ∧ inv(v') ∧ round(v, v')
3. well-founded(round(v, v'))

$$(\text{init}(v), \text{next}(v, v')) \models EF\ (x \geq 0)$$

- solve for inv(v') and round(v, v')

Solving challenges

1. $\forall v: \mathit{init}(v) \rightarrow \mathit{inv}(v)$
2. $\forall v: \mathit{inv}(v) \wedge \neg (x \geq 0) \rightarrow \exists v': \mathit{next}(v, v') \wedge$
   $\mathit{inv}(v') \wedge \mathit{round}(v, v')$
3. $\text{well-founded}(\mathit{round}(v, v'))$

$$(\mathit{init}(v), \mathit{next}(v, v')) \models EF\ (x \geq 0)$$

- solve for inv(v') and round(v, v')

Solving challenges

1. $\forall v\colon \mathit{init}(v) \rightarrow \mathit{inv}(v)$
2. $\forall v\colon \mathit{inv}(v) \land \neg (x \geq 0) \rightarrow \exists v'\colon \mathit{next}(v, v') \land$

   $\mathit{inv}(v') \land \mathit{round}(v, v')$
3. $\text{well-founded}(\mathit{round}(v, v'))$

$$(\mathit{init}(v), \mathit{next}(v, v')) \models EF\ (x \geq 0)$$

- solve for $\mathit{inv}(v')$ and $\mathit{round}(v, v')$

Example: one solution

```
assume(y=0)
while(1) {
      x = x+y
      y = nondet()
   }
```
EF $(x \geq 0)$?

$v = (x, y)$

$init(v) = (y=0)$

$next(v, v') = (x'=x+y \wedge y'=?)$

$(init(v), next(v, v')) \models EF\ (x \geq 0)$?

1. $\forall v: init(v) \rightarrow inv(v)$
2. $\forall v: inv(v) \wedge \neg (x \geq 0) \rightarrow \exists v':next(v, v') \wedge inv(v') \wedge round(v, v')$
3. well-founded$(round(v, v'))$

$$sol(inv(v)) := (x \geq 0 \vee x<0 \wedge y=0 \vee x<0 \wedge \mathbf{y>0})$$

$$sol(round(v, v')) := (x<0 \wedge y=0 \wedge x'=x \wedge \boldsymbol{y'>0} \vee$$
$$x<0 \wedge y>0 \wedge x'=x+y \wedge \boldsymbol{y'>0})$$

Outline

- Illustration
- Solving engine: E-HSF
- Proof rules
- Experimental evaluation

E-HSF: inputs and outputs

- Inputs: **Horn constraints** and **Skolem Templates**
  - e.g., **{** … → aux(x),

    aux(x) → ∃y: … ∧ aux(y)

    }
- Output: model for each **auxiliary predicate**
  - instantiations for **existentially** quantified variables

E-HSF: skolemization

- Reformulates problem as witness finding
- Given **$\forall v$: body(v) → $\exists w$: head(v, w)**
    1. **$\forall v,w$: body(v) ∧ wit(v, w) → head(v, w)**
    2. **body(v)** → domain(**wit(v, w)**)
- (2) ***ensures*** that for any state in **body(v)**, a witness is defined by **wit(v, w)**.
- **Skolem Templates** define space of witness relations
- applies **HSF**

E-HSF: basic features

- A **CEGAR**-like scheme
- Refines **skolem template** instantiation
  - unlike in **CEGAR**, no monotonicity!
- A **global constraint** ensures progress of template refinement
  - by keeping track of **counter-examples** from previous template instantiations

Outline

- Illustration
- Solving engine: E-HSF
- Proof rules
- Experimental evaluation

Proof rules

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

- 2 set of proof rules

    - **Constraint generation**: for basic CTL state formulas.

    - **Decomposition**: for non-basic CTL state formulas.

Proof rule RuleCtlEG

Find an assertion $inv(v)$ such that:

$$\frac{\begin{array}{c} p(v) \rightarrow inv(v) \\ inv(v) \rightarrow \exists v' : next(v,v') \wedge inv(v') \\ inv(v) \rightarrow q(v) \end{array}}{(p(v), next(v,v')) \models_{CTL} EG\ q(v)}$$

- for basic CTL formula with EG outer operator

Proof rule RuleCtlDecompBin

Given a CTL formula $f(\psi_1(v),\ \psi_2(v))$ where $f \in \{AU, EU, \wedge, \vee\}$, and a transition system $(p(v), next(v, v'))$, find assertions $q_1(v)$ and $q_2(v)$ such that:

$$\frac{\begin{array}{c} p(v) \rightarrow f(q_1(v), q_2(v)), \\ (q_1(v), next(v, v')) \models_{CTL} \psi_1(v) \quad (q_2(v), next(v, v')) \models_{CTL} \psi_2(v) \end{array}}{(p(v), next(v, v')) \models_{CTL} f(\psi_1(v), \psi_2(v))}$$

- for non-basic CTL state formulas with binary outer operator

Proof rules

1. RuleCtlDecompUni
2. RuleCtlDecompBin
3. RuleCtlEX
4. RuleCtlEG
5. RuleCtlEU
6. RuleCtlAX
7. RuleCtlAG
8. RuleCtlAU
9. RuleCtlEF
10. RuleCtlAF

Outline

- Illustration
- Solving engine: E-HSF
- Proof rules
- Experimental evaluation

Experiment setup

- **Challenging benchmarks**: *Windows OS fragment and PostgreSQL pgarch* [***Cook-Koskinen PLDI 13, Cook-Khlaff-Piterman FMCAD 14***]

- Linear **Skolem Templates** were sufficient

Evaluation

Program P	Property φ	$P \models_{CTL} \varphi$		$P \models_{CTL} \neg\varphi$	
		E-HSF	Cook [37]	E-HSF	Cook [37]
Windows OS fragment 1 (29 LOC)	$AG(p \to AFq)$	0.3	1.0	0.3	1.4
	$EF(p \wedge EGq)$	0.3	0.1	0.3	0.7
	$AG(p \to EFq)$	0.3	0.1	0.3	0.1
	$EF(p \wedge AGq)$	0.3	0.1	0.3	0.1
Windows OS fragment 2 (58 LOC)	$EF(p \wedge EGq)$	0.4	1.0	0.3	1.2
	$EF(p \wedge AGq)$	0.4	0.8	0.3	0.2
Windows OS fragment 3 (370 LOC)	$AG(p \to AFq)$	0.6	5.9	1.2	6.2
	$EF(p \wedge EGq)$	9.4	2.3	0.5	6.0
	$AG(p \to EFq)$	0.7	6.8	0.8	3.4
	$EF(p \wedge AGq)$	0.9	4.7	1.1	3.1
Windows OS fragment 4 (380 LOC)	$AFp \vee AFq$	5.7	18.5	5.2	13.9
	$EGp \wedge EGq$	0.3	13.5	1.0	14.2
	$EFp \wedge EFq$	5.0	14.7	0.3	4.8
	$AGp \vee AGq$	0.3	8.0	6.4	3.7
Windows OS fragment 5 (43 LOC)	$AG(AFp)$	0.3	1.0	0.3	0.2
	$EF(EGp)$	0.3	0.1	0.3	0.0
	$AG(EFp)$	0.3	1.0	0.3	0.0
	$EF(AGp)$	0.3	0.1	0.3	0.1
PostgreSQL pgarch (70 LOC)	$AG(AFp)$	0.4	2.0	0.3	1.3
	$EF(EGp)$	0.3	0.1	0.4	0.1
	$AG(EFp)$	0.3	2.0	0.3	0.0
	$EF(AGp)$	0.3	2.0	0.3	2.4

Evaluation

Program P	Property φ	$P \models_{CTL} \varphi$		$P \models_{CTL} \neg\varphi$	
		E-HSF	Cook [37]	E-HSF	Cook [37]
Windows OS fragment 1 (29 LOC)	$AG(p \rightarrow AFq)$	0.3	1.0	0.3	1.4
	$EF(p \wedge EGq)$	0.3	0.1	0.3	0.7
	$AG(p \rightarrow EFq)$	0.3	0.1	0.3	0.1
	$EF(p \wedge AGq)$	0.3	0.1	0.3	0.1
Windows OS fragment 2 (58 LOC)	$EF(p \wedge EGq)$	0.4	1.0	0.3	1.2
	$EF(p \wedge AGq)$	0.4	0.8	0.3	0.2
Windows OS fragment 3 (370 LOC)	$AG(p \rightarrow AFq)$	0.6	5.9	1.2	6.2
	$EF(p \wedge EGq)$	9.4	2.3	0.5	6.0
	$AG(p \rightarrow EFq)$	0.7	6.8	0.8	3.4
	$EF(p \wedge AGq)$	0.9	4.7	1.1	3.1
Windows OS fragment 4 (380 LOC)	$AFp \vee AFq$	5.7	18.5	5.2	13.9
	$EGp \wedge EGq$	0.3	13.5	1.0	14.2
	$EFp \wedge EFq$	5.0	14.7	0.3	4.8
	$AGp \vee AGq$	0.3	8.0	6.4	3.7
Windows OS fragment 5 (43 LOC)	$AG(AFp)$	0.3	1.0	0.3	0.2
	$EF(EGp)$	0.3	0.1	0.3	0.0
	$AG(EFp)$	0.3	1.0	0.3	0.0
	$EF(AGp)$	0.3	0.1	0.3	0.1
PostgreSQL pgarch (70 LOC)	$AG(AFp)$	0.4	2.0	0.3	1.3
	$EF(EGp)$	0.3	0.1	0.4	0.1
	$AG(EFp)$	0.3	2.0	0.3	0.0
	$EF(AGp)$	0.3	2.0	0.3	2.4

Evaluation

Program P	Property φ	$P \models_{CTL} \varphi$		$P \models_{CTL} \neg\varphi$	
		E-HSF	Cook [37]	E-HSF	Cook [37]
Windows OS fragment 1 (29 LOC)	$AG(p \to AFq)$	0.3	1.0	0.3	1.4
	$EF(p \wedge EGq)$	0.3	0.1	0.3	0.7
	$AG(p \to EFq)$	0.3	0.1	0.3	0.1
	$EF(p \wedge AGq)$	0.3	0.1	0.3	0.1
Windows OS fragment 2 (58 LOC)	$EF(p \wedge EGq)$	0.4	1.0	0.3	1.2
	$EF(p \wedge AGq)$	0.4	0.8	0.3	0.2
Windows OS fragment 3 (370 LOC)	$AG(p \to AFq)$	0.6	5.9	1.2	6.2
	$EF(p \wedge EGq)$	9.4	2.3	0.5	6.0
	$AG(p \to EFq)$	0.7	6.8	0.8	3.4
	$EF(p \wedge AGq)$	0.9	4.7	1.1	3.1
Windows OS fragment 4 (380 LOC)	$AFp \vee AFq$	5.7	18.5	5.2	13.9
	$EGp \wedge EGq$	0.3	13.5	1.0	14.2
	$EFp \wedge EFq$	5.0	14.7	0.3	4.8
	$AGp \vee AGq$	0.3	8.0	6.4	3.7
Windows OS fragment 5 (43 LOC)	$AG(AFp)$	0.3	1.0	0.3	0.2
	$EF(EGp)$	0.3	0.1	0.3	0.0
	$AG(EFp)$	0.3	1.0	0.3	0.0
	$EF(AGp)$	0.3	0.1	0.3	0.1
PostgreSQL pgarch (70 LOC)	$AG(AFp)$	0.4	2.0	0.3	1.3
	$EF(EGp)$	0.3	0.1	0.4	0.1
	$AG(EFp)$	0.3	2.0	0.3	0.0
	$EF(AGp)$	0.3	2.0	0.3	2.4
		26.4	**85.0**	**20.8**	**62.6**

Evaluation

Program P	Property φ	$P \models_{CTL} \varphi$		$P \models_{CTL} \neg\varphi$	
		E-HSF	Cook [37]	E-HSF	Cook [37]
Windows OS fragment 1 (29 LOC)	$AG(p \rightarrow AFq)$	0.3	1.0	0.3	1.4
	$EF(p \wedge EGq)$	0.3	0.1	0.3	0.7
	$AG(p \rightarrow EFq)$	0.3	0.1	0.3	0.1
	$EF(p \wedge AGq)$	0.3	0.1	0.3	0.1
Windows OS fragment 2 (58 LOC)	$EF(p \wedge EGq)$	0.4	1.0	0.3	1.2
	$EF(p \wedge AGq)$	0.4	0.8	0.3	0.2
Windows OS fragment 3 (370 LOC)	$AG(p \rightarrow AFq)$	0.6	5.9	1.2	6.2
	$EF(p \wedge EGq)$	9.4	2.3	0.5	6.0
	$AG(p \rightarrow EFq)$	0.7	6.8	0.8	3.4
	$EF(p \wedge AGq)$	0.9	4.7	1.1	3.1
Windows OS fragment 4 (380 LOC)	$AFp \vee AFq$	5.7	18.5	5.2	13.9
	$EGp \wedge EGq$	0.3	13.5	1.0	14.2
	$EFp \wedge EFq$	5.0	14.7	0.3	4.8
	$AGp \vee AGq$	0.3	8.0	6.4	3.7
Windows OS fragment 5 (43 LOC)	$AG(AFp)$	0.3	1.0	0.3	0.2
	$EF(EGp)$	0.3	0.1	0.3	0.0
	$AG(EFp)$	0.3	1.0	0.3	0.0
	$EF(AGp)$	0.3	0.1	0.3	0.1
PostgreSQL pgarch (70 LOC)	$AG(AFp)$	0.4	2.0	0.3	1.3
	$EF(EGp)$	0.3	0.1	0.4	0.1
	$AG(EFp)$	0.3	2.0	0.3	0.0
	$EF(AGp)$	0.3	2.0	0.3	2.4
		26.4	**85.0**	**20.8**	**62.6**

↓ 70% time

Summary

- contrasting success of temporal reasoning for existential and universal fragments
- **Horn-constraint** based solution: **CTL proof rules** and **E-HSF engine**
- experimental evaluation on a set of challenging benchmarks
- generic approach performs better!