Software Verification via Fixed-Point Logics

Hiroshi Unno (hiroshi.unno@acm.org)

Research Institute of Electrical Communication,
Tohoku University, Japan

The Societal Impact of Software Failures

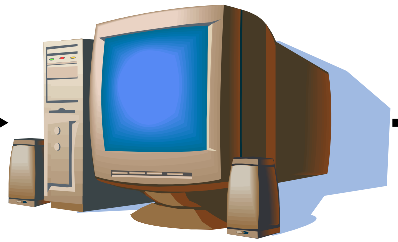
- Our society heavily relies on computer systems
- Failure or malfunction of safety-critical systems would lead to human, social, economic, and environmental damage
  - 1985-1987 – Therac-25 medical accelerator delivered lethal radiation doses to patients
  - June 4, 1996 – Ariane 5 Flight 501 exploded
  - February, 2014 – 1.9 million Prius cars recalled
  - April, 2014 – OpenSSL Heartbleed vulnerability disclosed
  - June 17, 2016 – Ethereum DAO attacked, over $55M stolen
- ***Reliability assurance of safety-critical systems is crucial***

Software Verification

- Formally prove or disprove a mathematical proposition Q:
  "The given program satisfies its formal specification"
- Great attentions from industry and academia
  - Microsoft's SLAM & Everest projects, Facebook's Infer, AWS
  - Turing awards to Hoare logic, temporal logic, model checking, …

Program & Spec.

```
let rec mc x =
  if x > 100 then
    x - 10
  else
    mc (mc (x + 11))
in
let n = randi() in
if n ≤ 101 then
  assert (mc n = 91)
```



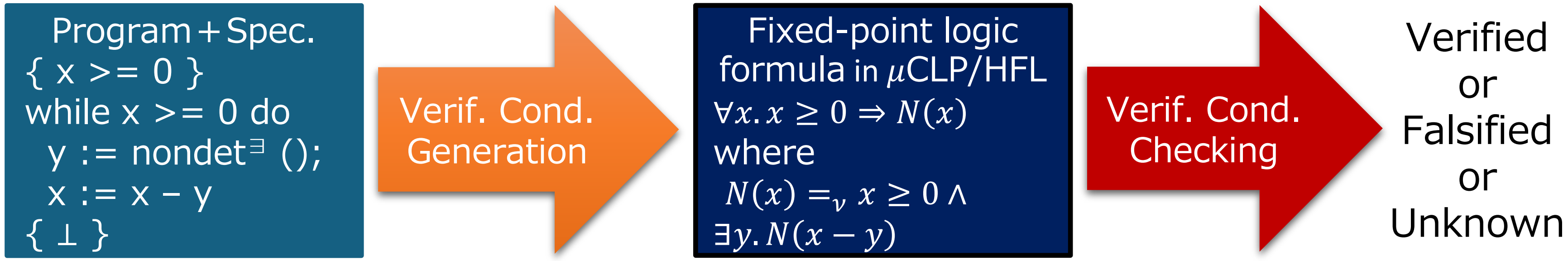
Software Verifier

A proof of Q
or
A counterexample to Q
(e.g., a program input
leading to a spec. violation)
or
Unknown

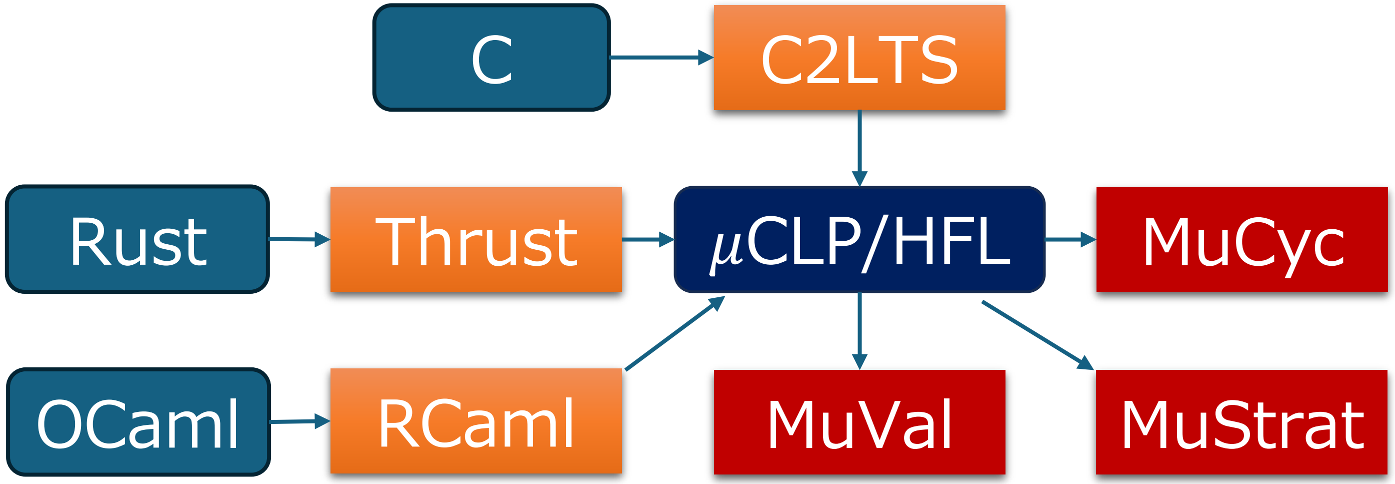
Enabling Technologies

- ***Program logics & type systems*** for formal reasoning
  - Hoare logic, Separation logic, …
  - Dependent refinement type systems, …
- ***Theorem provers & constraint solvers*** for automated reasoning
  - SAT solvers: satisfiability checkers for propositional formulas
  - SMT solvers: satisfiability checkers for predicate formulas over first-order theories on ***integers, reals, lists, arrays***, …
  - **Predicate constraint solvers**: satisfiability checkers for logical constraints on ***predicate variables*** that represent ***inductive invariants, well-founded relations (or ranking functions), Skolem functions (or recurrent sets), …***
  - **Fixed-point logic solvers**: validity checkers for fixed-point logic formulas

This Course



- Introduction to ***software verification*** based on ***fixed-point logics***
  - How to reduce software verification to validity checking for fixed-point logics?
  - How to check validity? Three complementary approaches:
    1. Reduction to the ***constraint-solving problem*** over predicate variables (**MuVal**)
    2. Reduction to the ***proof search problem*** in a cyclic proof system (**MuCyc**)
    3. Reduction to the ***game-solving problem*** induced by fixed-point logic formulas (**MuStrat**)



CoAR: Verification tools based on fixed-point logics (https://github.com/hiroshi-unno/coar)

Course Schedule

- Wed. 21 May (8:50-10:30)
  1. Reduction from software verification to fixed-point logic validity checking
  2. Predicate constraint solving for validity checking
- Thu. 22 May (11:20-12:20)
  3. Cyclic-proof search for validity checking
  4. Game solving for validity checking

1. Reduction from Software Verification to Validity Checking for Fixed-Point Logics

Outline

- Reduction from imperative programs to fixed-point logics (Mu-Arithmetic [CSL 1999] and μCLP [POPL 2023])
  - Safety verification
  - Termination verification
  - Non-termination verification
  - Modal μ-calculus model checking [SAS 2019]
- Reduction from higher-order probabilistic programs to a higher-order and quantitative fixed-point logic
  - Upper bounds verification of weakest pre-expectation [ICFP 2024]

[CSL 1999] Bradfield. Fixpoint Alternation and the Game Quantifier.
[SAS 2019] Kobayashi et al. Temporal Verification of Programs via First-Order Fixpoint Logic.
[POPL 2023] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.

Outline

- **Reduction from imperative programs to fixed-point logics (Mu-Arithmetic [CSL 1999] and μCLP [POPL 2023])**
  - Safety verification
  - Termination verification
  - Non-termination verification
  - Modal μ-calculus model checking [SAS 2019]
- Reduction from higher-order probabilistic programs to a higher-order and quantitative fixed-point logic
  - Upper bounds verification of weakest pre-expectation [ICFP 2024]

[CSL 1999] Bradfield. Fixpoint Alternation and the Game Quantifier.
[SAS 2019] Kobayashi et al. Temporal Verification of Programs via First-Order Fixpoint Logic.
[POPL 2023] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.

Mu-Arithmetic: A First-Order *Fixed-Point Logic* with Integer Arithmetic [CSL 1999]

predicate symbols: $>$ and $=$

(*formulas*) $\phi ::= \bot \mid \top \mid \neg\phi \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \forall x.\phi \mid \exists x.\phi \mid P(\vec{t}) \mid p(\vec{t})$

(*predicates*) $P ::= X \mid \boldsymbol{\mu X.\lambda\vec{x}.\phi} \mid \boldsymbol{\nu X.\lambda\vec{x}.\phi}$ (*terms*) $t ::= x \mid f(\vec{t})$

predicate variables

Least fixed-point (X occurs only positively in ϕ)

Greatest fixed-point (X occurs only positively in ϕ)

term variables

constant and function symbols: $n \in \mathbb{Z}$, $+$, and $\times$

- We assume that formulas, predicates, and terms are well-sorted
- Least fixpoints $\boldsymbol{\mu X.\lambda\vec{x}.\phi}$ represent *inductive predicates*, and greatest fixpoints $\boldsymbol{\nu X.\lambda\vec{x}.\phi}$ represent *co-inductive predicates*
- We also use (hierarchical) equational form: $X(\vec{x}) =_\mu \phi$ and $X(\vec{x}) =_\nu \phi$

$\boldsymbol{\mu}$CLP: A First-Order *Fixed-Point Logic* Modulo Background Theories T [POPL 2023]

predicate symbols of T

(formulas) $\phi ::= \bot \mid \top \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall x. \phi \mid \exists x. \phi \mid P(\vec{t}) \mid p(\vec{t})$

(predicates) $P ::= X \mid \boldsymbol{\mu X. \lambda\vec{x}. \phi} \mid \boldsymbol{\nu X. \lambda\vec{x}. \phi}$ *(terms)* $t ::= x \mid f(\vec{t})$

predicate variables

Least fixed-point (X occurs only positively in ϕ)

Greatest fixed-point (X occurs only positively in ϕ)

term variables

constant and function symbols of T

- We assume that formulas, predicates, and terms are well-sorted
- Least fixpoints $\boldsymbol{\mu X. \lambda\vec{x}. \phi}$ represent *inductive predicates*, and greatest fixpoints $\boldsymbol{\nu X. \lambda\vec{x}. \phi}$ represent *co-inductive predicates*
- We also use (hierarchical) equational form: $X(\vec{x}) =_\mu \phi$ and $X(\vec{x}) =_\nu \phi$

Example Formulas of Mu-Arithmetic

$$\big(\mu X.\lambda x.x = 0 \vee X(x-1)\big)(n)$$
$$\Leftrightarrow \Big(\lambda x.x = 0 \vee \big(\mu X.\lambda x.x = 0 \vee X(x-1)\big)(x-1)\Big)(n)$$
$$\Leftrightarrow n = 0 \vee \big(\mu X.\lambda x.x = 0 \vee X(x-1)\big)(n-1)$$
$$\Leftrightarrow n = 0 \vee n-1 = 0 \vee \big(\mu X.\lambda x.x = 0 \vee X(x-1)\big)(n-2)$$
$$\Leftrightarrow n = 0 \vee n = 1 \vee \big(\mu X.\lambda x.x = 0 \vee X(x-1)\big)(n-2)$$
$$\Leftrightarrow n = 0 \vee n = 1 \vee n = 2 \vee \cdots \vee \big(\mu X.\lambda x.x = 0 \vee X(x-1)\big)(k)$$
$$\Leftrightarrow \exists z \geq 0.n = z \Leftrightarrow n \geq 0$$

$$\big(\nu X.\lambda x.x \geq 0 \wedge X(x+1)\big)(n)$$
$$\Leftrightarrow n \geq 0 \wedge \big(\nu X.\lambda x.x \geq 0 \wedge X(x+1)\big)(n+1)$$
$$\Leftrightarrow n \geq 0 \wedge n+1 \geq 0 \wedge \cdots \wedge \big(\nu X.\lambda x.x \geq 0 \wedge X(x+1)\big)(k)$$
$$\Leftrightarrow \forall z \geq 0.n+z \geq 0 \Leftrightarrow n \geq 0$$

Example Formulas of Mu-Arithmetic

$$(\nu X.\lambda x.x = 0 \vee X(x-1))(n)$$
$$\Leftrightarrow n = 0 \vee (\nu X.\lambda x.x = 0 \vee X(x-1))(n-1)$$
$$\Leftrightarrow n = 0 \vee n = 1 \vee \cdots \vee (\nu X.\lambda x.x = 0 \vee X(x-1))(k)$$
$$\Leftrightarrow n = 0 \vee n = 1 \vee \cdots \vee \top \Leftrightarrow \top$$

$$(\mu X.\lambda x.x \geq 0 \wedge X(x+1))(n)$$
$$\Leftrightarrow n \geq 0 \wedge (\mu X.\lambda x.x \geq 0 \wedge X(x+1))(n+1)$$
$$\Leftrightarrow n \geq 0 \wedge n+1 \geq 0 \wedge \cdots \wedge (\mu X.\lambda x.x \geq 0 \wedge X(x+1))(k)$$
$$\Leftrightarrow n \geq 0 \wedge n+1 \geq 0 \wedge \cdots \wedge \bot \Leftrightarrow \bot$$

Example: Fixpoint Alternation

```
let rec f y = if y = 0 then g 10 else f (y - 1)
and g x = f x
```

- Q1. During the evaluation of `g 1`, does the call to `f` eventually invoke `g` recursively, and does the call to `g` repeat this infinitely often?

$$\left(\nu G.\lambda x.\left(\mu F.\lambda y.(y = 0 \wedge G\ 10) \vee \left(y \neq 0 \wedge F\ (y-1)\right)\right)(x)\right)(1)$$

$G\ 1$ where
$G\ x =_\nu F\ x$
$F\ y =_\mu (y = 0 \wedge G\ 10) \vee \left(y \neq 0 \wedge F\ (y-1)\right)$

- Q2. During the evaluation of `f 1`, is `f` not recursively called infinitely?

$$\left(\mu F.\lambda y.\left(y = 0 \wedge (\nu G.\lambda x.F\ x)(10)\right) \vee \left(y \neq 0 \wedge F\ (y-1)\right)\right)(1)$$

$\Leftrightarrow F\ 10$

$F\ 1$ where
$F\ y =_\mu (y = 0 \wedge G\ 10) \vee \left(y \neq 0 \wedge F\ (y-1)\right)$
$G\ x =_\nu F\ x$

Example: Fixpoint Alternation (cont.)

- Q1. During the evaluation of `g 1`, does the call to `f` eventually invoke g recursively, and does the call to g repeat this infinitely often?

$$\Big(\nu G.\lambda x.\Big(\mu F.\lambda y.(y = 0 \land G\ 10) \lor \big(y \neq 0 \land F\ (y-1)\big)\Big)(x)\Big)(1)$$

$$\Leftrightarrow \Big(\mu F.\lambda y.\big(y = 0 \land \big(\nu G.\lambda x.(\mu F.\cdots)\big)\ 10\big) \lor \big(y \neq 0 \land F\ (y-1)\big)\Big)(1)$$

$$\Leftrightarrow \Big(\mu F.\lambda y.\big(y = 0 \land \big(\nu G.\lambda x.(\mu F.\cdots)\big)\ 10\big) \lor \big(y \neq 0 \land F\ (y-1)\big)\Big)(0)$$

$$\Leftrightarrow \big(\nu G.\lambda x.(\mu F.\cdots)\big)\ 10 \Leftrightarrow (\mu F.\cdots)\ 10 \Leftrightarrow (\mu F.\cdots)\ 0$$

This is not used!

$$\Leftrightarrow \big(\nu G.\lambda x.(\mu F.\cdots)\big)\ 10 \Leftrightarrow \top$$

This is called indefinitely!

Example: Fixpoint Alternation (cont.)

- Q2. During the evaluation of `f 1`, is `f` not recursively called infinitely?

$$\Big(\mu F.\lambda y.\big(y = 0 \wedge (\nu G.\lambda x.F\ x)(10)\big) \vee \big(y \neq 0 \wedge F\ (y-1)\big)\Big)(1)$$

$$\Leftrightarrow \Big(\mu F.\lambda y.(y = 0 \wedge (\nu G.\lambda x.F\ x)\ 10) \vee \big(y \neq 0 \wedge F\ (y-1)\big)\Big)(0)$$

$$\Leftrightarrow \big(\nu G.\lambda x.(\mu F.\cdots)(x)\big)\ 10 \Leftrightarrow (\mu F.\cdots)\ 10 \Leftrightarrow (\mu F.\cdots)\ 0$$

$$\Leftrightarrow \big(\nu G.\lambda x.(\mu F.\cdots)(x)\big)\ 10 \Leftrightarrow \bot$$

Both are called indefinitely!

Example: *Partial Correctness* Verification

Pre-condition

$\{x = x_0\}$
$y = 0;$
while $x \neq 0$ **do**
$\quad y \leftarrow y + 1;$
$\quad x \leftarrow x - 1$
done
$\{y = x_0\}$

Post-condition

If the initial state satisfies the pre-condition $x = x_0$ and *the loop terminates*

the post-condition $y = x_0$ is satisfied by the resulting state

Verification Condition in Mu-Arithmetic:

$$\forall x_0, x, y. (R(x_0, x, y) \wedge x = 0 \Rightarrow y = x_0) \text{ where}$$

$$R(x_0, x, y) =_\mu (x = x_0 \wedge y = 0) \vee \exists x', y'. \begin{pmatrix} R(x_0, x', y') \wedge x' \neq 0 \wedge \\ x = x' - 1 \wedge y = y' + 1 \end{pmatrix}$$

Example: *Partial Correctness* Verification

Pre-condition

$\{x = x_0\}$
$y = 0;$
$\textbf{while } x \neq 0 \textbf{ do}$
$\quad y \leftarrow y + 1;$
$\quad x \leftarrow x - 1$
$\textbf{done}$
$\{y = x_0\}$

Post-condition

If the initial state satisfies the pre-condition $x = x_0$ and *the loop terminates*

the post-condition $y = x_0$ is satisfied by the resulting state

Verification Condition in Mu-Arithmetic:

$$\forall x_0, x, y.\, (R(x_0, x, y) \wedge x = 0 \Rightarrow y = x_0) \text{ where}$$
$$R(x_0, x, y) =_\mu (x = x_0 \wedge y = 0) \vee (R(x_0, x + 1, y - 1) \wedge x + 1 \neq 0)$$

Example: *Partial Correctness* Verification

Pre-condition

Post-condition

$\{x = x_0\}$
$y = 0;$
while $x \neq 0$ **do**
    $y \leftarrow y + 1;$
    $x \leftarrow x - 1$
done
$\{y = x_0\}$

If the initial state satisfies the pre-condition $x = x_0$ and *the loop terminates*

the post-condition $y = x_0$ is satisfied by the resulting state

Verification Condition in Mu-Arithmetic:

$$\forall x_0, x, y. \left((x = x_0 \wedge y = 0) \Rightarrow S(x_0, x, y) \right) \text{ where}$$
$$S(x_0, x, y) =_\nu (x = 0 \Rightarrow y = x_0) \wedge \left(x \neq 0 \Rightarrow S(x_0, x - 1, y + 1) \right)$$

Example: *Total Correctness* Verification

Pre-condition

$[x \neq 0]$

while $x \neq 0$ **do**

  if $x > 0$ **then**

    $x \leftarrow x - 1$

  else

    $x \leftarrow x + 1$

done

$[\top]$

Post-condition

Tautology

If the initial state satisfies the pre-condition $x \neq 0$

the loop always terminates and the post-condition $\top$ is satisfied by the resulting state

Verification Condition in Mu-Arithmetic:

$$\forall x. \big(x \neq 0 \Rightarrow S(x)\big) \text{ where}$$

$$S(x) =_{\mu} (x = 0 \Rightarrow \top) \wedge \big(x > 0 \Rightarrow S(x-1)\big) \wedge \big(x < 0 \Rightarrow S(x+1)\big)$$

Example: *Total Correctness* Verification

Pre-condition

$[x \neq 0]$

while $x \neq 0$ **do**

    if $x > 0$ **then**

        $x \leftarrow x - 1$

    else

        $x \leftarrow x + 1$

done

$[\top]$

Post-condition

Tautology

If the initial state satisfies the pre-condition $x \neq 0$

the loop always terminates and the post-condition $\top$ is satisfied by the resulting state

Verification Condition in Mu-Arithmetic:

$\forall x. \left(x \neq 0 \Rightarrow S(x)\right)$ where

$S(x) =_\mu \left(x > 0 \Rightarrow S(x-1)\right) \wedge \left(x < 0 \Rightarrow S(x+1)\right)$

Example: *Partial Correctness* Verification with *Finitely-Branching Angelic Non-Determinism*

Pre-condition

$\{x > 0\}$

$\textbf{while } x \neq 0 \textbf{ do}$

Non-det. branching

$\quad \textbf{if read\_bool}()^{\exists} \textbf{ then}$

$\qquad x \leftarrow x - 1$

$\quad \textbf{else}$

$\qquad x \leftarrow x + 1$

$\textbf{done}$

Post-condition

$\{\bot\}$

Contradiction

If the initial state satisfies the pre-condition $x > 0$

***there is* an execution of the program such that**

the post-condition $\bot$ is satisfied when the while loop terminates

Verification Condition in Mu-Arithmetic:

Example: *Partial Correctness* Verification with *Finitely-Branching Angelic Non-Determinism*

Pre-condition $\{x > 0\}$

If the initial state satisfies the pre-condition $x > 0$

while $x \neq 0$ **do**

Non-det. branching

if **read_bool( )**$^{\exists}$ **then**

$x \leftarrow x - 1$

else

$x \leftarrow x + 1$

done

there is an execution of the program such that

the while loop *never* terminates

Post-condition $\{\perp\}$ Contradiction

Verification Condition in Mu-Arithmetic:

$$\forall x. \left(x > 0 \Rightarrow S(x)\right) \text{ where}$$

$$S(x) =_{\nu} (x = 0 \Rightarrow \perp) \wedge \left(x \neq 0 \Rightarrow \left(S(x-1) \vee S(x+1)\right)\right)$$

Example: *Partial Correctness* Verification with *Finitely-Branching Angelic Non-Determinism*

Pre-condition $\{x > 0\}$

$\textbf{while } x \neq 0 \textbf{ do}$

Non-det. branching $\quad \textbf{if } \mathbf{read\_bool}(\,)^{\exists} \textbf{ then}$

$\quad\quad x \leftarrow x - 1$

$\quad \textbf{else}$

$\quad\quad x \leftarrow x + 1$

$\textbf{done}$

Post-condition $\{\bot\}$ **Contradiction**

If the initial state satisfies the pre-condition $x > 0$

***there is* an execution of the program such that**

the while loop *never* terminates

Verification Condition in Mu-Arithmetic:

$$\forall x.\left(x > 0 \Rightarrow S(x)\right) \text{ where}$$

$$S(x) =_{\nu} x \neq 0 \wedge \left(S(x-1) \vee S(x+1)\right)$$

Example: *Partial Correctness* Verification with *Infinitely-Branching Angelic Non-Determinism*

Pre-condition $\{x > 0\}$

$$\mathbf{while}\ x > 0\ \mathbf{do}$$
$$\quad x \leftarrow \mathbf{read\_int}(\,)^{\exists} - x$$
$$\mathbf{done}$$

Non-det. integer

Post-condition $\{\bot\}$ **Contradiction**

If the initial state satisfies the pre-condition $x > 0$

there is **an execution of the program such that**

the while loop *never* terminates

Verification Condition in Mu-Arithmetic:

$$\forall x. \left(x > 0 \Rightarrow S(x)\right) \text{ where}$$
$$S(x) =_{\nu} (x \leq 0 \Rightarrow \bot) \wedge \left(x > 0 \Rightarrow \exists r. S\,(r - x)\right)$$

Example: *Partial Correctness* Verification with *Infinitely-Branching Angelic Non-Determinism*

Pre-condition $\{x > 0\}$

$\textbf{while } x > 0 \textbf{ do}$

Non-det. integer $\quad x \leftarrow \textbf{read\_int}(\,)^{\exists} - x$

$\textbf{done}$

Post-condition $\{\bot\}$ **Contradiction**

If the initial state satisfies the pre-condition $x > 0$

***there is* an execution of the program such that**

the while loop *never* terminates

Verification Condition in Mu-Arithmetic:

$$\forall x. \left(x > 0 \Rightarrow S(x)\right) \text{ where}$$

$$S(x) =_{\nu} x > 0 \wedge \exists r. S\,(x, r - x)$$

Key Advantages of $\boldsymbol{\mu}$CLP for Verification

1. Can naturally encode a wide variety of verification problems by exploiting the ***modularity*** in both the program and the specification
2. ***Closed under complement***: the complement of each (co-)inductive predicate is obtained by taking the standard De Morgan's dual:
$$\neg(\mu X.\lambda\vec{x}.\phi) \Leftrightarrow (\nu Y.\lambda\vec{x}.\neg[\neg Y/X]\phi)$$
$$\neg(\nu X.\lambda\vec{x}.\phi) \Leftrightarrow (\mu Y.\lambda\vec{x}.\neg[\neg Y/X]\phi)$$
$\Rightarrow$ By utilizing this, we present a novel $\boldsymbol{\mu}$**CLP** validity checking method **MuVal** that ***solves the primal and dual problems in parallel by exchanging useful information*** to reduce each others' solution spaces

μCLP Encoding of Various Verification Problems

- Can exploit the modularity of both the program and the specification by expressing each loop and (recursive) function in the program, as well as each subformula of the property, as separate (possibly mutually dependent) (co-)inductive predicates
  - Modular (non-)termination verification of imperative programs [POPL 2023]
  - Omega-regular model checking of labeled transition systems [POPL 2023]
  - Modal μ-calculus model checking of imperative programs [SAS 2019]
  - Omega-regular model checking of first-order recursive programs [SAS 2019]

[SAS 2019] Kobayashi et al. Temporal Verification of Programs via First-Order Fixpoint Logic.
[POPL 2023] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.

μCLP Encoding of Various Verification Problems

- Can exploit the modularity of both the program and the specification by expressing each loop and (recursive) function in the program, as well as each subformula of the property, as separate (possibly mutually dependent) (co-)inductive predicates
  - **Modular (non-)termination verification of imperative programs [POPL 2023]**
  - Omega-regular model checking of labeled transition systems [POPL 2023]
  - Modal μ-calculus model checking of imperative programs [SAS 2019]
  - Omega-regular model checking of first-order recursive programs [SAS 2019]

[SAS 2019] Kobayashi et al. Temporal Verification of Programs via First-Order Fixpoint Logic.
[POPL 2023] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.

Running Example (from [Urban+'13,'14]):

See the paper for a formal definition of this sound and complete modular encoding for termination

```
assume (x2 <= 3);
while (x1 >= 0 && x2 >= 0) {
  if (nondet()) {
    while (x2 != 3 && nondet()) { x2 = x2 + 1; }
    x1 = x1 - 1; }
  x2 = x2 - 1; }
```

The weakest precondition for the *termination* of the outer loop

Modular Encoding for Termination

$$\forall x_1, x_2 \colon \texttt{int}.\ x_2 > 3 \lor I(x_1, x_2) \text{ where}$$

The weakest precondition for the *termination* of the inner loop

$$I(x_1, x_2) \;=_\mu\; \neg(x_1 \geq 0 \land x_2 \geq 0) \lor \left(\begin{array}{l} I(x_1, x_2 - 1) \land \mathcal{J}(x_2) \land \\ \forall x_2' \colon \texttt{int}.\ P(x_2, x_2') \Rightarrow I(x_1 - 1, x_2' - 1) \end{array} \right)$$

$$\mathcal{J}(x_2) \;=_\mu\; \neg(x_2 \neq 3) \lor \mathcal{J}(x_2 + 1)$$

$$P(x_2, x_2') \;=_\mu\; x_2' = x_2 \lor x_2 \neq 3 \land P(x_2 + 1, x_2')$$

The strongest postcondition of the inner loop

Running Example (from [Urban+'13,'14]):

See the paper for a formal definition of this sound and complete modular encoding for termination

```
assume (x2 <= 3);
while (x1 >= 0 && x2 >= 0) {
  if (nondet()) {
    while (x2 != 3 && nondet()) { x2 = x2 + 1; }
    x1 = x1 - 1; }
  x2 = x2 - 1; }
```

Modular Encoding for Termination

The weakest precondition for the *termination* of the *outer* loop

$$\forall x_1, x_2 \colon \mathtt{int}.\ x_2 > 3 \vee I(x_1, x_2) \text{ where}$$

The weakest precondition for the *termination* of the inner loop

$$\begin{aligned}
I(x_1, x_2) &=_\mu \neg(x_1 \geq 0 \wedge x_2 \geq 0) \vee \left(\begin{array}{l} I(x_1, x_2 - 1) \wedge \mathcal{J}(x_2) \wedge \\ \forall x_2' \colon \mathtt{int}.\ NP(x_2, x_2') \vee I(x_1 - 1, x_2' - 1) \end{array} \right) \\
\mathcal{J}(x_2) &=_\mu \neg(x_2 \neq 3) \vee \mathcal{J}(x_2 + 1) \\
NP(x_2, x_2') &=_\nu \neg \left(x_2' = x_2 \vee x_2 \neq 3 \wedge \neg NP(x_2 + 1, x_2') \right)
\end{aligned}$$

The *complement* of the strongest postcondition of the inner loop

μ**CLP** encoding of the **termination** verification problem

$\forall x_1, x_2 \colon \mathtt{int}.\ x_2 > 3 \lor I(x_1, x_2)$ where

$$\begin{aligned} I(x_1,x_2) \quad &=_\mu \quad \neg(x_1 \geq 0 \land x_2 \geq 0) \lor \left(\begin{array}{l} I(x_1, x_2 - 1) \land \mathcal{J}(x_2) \land \\ \forall x_2' \colon \mathtt{int}.\ \ NP(x_2, x_2') \lor I(x_1 - 1, x_2' - 1) \end{array} \right) \\ \mathcal{J}(x_2) \quad &=_\mu \neg(x_2 \neq 3) \lor \mathcal{J}(x_2 + 1) \\ NP(x_2, x_2') \quad &=_\nu \neg\left(\ x_2' = x_2 \lor x_2 \neq 3 \land \neg NP(x_2 + 1, x_2')\ \right) \end{aligned}$$

The *complement* of I: The weakest precondition for the *non-termination* of the *outer* loop

Morgan's Dual

μ**CLP** encoding of the **n** **ermination** verific

The *complement* of J: The weakest precondition for the *non-termination* of the *inner* loop

$\exists x_1, x_2 \colon \mathtt{int}.\ x_2 \leq 3 \land NI(x_1, x_2)$ where

$$\begin{aligned} NI(x_1,x_2) \quad &=_\nu \quad x_1 \geq 0 \land x_2 \geq 0 \land \left(\begin{array}{l} NI(x_1, x_2 - 1) \lor N\mathcal{J}(x_2) \lor \\ \exists x_2' \colon \mathtt{int}.\ \ P(x_2, x_2') \land NI(x_1 - 1, x_2' - 1) \end{array} \right) \\ N\mathcal{J}(x_2) \quad &=_\nu x_2 \neq 3 \land N\mathcal{J}(x_2 + 1) \\ P(x_2, x_2') \quad &=_\mu \quad x_2' = x_2 \lor x_2 \neq 3 \land P(x_2 + 1, x_2') \end{aligned}$$

The *complement* of NP: the strongest postcondition of the inner loop

μCLP Encoding of Various Verification Problems

- Can exploit the modularity of both the program and the specification by expressing each loop and (recursive) function in the program, as well as each subformula of the property, as separate (possibly mutually dependent) (co-)inductive predicates
  - Modular (non-)termination verification of imperative programs [POPL 2023]
  - Omega-regular model checking of labeled transition systems [POPL 2023]
  - **Modal μ-calculus model checking of imperative programs [SAS 2019]**
  - Omega-regular model checking of first-order recursive programs [SAS 2019]

[SAS 2019] Kobayashi et al. Temporal Verification of Programs via First-Order Fixpoint Logic.
[POPL 2023] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.

Example 1

Initial state: $\{x \mapsto 0, y \mapsto 0\}$

Code:

$$\begin{Bmatrix} 0 \mapsto x \coloneqq x - 1; \mathbf{goto}\ 1, \\ 1 \mapsto y \coloneqq y + 1; \mathbf{goto}\ 0 \end{Bmatrix}$$

Specification:

$$X =_\nu x + y \geq 0 \land \Box Y$$
$$Y =_\nu \Box X$$

μCLP: $X^{(0)}(0,0)$ where

$$X^{(0)}(x, y) =_\nu x + y \geq 0 \land Y^{(1)}(x - 1, y)$$
$$Y^{(1)}(x, y) =_\nu X^{(0)}(x, y + 1)$$

Example 2

Initial state: $\{x \mapsto 0\}$
Code:

$$\left\{\begin{array}{c} 0 \mapsto x \coloneqq *; \mathbf{goto}\ 1, \\ 1 \mapsto \mathbf{if}\ x \leq 0\ \mathbf{then}\ \mathbf{goto}\ 0 \\ \mathbf{else}\ x \coloneqq x - 1;\ \mathbf{goto}\ 1 \end{array}\right\}$$

Specification:

$$X =_\nu x \geq 0 \land \diamond X$$

μCLP: $X^{(0)}(0)$ where

$$X^{(0)}(x) =_\nu x \geq 0 \land \exists r. X^{(1)}(r)$$

$$X^{(1)}(x) =_\nu \left(\begin{array}{c} x = 0 \land X^{(0)}(x) \lor \\ x > 0 \land X^{(1)}(x-1) \end{array}\right)$$

Outline

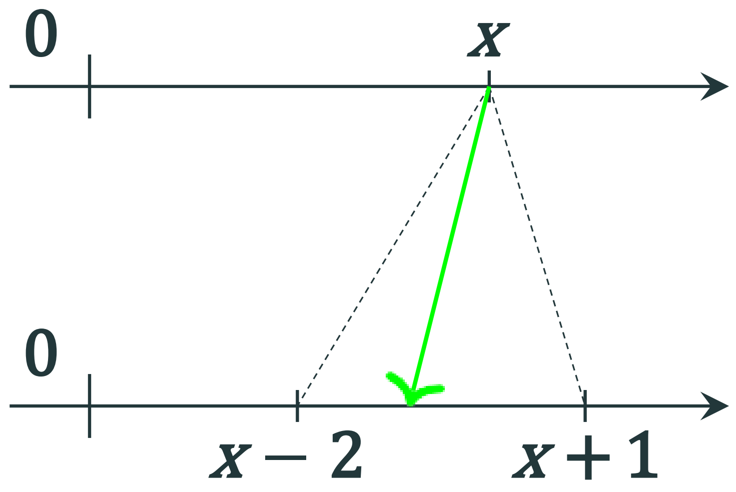
- Reduction from imperative programs to fixed-point logics (Mu-Arithmetic [CSL 1999] and μCLP [POPL 2023])
  - Safety verification
  - Termination verification
  - Non-termination verification
  - Modal μ-calculus model checking [SAS 2019]
- **Reduction from higher-order probabilistic programs to a higher-order and quantitative fixed-point logic**
  - Upper bounds verification of weakest pre-expectation [ICFP 2024]

[CSL 1999] Bradfield. Fixpoint Alternation and the Game Quantifier.
[SAS 2019] Kobayashi et al. Temporal Verification of Programs via First-Order Fixpoint Logic.
[POPL 2023] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.

Running Example

Program: random walk

$$
\begin{array}{l}
\mathbf{let\ rec\ rw}\ x = \ \mathbf{if}\ x \geq 0 \\
\qquad \mathbf{then}\ y \leftarrow \mathbf{uniform}_{[0,1]}; \\
\qquad\qquad (\mathbf{rw}\ (x + 3 \cdot y - 2))^{\checkmark} \\
\qquad \mathbf{else}\ ()
\end{array}
$$



Specification: "(the expected cost of $\mathbf{rw}\ 1$) ≤ 6"

where (the expected cost) = (the expected number of ✓).

Expected Cost Analysis via CPS Transformation

$$\begin{array}{c} \boxed{\text{functional (probabilistic) program}} \\ \Big\downarrow \text{CPS} \\ \boxed{\text{HFL formula}} \end{array}$$

$\mathbf{rw} : \mathbf{real} \to \mathbf{unit}$

$\mathbf{let\ rec\ rw}\ x = \ \mathbf{if}\ x \geq 0$

$\quad \mathbf{then}\ y \leftarrow \mathbf{uniform}_{[0,1]}; (\mathbf{rw}\ (x + 3 \cdot y - 2))^{\checkmark}$

$\quad \mathbf{else}\ ()$

$\mathbf{rw}' : \mathbf{real} \to (\mathbf{unit} \to [0,\infty]) \to [0,\infty]$

$\mathbf{let\ fix\ rw}'\ x\ k = \ \mathbf{if}\ x \geq 0$

$\quad \mathbf{then\ unif}(\lambda y.1 + \mathbf{rw}'(x + 3 \cdot y - 2)\ k)$

$\quad \mathbf{else}\ k\,()$

(expected cost of $\mathbf{rw}\ x$) = $\mathbf{rw}'\,x(\lambda r.0)$ [Avanzini et al., ICFP'21]

CPS = Continuation-Passing Style, HFL = (generalized) Higher-order Fixed-point Logic

Generic Weakest Precondition via CPS (1/2)

[Kura, 2023]
[Aguirre et al., 2022]

[Avanzini et al., ICFP'21]

- Expected cost
- (Weakest pre-expectation)

generalize →

Generic weakest preconditions

- Expected cost
- Higher moments of cost
- Weakest pre-expectation
- Conditional weakest pre-expectation
- . . .

Generic Weakest Precondition via CPS (2/2)

- There are general category-theoretic frameworks for WPs
- CPS $\cong$ WP holds for various kinds of effectful programs.

Input continuation

$$\mathtt{cps}[M] : X \to (Y \to \mathtt{Ans}) \to \mathtt{Ans}$$

$$M : X \to Y \qquad \cong$$

$$\mathtt{wp}[M] : (Y \to \mathtt{Prop}) \to (X \to \mathtt{Prop})$$

Postcondition weakest precondition

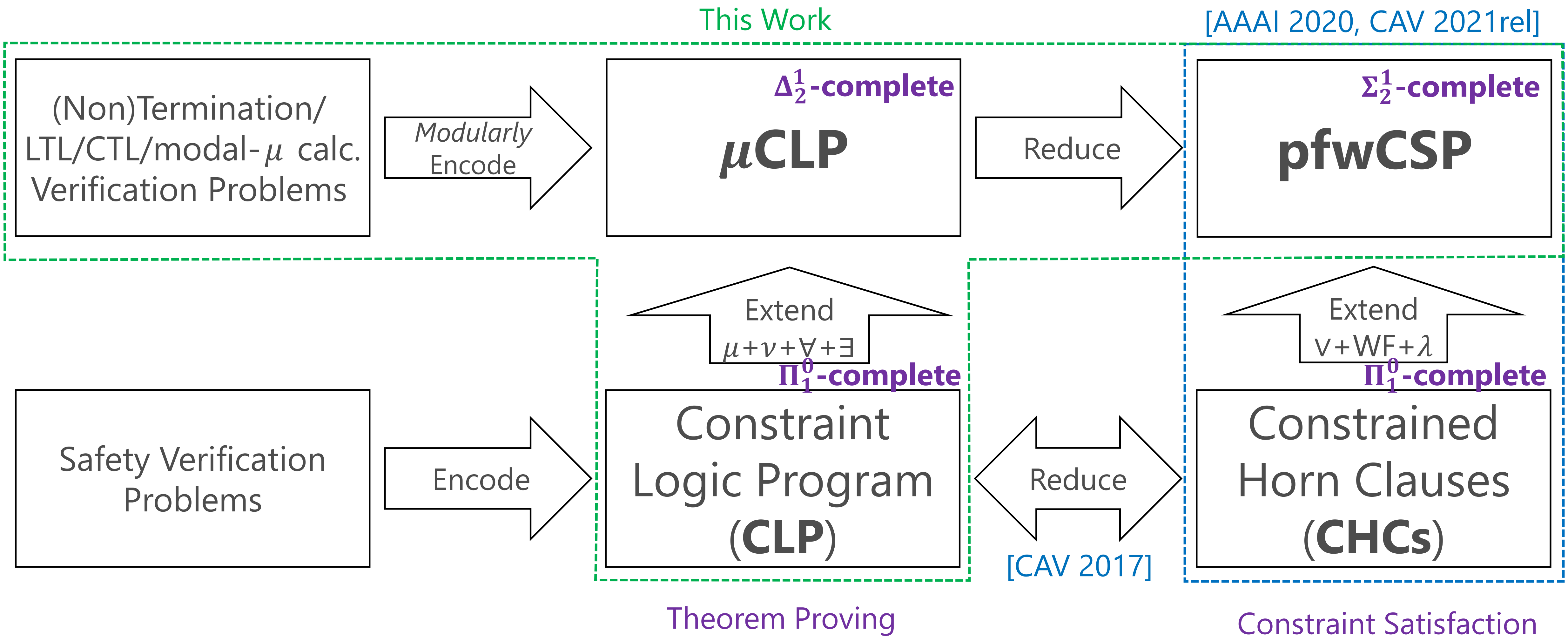
Our verification framework supports generic WPs.

Course Schedule

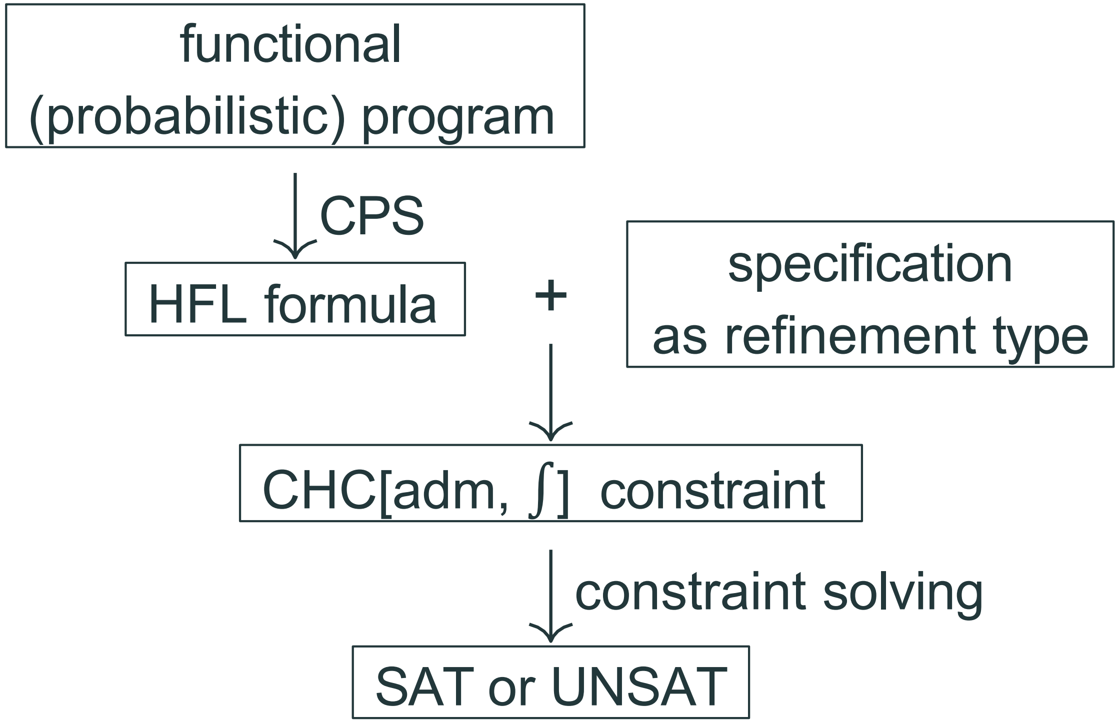
- Wed. 21 May (8:50-10:30)
    1. Reduction from software verification to fixed-point logic validity checking
    2. **Predicate constraint solving for validity checking**
- Thu. 22 May (11:20-12:20)
    3. Cyclic-proof search for validity checking
    4. Game solving for validity checking

2. Predicate Constraint Solving for Validity Checking

Overview of Our μCLP-based Framework [POPL 2023mod]



Overview of Our HFL-based Framework [ICFP 2024]



A uniform framework for

- expected cost
- cost moment
- weakest pre-expectation
- conditional weakest pre-expectation
- (other WP-based verification)

CPS = Continuation-Passing Style, HFL = (generalized) Higher-order Fixed-point Logic

Outline

- Classes of predicate constraint solving problems
- Reduction from validity checking for Mu-Arithmetic and μCLP [POPL 2023mod]
- Reduction from validity checking for the quantitative variant of HFL [ICFP 2024]
- CounterExample Guided Inductive Synthesis (CEGIS) for predicate constraint solving [AAAI 2020, CAV 2021rel, CAV 2021dt, ICFP 2024]

[POPL 2023mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.
[AAAI 2020] Satake et al. Probabilistic Inference for Predicate Constraint Satisfaction.
[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[POPL 2023opt] Gu et al. Optimal CHC Solving via Termination Proofs.

Outline

- **Classes of predicate constraint solving problems**
- Reduction from validity checking for Mu-Arithmetic and μCLP [POPL 2023mod]
- Reduction from validity checking for the quantitative variant of HFL [ICFP 2024]
- CounterExample Guided Inductive Synthesis (CEGIS) for predicate constraint solving [AAAI 2020, CAV 2021rel, CAV 2021dt, ICFP 2024]

[POPL 2023mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.
[AAAI 2020] Satake et al. Probabilistic Inference for Predicate Constraint Satisfaction.
[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[POPL 2023opt] Gu et al. Optimal CHC Solving via Termination Proofs.

Constraint-based Verification with Constrained Horn Clauses (**CHCs**)

Target Program $\mathcal{P}$ & Specification ψ

Verification intermediary independent of particular target and method ☺

Constraint Generation

RustHorn [Matsushita+'20,...]
JayHorn [Kahsai+'16,...]
SeaHorn [Gurfinkel+'15,...]
RCaml [PPDP09,...]

CHCs Constraints $\mathcal{C}$ on ***Predicate Variables***

Constraint Solving

SPACER [Komuravelli+ '14,...]
Eldarica [Hojjat+ '18,...]
Hoice [Champion+ '18,...]
PCSat [AAAI20,CAV21,...]

$\mathcal{C}$ is **Sat** ($\mathcal{P}$ satisfies ψ),
$\mathcal{C}$ is **Unsat** ($\mathcal{P}$ violates ψ),
or **Unknown**

CHCs: Constrained Horn Clauses (see e.g., [Bjørner+ '15])

- A finite set $\mathcal{C}$ of ***Horn-clauses*** of either form:

$$X_0(\overrightarrow{t_0}) \Longleftarrow \left(X_1(\overrightarrow{t_1}) \land \cdots \land X_m(\overrightarrow{t_m}) \land \phi\right)$$

$$\text{or } \bot \Longleftarrow \left(X_1(\overrightarrow{t_1}) \land \cdots \land X_m(\overrightarrow{t_m}) \land \phi\right)$$

  where $X_0, X_1 \dots, X_m$ are predicate variables,
  $\overrightarrow{t_0}, \dots, \overrightarrow{t_m}$ are sequences of terms of a first-order theory T,
  ϕ is a formula of T without predicate variables.

- $\mathcal{C}$ is ***satisfiable*** (modulo T) if there is an interpretation ρ of predicate variables such that $\rho \vDash \bigwedge \mathcal{C}$

Prog. & Spec. → CHCs → SAT or UNSAT

Constraint Generation   Constraint Solving

Example Program and *Partial Correctness* Specification:

Pre-condition

$\{x = x_0\}$
$y = 0;$
$\textbf{while}\ x \neq 0\ \textbf{do}$
$\quad y \leftarrow y + 1;$
$\quad x \leftarrow x - 1$
$\textbf{done}$
$\{y = x_0\}$

Post-condition

If the initial state satisfies the pre-condition $x = x_0$ and *the loop terminates*

the post-condition $y = x_0$ is satisfied by the resulting state

Prog. & Spec. → CHCs → SAT or UNSAT

Constraint Generation

Constraint Solving

Input:

$\{x = x_0\}$
$y = 0;$
while $x \neq 0$ **do**
$\quad y \leftarrow y + 1;$
$\quad x \leftarrow x - 1$
done
$\{y = x_0\}$

Output $\mathcal{C}$:

represents a ***loop invariant***

① $\underline{S}(x_0, x, y) \Leftarrow x = x_0 \land y = 0$,

② $\underline{S}(x_0, x - 1, y + 1)$
$\Leftarrow \underline{S}(x_0, x, y) \land x \neq 0$,

③ $y = x_0 \Leftarrow \underline{S}(x_0, x, y) \land x = 0$

$\mathcal{C}$ is ***satisfiable***, witnessed by a solution $\underline{S}(x_0, x, y) \equiv x_0 = x + y$

$\forall x_0, x, y. \big((x = x_0 \wedge y = 0) \Rightarrow S(x_0, x, y)\big)$ where
$S(x_0, x, y) =_\nu (x = 0 \Rightarrow y = x_0) \wedge \big(x \neq 0 \Rightarrow S(x_0, x - 1, y + 1)\big)$

Constraint Generation **Constraint Solving**

Input:

$\{x = x_0\}$
$y = 0;$
while $x \neq 0$ **do**
$\quad y \leftarrow y + 1;$
$\quad x \leftarrow x - 1$
done
$\{y = x_0\}$

Output $\mathcal{C}$:

represents a ***loop invariant***

① $\underline{S}(x_0, x, y) \Longleftarrow x = x_0 \wedge y = 0,$

② $\underline{S}(x_0, x - 1, y + 1)$
$\quad \Longleftarrow \underline{S}(x_0, x, y) \wedge x \neq 0,$

③ $y = x_0 \Longleftarrow \underline{S}(x_0, x, y) \wedge x = 0$

$\mathcal{C}$ is ***satisfiable***, witnessed by a solution $\underline{S}(x_0, x, y) \equiv x_0 = x + y$

Limitations of the class of CHCs

- Basically limited to verification of **∀linear-time safety** properties
- **Safety** vs. **liveness** properties
  - **Safety** is a class of properties of the form ***"something bad will never happen"***
    - Examples (absence of): assertion failure, division-by-zero, array boundary violation, …
  - **Liveness** is a class of properties of the form ***"something good will eventually happen"***
    - Examples: termination, deadlock freedom, …
- **Linear-time** vs. **branching-time** properties
  - The target program $\boldsymbol{P}$ may exhibit non-determinism caused by user input, scheduling, …
  - ***Linear-time*** verification concerns properties of the ***set of execution traces*** of $\boldsymbol{P}$
    - ∀***Linear-time***: ***any*** execution of $\boldsymbol{P}$ satisfies the specification?
    - ∃***Linear-time***: ***some*** execution of $\boldsymbol{P}$ satisfies the specification?
  - ***Branching-time*** verification concerns properties of the ***computation tree*** of $\boldsymbol{P}$
    - Allows arbitrary alternation of ∀ and ∃ and subsumes both ∀ **and** ∃ **linear-time** verification

pCSP: Predicate Constraint Satisfaction Problem [AAAI 2020]

- Generalize the class of CHCs with ***non-Horn clauses***
- A finite set $\mathcal{C}$ of ***clauses*** of the form:

$\mathcal{C}$ is ***CHCs*** if $\ell \leq 1$ for all clause in $\mathcal{C}$

$$\boxed{\left(X_1(\overrightarrow{t_1}) \vee \cdots \vee X_\ell(\overrightarrow{t_\ell})\right)} \Leftarrow \left(X_{\ell+1}(\overrightarrow{t_{\ell+1}}) \wedge \cdots \wedge X_m(\overrightarrow{t_m}) \wedge \phi\right)$$

where $X_1, \ldots, X_\ell, X_{\ell+1}, \ldots, X_m$ are predicate variables,
$\overrightarrow{t_1}, \ldots, \overrightarrow{t_m}$ are sequences of terms of a first-order theory T,
ϕ is a formula of T without predicate variables

- $\mathcal{C}$ is ***satisfiable*** (modulo T) if there is an interpretation ρ of predicate variables such that $\rho \vDash \bigwedge \mathcal{C}$
- Applicable to ***(finitely-) branching-time safety*** verification ☺

Prog. & Spec. → pCSP → SAT or UNSAT

Constraint Generation　Constraint Solving

Example Program and Specification:

Pre-condition

$\{x > 0\}$

while $x \neq 0$ **do**

Non-det. branching

if $\mathbf{read\_bool(\,)}^{\exists}$ **then**

$x \leftarrow x - 1$

else

$x \leftarrow x + 1$

done

Post-condition

$\{\bot\}$

Contradiction

If the initial state satisfies the pre-condition $x > 0$

there is an execution of the program such that

the post-condition $\bot$ is satisfied when the while loop terminates

Prog. & Spec. → **pCSP** → **SAT or UNSAT**

Constraint Generation   Constraint Solving

Example Program and Specification:

Pre-condition: $\{x > 0\}$

$\textbf{while } x \neq 0 \textbf{ do}$

Non-det. branching: $\quad \textbf{if } \mathbf{read\_bool(\,)}^{\exists} \textbf{ then}$

$\qquad x \leftarrow x - 1$

$\quad \textbf{else}$

$\qquad x \leftarrow x + 1$

$\textbf{done}$

Post-condition: $\{\perp\}$

Contradiction

If the initial state satisfies the pre-condition $x > 0$

there is an execution of the program such that

the while loop *never* terminates

Prog. & Spec. → **pCSP** → **SAT or UNSAT**

Constraint Generation

Constraint Solving

Input:

$\{x > 0\}$
while $x \neq 0$ **do**
  if **read_bool( )**$^{\exists}$ **then**
    $x \leftarrow x - 1$
  else
    $x \leftarrow x + 1$
done
$\{\perp\}$

Output $\mathcal{C}$**:**

represents a ***loop invariant*** preserved by ***some*** execution (i.e., recurrent set)

① $\underline{S}(x) \Leftarrow x > 0,$

② $\underline{S}(x-1) \vee \underline{S}(x+1) \Leftarrow \underline{S}(x) \wedge x \neq 0,$

$\mathcal{C}$ is beyond CHCs!

③ $\perp \Leftarrow \underline{S}(x) \wedge x = 0$

$\mathcal{C}$ is ***satisfiable***, witnessed by a solution $\underline{S}(x) \equiv x > 0$

$\forall x. \left(x > 0 \Rightarrow S(x)\right)$ where

$S(x) =_\nu (x = 0 \Rightarrow \bot) \wedge \left(x \neq 0 \Rightarrow \left(S(x-1) \vee S(x+1)\right)\right)$

Constraint Generation

Constraint Solving

Input:

$\{x > 0\}$
while $x \neq 0$ **do**
  if **read_bool( )**$^\exists$ **then**
    $x \leftarrow x - 1$
  else
    $x \leftarrow x + 1$
done
$\{\bot\}$

Output $\mathcal{C}$:

represents a ***loop invariant*** preserved by ***some*** execution (i.e., recurrent set)

① $\underline{S}(x) \Leftarrow x > 0,$

② $S(x-1) \vee \underline{S}(x+1) \Leftarrow \underline{S}(x) \wedge x \neq 0,$

$\mathcal{C}$ is beyond CHCs!

③ $\bot \Leftarrow S(x) \wedge x = 0$

$\mathcal{C}$ is ***satisfiable***, witnessed by a solution $\underline{S}(x) \equiv x > 0$

pfwCSP: pCSP with Functional and Well-founded Predicates [CAV 2021rel] (cf. ∀∃CHCs with dwf [Beyene+'13])

- A finite set $\mathcal{C}$ of ***clauses*** with a map $\mathcal{K}$ from predicate variables X in $\mathcal{C}$ to $\{\star, \lambda, \Downarrow\}$
  - X is ordinary predicate if $\mathcal{K}(X) = \star$
  - X is ***functional*** predicate if $\mathcal{K}(X) = \lambda$
  - X is ***well-founded*** predicate if $\mathcal{K}(X) = \Downarrow$
- $\mathcal{C}$ is ***satisfiable*** (modulo T) if there is a predicate interpretation ρ such that
  - $\rho \models \bigwedge \mathcal{C}$
  - $\forall X. \mathcal{K}(X) = \lambda \implies \rho(X)$ characterizes a ***total function***
  - $\forall X. \mathcal{K}(X) = \Downarrow \implies \rho(X)$ represents a ***well-founded relation***
- Applicable to ***(infinitely-) branching-time safety & liveness*** verification ☺

Prog. & Spec. → pfwCSP → SAT or UNSAT

Constraint Generation — Constraint Solving

Example Program and Specification:

Pre-condition: $\{x > 0\}$

If the initial state satisfies the pre-condition $x > 0$

$\textbf{while}\ x > 0\ \textbf{do}$

Non-det. integer: $x \leftarrow \textbf{read\_int}(\,)^{\exists} - x$

$\textbf{done}$

there is an execution of the program such that

Post-condition: $\{\bot\}$ — Contradiction

the while loop *never* terminates

Prog. & Spec. → **pfwCSP** → **SAT or UNSAT**

Constraint Generation

Constraint Solving

Input:

$\{x > 0\}$
$\textbf{while } x > 0 \textbf{ do}$
$\quad x \leftarrow \textbf{read\_int}(\,)^{\exists} - x$
$\textbf{done}$
$\{\bot\}$

$\mathcal{C}$ is beyond **pCSP** but can be encoded in **pfwCSP** using a ***functional pred. var.*** that characterizes a ***Skolem function*** for r

Output $\mathcal{C}$:

represents a ***loop invariant*** preserved by ***some*** execution (i.e., recurrent set)

① $\underline{S}(x) \Leftarrow x > 0,$

② $\left(\exists r.\underline{S}(r - x)\right) \Leftarrow \underline{S}(x) \land x > 0,$

$\bot \Leftarrow \underline{S}(x) \land x \le 0$

Prog. & Spec. → **pfwCSP** → **SAT or UNSAT**

Constraint Generation — Constraint Solving

Input:

$\{x > 0\}$

$\textbf{while } x > 0 \textbf{ do}$

$\quad x \leftarrow \textbf{read\_int}(\,)^{\exists} - x$

$\textbf{done}$

$\{\bot\}$

Output $\mathcal{C}$:

characterizes a ***Skolem* function** mapping x to r

① $\underline{S}(x) \Leftarrow x > 0,$

② $\underline{S}(r - x) \Leftarrow T_\lambda(x, r) \land \underline{S}(x) \land x > 0,$

③ $\bot \Leftarrow \underline{S}(x) \land x \le 0$

$\mathcal{C}$ is ***satisfiable***, witnessed by a solution $\underline{S}(x) \equiv x > 0,\ T_\lambda(x, r) \equiv r = x + 1$

$$\forall x. \big(x > 0 \Rightarrow S(x)\big) \text{ where}$$
$$S(x) =_\nu (x \le 0 \Rightarrow \bot) \wedge \big(x > 0 \Rightarrow \exists r. S(r - x)\big)$$

Constraint Generation

Constraint Solving

represents a ***loop invariant*** preserved by ***some*** execution (i.e., recurrent set)

Input:

$\{x > 0\}$
while $x > 0$ **do**
  $x \leftarrow \mathbf{read\_int}(\,)^{\exists} - x$
done
$\{\bot\}$

Output $\mathcal{C}$**:**

① $\underline{S}(x) \Leftarrow x > 0,$

② $\big(\exists r. \underline{S}(r - x)\big)$
$\Leftarrow \underline{S}(x) \wedge x > 0,$

③ $\bot \Leftarrow \underline{S}(x) \wedge x \le 0$

Prog. & Spec. → **pfwCSP** → **SAT or UNSAT**

Constraint Generation — Constraint Solving

Example Program and *Total Correctness* Specification:

Pre-condition

$[x \neq 0]$

$\mathbf{while}\ x \neq 0\ \mathbf{do}$

$\quad \mathbf{if}\ x > 0\ \mathbf{then}$

$\quad\quad x \leftarrow x - 1$

$\quad \mathbf{else}$

$\quad\quad x \leftarrow x + 1$

$\mathbf{done}$

$[\top]$

Post-condition

Tautology

If the initial state satisfies the pre-condition $x \neq 0$

the loop always terminates and the post-condition $\top$ is satisfied by the resulting state

Prog. & Spec. → pfwCSP → SAT or UNSAT

Constraint Generation

Constraint Solving

Input:

$[x \neq 0]$
while $x \neq 0$ **do**
  if $x > 0$ **then**
    $x \leftarrow x - 1$
  else
    $x \leftarrow x + 1$
done
$[\top]$

Output $\mathcal{C}$:

represents a ***loop invariant***

① $\underline{S}(x) \Leftarrow x \neq 0,$

② $\underline{S}(x-1) \Leftarrow \underline{S}(x) \land x > 0,$

③ $\underline{S}(x+1) \Leftarrow \underline{S}(x) \land x < 0,$

④ $T_{\Downarrow}(x, x-1) \Leftarrow \underline{S}(x) \land x > 0,$

⑤ $T_{\Downarrow}(x, x+1) \Leftarrow \underline{S}(x) \land x < 0,$

represents a ***well-founded relation*** for termination of the loop

$\mathcal{C}$ is ***satisfiable***, witnessed by a solution
$\underline{S}(x) \equiv \top, T_{\Downarrow}(x, x') \equiv |x| > |x'| \geq 0$

$\forall x. \big(x \neq 0 \Rightarrow S(x)\big)$ where
$S(x) =_\mu \big(x > 0 \Rightarrow S(x-1)\big) \wedge \big(x < 0 \Rightarrow S(x+1)\big)$

Constraint Generation

Constraint Solving

Input:

$[x \neq 0]$
while $x \neq 0$ **do**
  if $x > 0$ **then**
    $x \leftarrow x - 1$
  else
    $x \leftarrow x + 1$
done
$[\top]$

Output $\mathcal{C}$:

represents a ***loop invariant***

① $\underline{S}(x) \Leftarrow x \neq 0,$
② $\underline{S}(x-1) \Leftarrow \underline{S}(x) \wedge x > 0,$
③ $\underline{S}(x+1) \Leftarrow \underline{S}(x) \wedge x < 0,$
④ $T_\Downarrow(x, x-1) \Leftarrow \underline{S}(x) \wedge x > 0,$
⑤ $T_\Downarrow(x, x+1) \Leftarrow \underline{S}(x) \wedge x < 0,$

represents a ***well-founded relation*** for termination of the loop

$\mathcal{C}$ is ***satisfiable***, witnessed by a solution
$\underline{S}(x) \equiv \top, T_\Downarrow(x, x') \equiv |x| > |x'| \geq 0$

CHC[adm, $\int$]: An Extension of CHCs for Generalized HFL

To support generalized HFL, we need to extend CHC

$$\begin{aligned}
&\mathbf{let\ fix\ rw}'\ x\ k = \ \mathbf{if}\ x \geq 0 && \leftarrow \text{ fixed point } \mathbf{fix} \\
&\qquad \mathbf{then\ unif}(\lambda y.1 + \ \mathbf{rw}'\ (x + \ 3 \cdot y - 2)\ \ k) && \leftarrow \text{ integration } \mathbf{unif} \\
&\qquad \mathbf{else}\ \ k\ ()
\end{aligned}$$

The extension CHC[adm, $\int$] has

- **admissible predicate variables** for fixed points,
- **integrable predicate variables** for integration operators

(These are explained later.)

Summary

- **pfwCSP**: an extension of CHCs with ***non-Horn clauses*** and ***functional***, ***well-founded*** predicate variables, with wide applications to:
  - Validity checking for Mu-Arithmetic, μCLP, HFL [CAV 2021dt, POPL 2023mod, ICFP 2024]
  - Dependent refinement type inference [PPDP 2009, …, POPL 2024, ICFP 2024, PLDI 2025]
  - Relational program verification [CAV 2021rel, VMCAI 2024]
    - Program equivalence, NI, co-termination, generalized NI, …
  - Optimality checking for solutions of CHCs [POPL 2023opt]
- **CHC[adm, ∫]**: an extension of CHCs for generalized HFL

[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[POPL 2023mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.
[PLDI 2025] Ogawa et al. Thrust: A Prophecy-based Refinement Type System for Rust.
[PPDP 2009] Unno and Kobayashi. Dependent Type Inference with Interpolants.
[POPL 2024] Kawamata et al. Answer Refinement Modification: Refinement Type System for Algebraic Effects and Handlers.
[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[VMCAI 2024] Unno. Automating Relational Program Verification.
[POPL 2023opt] Gu et al. Optimal CHC Solving via Termination Proofs.

Outline

- Classes of predicate constraint solving problems
- **Reduction from validity checking for Mu-Arithmetic and μCLP [POPL 2023mod]**
- Reduction from validity checking for the quantitative variant of HFL [ICFP 2024]
- CounterExample Guided Inductive Synthesis (CEGIS)
  for predicate constraint solving [AAAI 2020, CAV 2021rel, CAV 2021dt, ICFP 2024]

[POPL 2023mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.
[AAAI 2020] Satake et al. Probabilistic Inference for Predicate Constraint Satisfaction.
[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[POPL 2023opt] Gu et al. Optimal CHC Solving via Termination Proofs.

Example 1: Reduction of $\boldsymbol{\mu}$CLP Validity to pfwCSP Satisfiability

$$\forall x_0, x, y. \left(\boxed{(x = x_0 \wedge y = 0) \Rightarrow S(x_0, x, y)}\right) \text{ where}$$
$$S(x_0, x, y) =_\nu (x = 0 \Rightarrow y = x_0) \wedge \big(x \neq 0 \Rightarrow S(x_0, x-1, y+1)\big)$$

Knaster-Tarski:

$$\frac{\models \underline{S} \Rightarrow F(\underline{S})}{\models \underline{S} \Rightarrow \nu F}$$

$$F \triangleq \lambda \underline{S}. \lambda(x_0, x, y). \left(\begin{array}{c} (x = 0 \Rightarrow y = x_0) \wedge \\ \big(x \neq 0 \Rightarrow \underline{S}(x_0, x-1, y+1)\big) \end{array} \right)$$

① $\underline{S}(x_0, x, y) \Leftarrow x = x_0 \wedge y = 0,$

② $\underline{S}(x_0, x-1, y+1) \Leftarrow \underline{S}(x_0, x, y) \wedge x \neq 0,$

③ $y = x_0 \Leftarrow \underline{S}(x_0, x, y) \wedge x = 0$

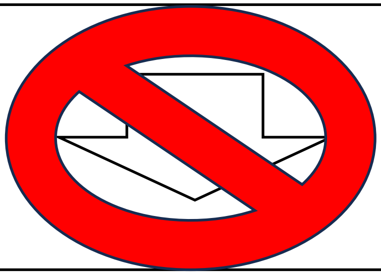
Example 2: Reduction of $\boldsymbol{\mu}$**CLP** Validity to **pfwCSP** Satisfiability

$\forall x. \boxed{\left(x \neq 0 \Rightarrow S(x)\right)}$ where

$S(x) =_{\mu} \left(x > 0 \Rightarrow S(x-1)\right) \wedge \left(x < 0 \Rightarrow S(x+1)\right)$

Knaster-Tarski:

$$\frac{\vDash F(\overline{S}) \Rightarrow \overline{S}}{\vDash \mu F \Rightarrow \overline{S}}$$



$$F \triangleq \lambda \overline{S}. \lambda x. \begin{pmatrix} \left(x > 0 \Rightarrow \overline{S}(x-1)\right) \wedge \\ \left(x < 0 \Rightarrow \overline{S}(x+1)\right) \end{pmatrix}$$

① $\overline{S}(x) \Leftarrow x \neq 0$,

② $\overline{S}(x-1) \Leftarrow \overline{S}(x) \wedge x > 0$,

③ $\overline{S}(x+1) \Leftarrow \overline{S}(x) \wedge x < 0$,

④ $T_{\Downarrow}(x, x-1) \Leftarrow \overline{S}(x) \wedge x > 0$,

⑤ $T_{\Downarrow}(x, x+1) \Leftarrow \overline{S}(x) \wedge x < 0$

Example 2: Reduction of $\boldsymbol{\mu}$**CLP** Validity to **pfwCSP** Satisfiability

$\forall x. \boxed{\left(x \neq 0 \Rightarrow S(x)\right)}$ where
$S(x) =_{\mu} \left(x > 0 \Rightarrow S(x-1)\right) \wedge \left(x < 0 \Rightarrow S(x+1)\right)$

Knaster-Tarski:

$$\frac{\vDash \underline{S} \Rightarrow G(\underline{S})}{\vDash \underline{S} \Rightarrow \nu G}$$

Use $G \triangleq \lambda \overline{S}. \lambda x. \left(\begin{array}{l} \left(x > 0 \Rightarrow T_{\Downarrow}(x, x-1) \wedge \overline{S}(x-1)\right) \wedge \\ \left(x < 0 \Rightarrow T_{\Downarrow}(x, x+1) \wedge \overline{S}(x+1)\right) \end{array} \right)$

s.t. $\nu G \Leftrightarrow \mu G \Rightarrow \mu F$

① $S(x) \Leftarrow x \neq 0$,

② $S(x-1) \Leftarrow S(x) \wedge x > 0$,

③ $S(x+1) \Leftarrow S(x) \wedge x < 0$,

④ $T_{\Downarrow}(x, x-1) \Leftarrow S(x) \wedge x > 0$,

⑤ $T_{\Downarrow}(x, x+1) \Leftarrow S(x) \wedge x < 0$

Sound and Complete Reduction of **μCLP** Validity to **pfwCSP** Satisfiability

1. Eliminate existential quantifiers via ***Skolemization*** using ***functional predicates***
2. Replace inductive predicates μF with *equivalent* co-inductive predicates νG where G is obtained from F by inserting ***guards*** (for checking the ***well-foundedness*** between the formal and actual arguments) for each recursion site
   - E.g. Let $F(x) =_\mu x = 0 \lor F(x-1)$ and $G(x) =_\nu x = 0 \lor G(x-1) \land WF(x, x-1)$.
     $\nu G \Leftrightarrow \mu G \Rightarrow \mu F$ for any w.f. rel. WF and $\mu G \Leftrightarrow \mu F$ for some w.f. rel. WF
   - Inspired by the deductive system [LICS 2018] for a first-order fixed-point logic and ***binary reachability analysis*** for reducing termination verification to safety verification using (disjunctively) well-founded relations [PLDI 2006, …]
3. Replace each co-inductive predicate X with a predicate variable $\underline{X}$ that represents an *unknown* under-approximation (or postfixpoint) of X to be synthesized

[LICS 2018] Nanjo et al. A Fixpoint Logic and Dependent Effects for Temporal Property Verification.
[PLDI 2006] Cook et al. Termination Proofs for Systems Code.

μCLP encoding the **termination** verification problem

$\forall x_1, x_2 \colon \mathtt{int}.\ x_2 > 3 \lor I(x_1, x_2)$ where

$$\begin{aligned}
I(x_1, x_2) \quad &=_\mu \quad \neg(x_1 \geq 0 \land x_2 \geq 0) \lor \begin{pmatrix} I(x_1, x_2 - 1) \land \mathcal{J}(x_2) \land \\ \forall x_2' \colon \mathtt{int}.\ \ NP(x_2, x_2') \lor I(x_1 - 1, x_2' - 1) \end{pmatrix} \\
\mathcal{J}(x_2) \quad &=_\mu \neg(x_2 \neq 3) \lor \mathcal{J}(x_2 + 1) \\
NP(x_2, x_2') \quad &=_\nu \neg\left(\ x_2' = x_2 \lor x_2 \neq 3 \land \neg NP(x_2 + 1, x_2')\ \right)
\end{aligned}$$

The corresponding pfwCSP

Predicate variable that represents an *under-approximation* of I

Well-founded predicate variable that represents the *guard* for the recursion on I

$$\begin{aligned}
&(1) \quad x_2 > 3 \lor \underline{I}(x_1, x_2) \\
&(2) \quad \underline{I}(x_1, x_2) \Rightarrow \ \neg(x_1 \geq 0 \land x_2 \geq 0) \lor \\
&\qquad \begin{pmatrix} \underline{I}(x_1, x_2 - 1) \land I_{\Downarrow}(x_1, x_2, x_1, x_2 - 1) \land \underline{\mathcal{J}}(x_2) \land \\ (\underline{NP}(x_2, x_2') \lor \underline{I}(x_1 - 1, x_2' - 1) \land I_{\Downarrow}(x_1, x_2, x_1 - 1, x_2' - 1)) \end{pmatrix} \\
&(3) \quad \underline{\mathcal{J}}(x_2) \Rightarrow \neg(x_2 \neq 3) \lor \underline{\mathcal{J}}(x_2 + 1) \land \mathcal{J}_{\Downarrow}(x_2, x_2 + 1) \\
&(4) \quad \underline{NP}(x_2, x_2') \Rightarrow \neg\left(\ x_2' = x_2 \lor x_2 \neq 3 \land \neg\underline{NP}(x_2 + 1, x_2')\ \right)
\end{aligned}$$

μCLP encoding the **non-termination** verification problem

$\exists x_1, x_2 \colon \mathtt{int}.\ x_2 \le 3 \land NI(x_1, x_2)$ where

$$
\begin{aligned}
NI(x_1,x_2) &=_\nu \quad x_1 \ge 0 \land x_2 \ge 0 \land \left(\begin{array}{l} NI(x_1, x_2 - 1) \lor NJ(x_2) \lor \\ \exists x_2' \colon \mathtt{int}.\ \ P(x_2, x_2') \land NI(x_1 - 1, x_2' - 1) \end{array} \right) \\
NJ(x_2) &=_\nu x_2 \ne 3 \land NJ(x_2 + 1) \\
P(x_2, x_2') &=_\mu \quad x_2' = x_2 \lor x_2 \ne 3 \land P(x_2 + 1, x_2')
\end{aligned}
$$

Functional predicate variable that represents the *Skolem function* for $\exists x_1$

duce

Functional predicate variable that represents the *Skolem function* for $\exists x_2'$

The

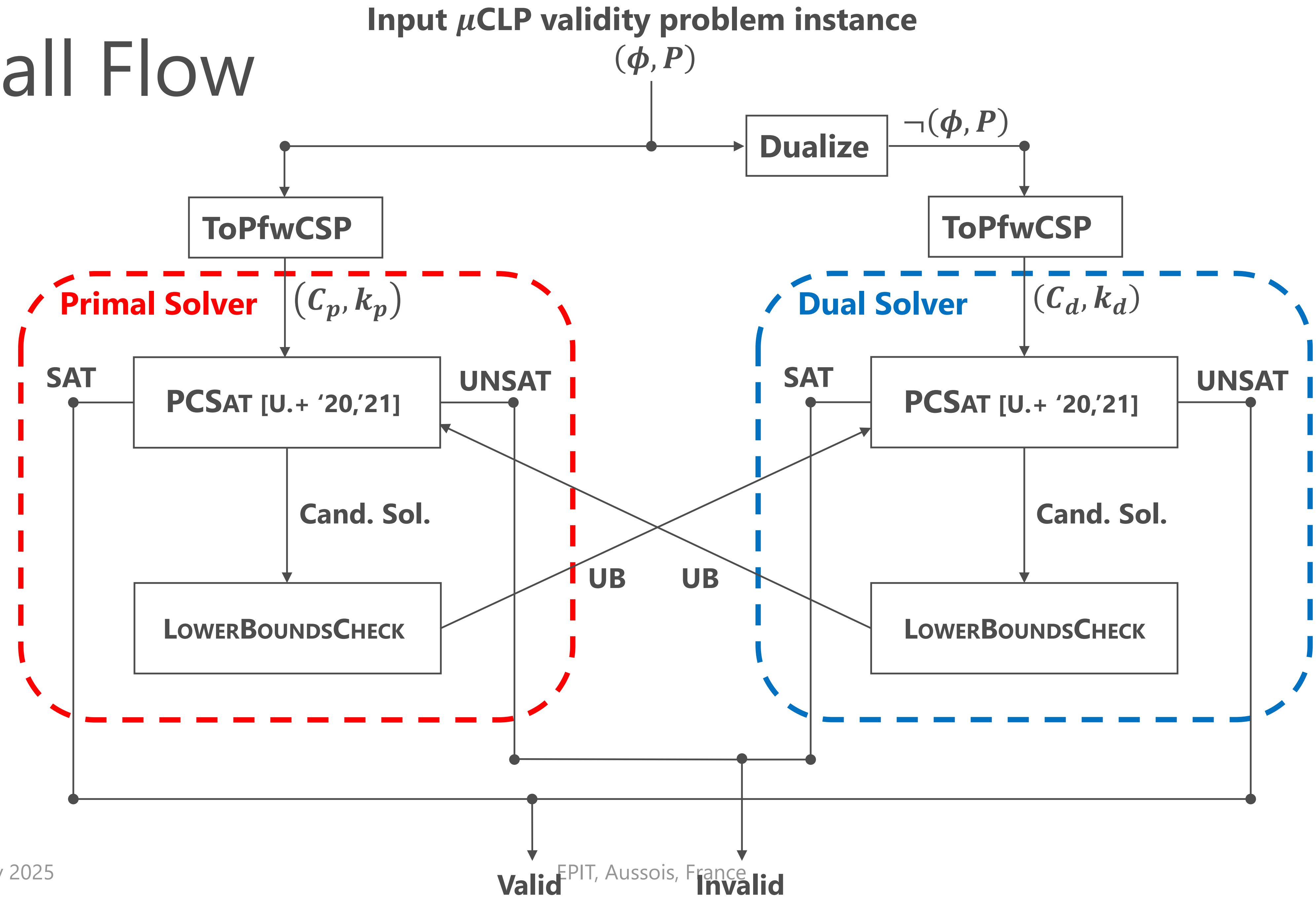
$$
\begin{array}{ll}
(5) & S_\lambda(x_1) \land T_\lambda(x_2) \Rightarrow x_2 \le 3 \land \underline{NI}(x_1, x_2) \\
(6) & \underline{NI}(x_1, x_2) \Rightarrow \quad x_1 \ge 0 \land x_2 \ge 0 \land \\
 & \qquad \left(\begin{array}{l} \underline{NI}(x_1, x_2 - 1) \lor \underline{NJ}(x_2) \lor \\ U_\lambda(x_1, x_2, x_2') \Rightarrow \left(\ \underline{P}(x_2, x_2') \land \underline{NI}(x_1 - 1, x_2' - 1) \ \right) \end{array} \right) \\
(7) & \underline{NJ}(x_2) \Rightarrow x_2 \ne 3 \land \underline{NJ}(x_2 + 1) \\
(8) & \underline{P}(x_2, x_2') \Rightarrow \quad x_2' = x_2 \lor x_2 \ne 3 \land \underline{P}(x_2 + 1, x_2') \land P_\Downarrow(x_2, x_2', x_2 + 1, x_2')
\end{array}
$$

MuVal: A μ**CLP** Validity Checking Method

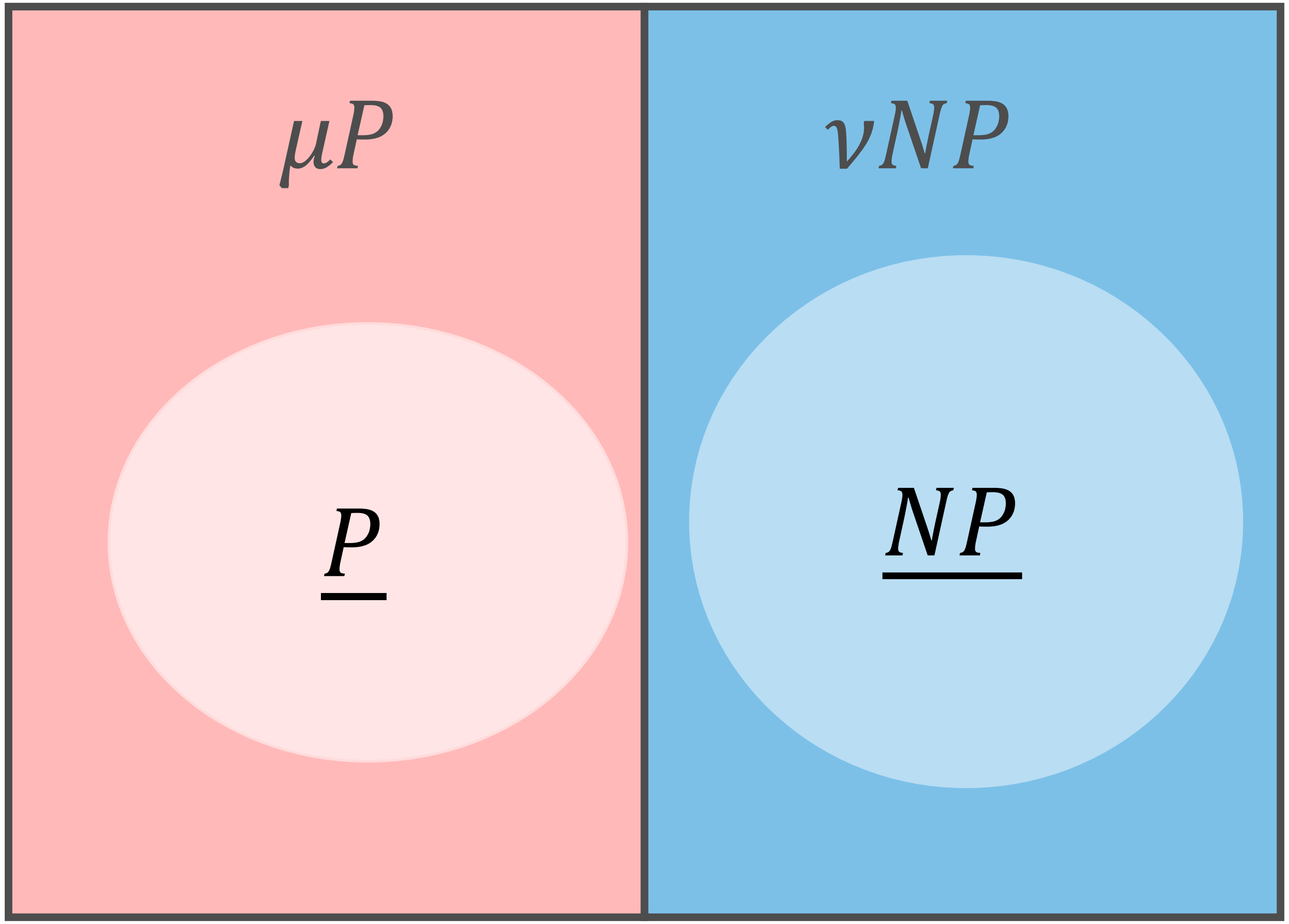
- The reduction to **pfwCSP** coupled with **PCSat** [AAAI 2020, CAV 2021rel], an existing CEGIS-based **pfwCSP** solver, already gives a method for checking μ**CLP** validity
- We further improve the method to ***Modular Primal-Dual Parallel Solving***
  - The ***primal*** and ***dual*** **pfwCSP** problems are constructed and solved in ***parallel***
  - **PCSat** is extended to synthesize ***lower-bounds*** for (co-)inductive predicates that can be used as ***upper-bounds*** of the corresponding dual predicates
  - Exchange each others' ***bounds*** to reduce each others' solution spaces

  Note that the bounds are synthesized and exchanged ***modularly***, at granularity of individual (co-)inductive predicates

Overall Flow



Intuition behind Exchanging Upper Bounds



$\nu NP \Leftrightarrow \neg\, \mu P$

$\underline{P}, \underline{NP}$ represent lower bounds synthesized by **PCSat**:
$\underline{P} \Rightarrow \mu P$ and $\underline{NP} \Rightarrow \nu NP$

Therefore,

$$\mu P \Rightarrow \neg\, \underline{NP}$$

and

$$\nu NP \Rightarrow \neg\, \underline{P}$$

The primal pfwCSP

$$
\begin{array}{ll}
(1) & x_2 > 3 \vee \underline{I}(x_1, x_2) \\
(2) & \underline{I}(x_1, x_2) \Rightarrow \neg(x_1 \geq 0 \wedge x_2 \geq 0) \vee \\
& \quad \left(\begin{array}{l} \underline{I}(x_1, x_2 - 1) \wedge I_{\Downarrow}(x_1, x_2, x_1, x_2 - 1) \wedge \underline{\mathcal{J}}(x_2) \wedge \\ (\underline{NP}(x_2, x_2') \vee \underline{I}(x_1 - 1, x_2' - 1) \wedge I_{\Downarrow}(x_1, x_2, x_1 - 1, x_2' - 1)) \end{array} \right) \\
(3) & \underline{\mathcal{J}}(x_2) \Rightarrow \neg(x_2 \neq 3) \vee \underline{\mathcal{J}}(x_2 + 1) \wedge \mathcal{J}_{\Downarrow}(x_2, x_2 + 1) \\
(4) & \underline{NP}(x_2, x_2') \Rightarrow \neg\left(x_2' = x_2 \vee x_2 \neq 3 \wedge \neg\underline{NP}(x_2 + 1, x_2') \right)
\end{array}
$$

The primal solver found a candidate solution $\{\underline{J}(x_2) \mapsto x_2 = 3, J_{\Downarrow} \mapsto \cdots\}$, which is a partial solution for $\underline{J}$ since it satisfies clause (3). We thus learned:

$$x_2 = 3 \Rightarrow J(x_2) \text{ and } NJ(x_2) \Rightarrow x_2 \neq 3$$

Send $\underline{NJ}(x_2) \Rightarrow x_2 \neq 3$ to the dual solver!

The dual solver then uses that information to learn $J(x_2) \Rightarrow x_2 \leq 3$.

Send $\underline{J}(x_2) \Rightarrow x_2 \leq 3$ to the primal solver.

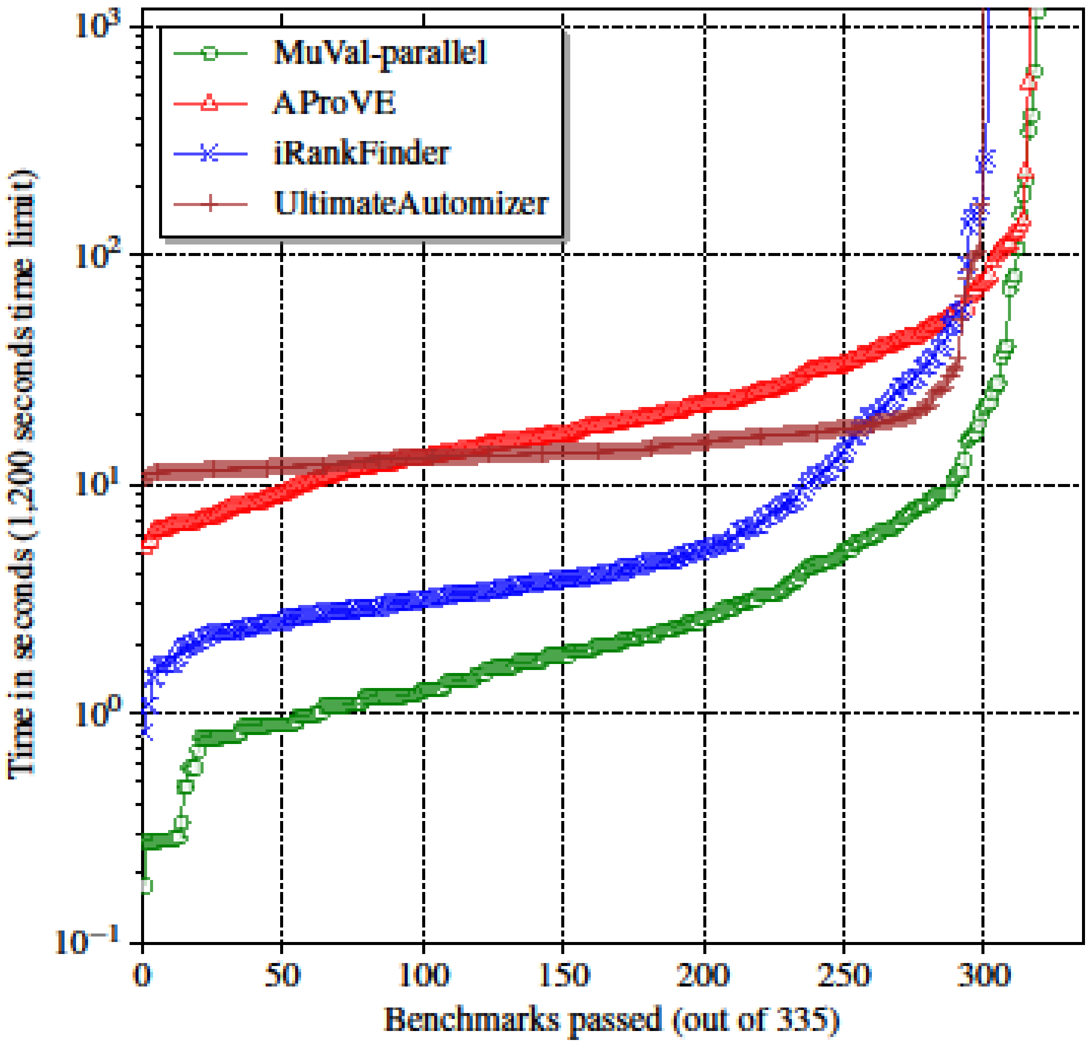
The primal solver then uses it to obtain an actual solution, thus proving termination:

$$
\begin{array}{rcl}
\underline{I}(x_1, x_2) & \mapsto & 3 \geq x_2, \\
\underline{\mathcal{J}}(x_2) & \mapsto & x_2 \geq 0 \wedge x_2 \leq 3, \\
\underline{NP}(x_2, x_2') & \mapsto & x_2 \geq 0 \wedge x_2 \leq 3 \wedge x_2' \geq 4
\end{array}
$$

Implementation and Evaluation

- Implemented **MuVal** in OCaml 5, using Z3 as the backend SMT solver
  - Support integers, reals, and algebraic data types as background theories
- Evaluated with
  1. (Non-)termination verification benchmarks from TermComp '21 (C Integer)
  2. Temporal verification benchmarks
     - LTL verification benchmarks from [Cook&Koskinen'13]
     - CTL verification benchmarks from [Dietsch+'15]
     - MuArith (= μCLP over integer arithmetic) benchmarks from [Kobayashi+'19]
       - Contain CTL* and modal-μ calculus model checking problems of infinite state systems
     - Termination verification benchmarks from [Urban+'13,'14] modularly encoded as μCLP

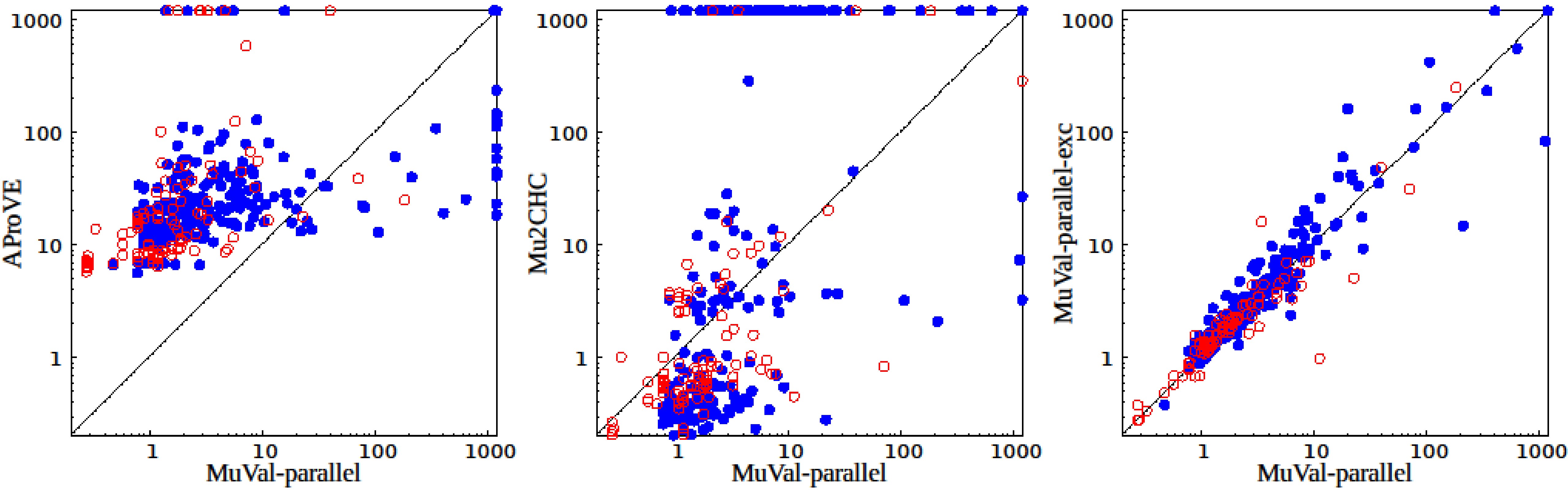
Evaluation with TermComp Benchmarks



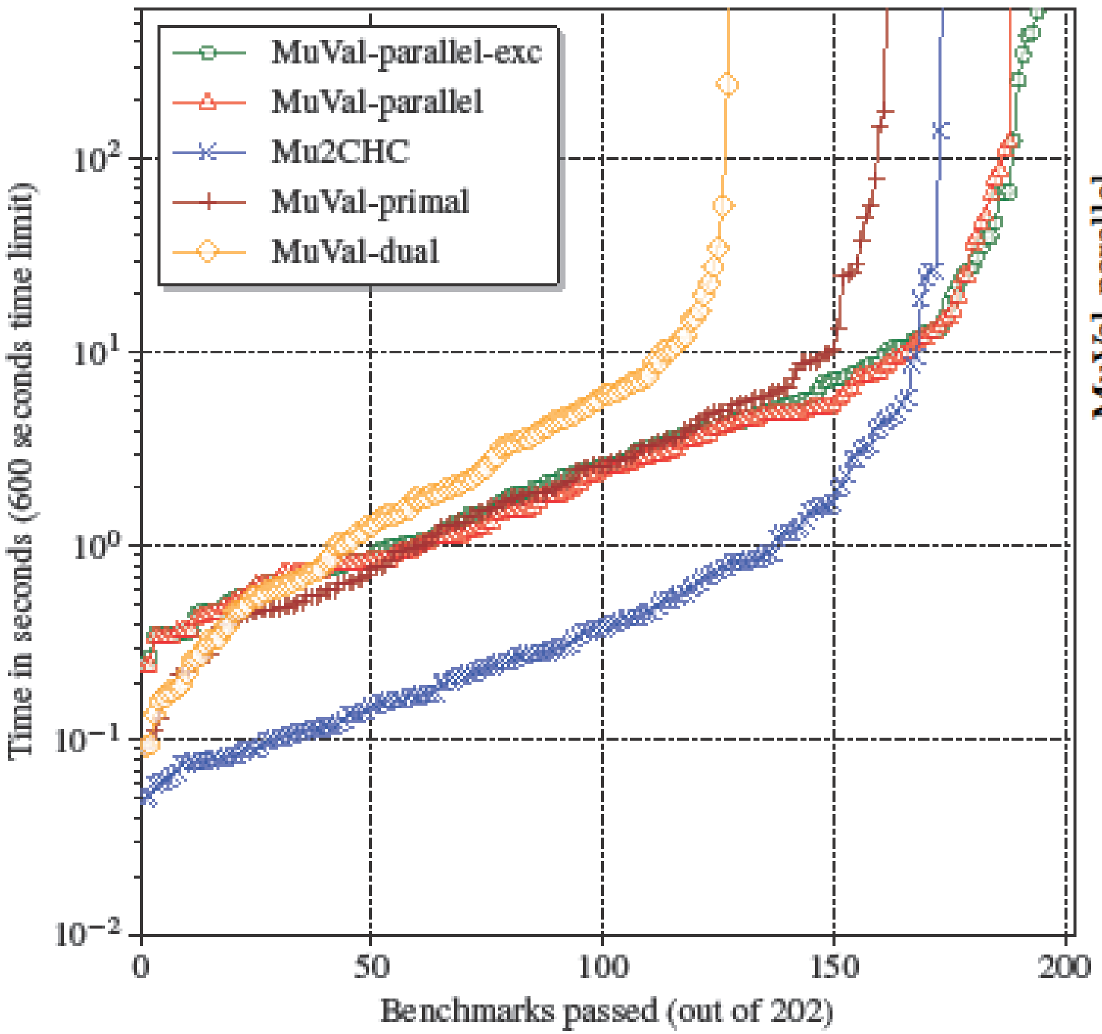
- MuVal-parallel: **MuVal** with primal-dual parallel solving (but without exchange of learned upper-bounds)
- AProVE: The winner of the C Integer track in 2018, 2020, and 2021
- iRankFinder [Ben-Amram&Genaim'14]
- UltimateAutomizer [Heizmann+'14]

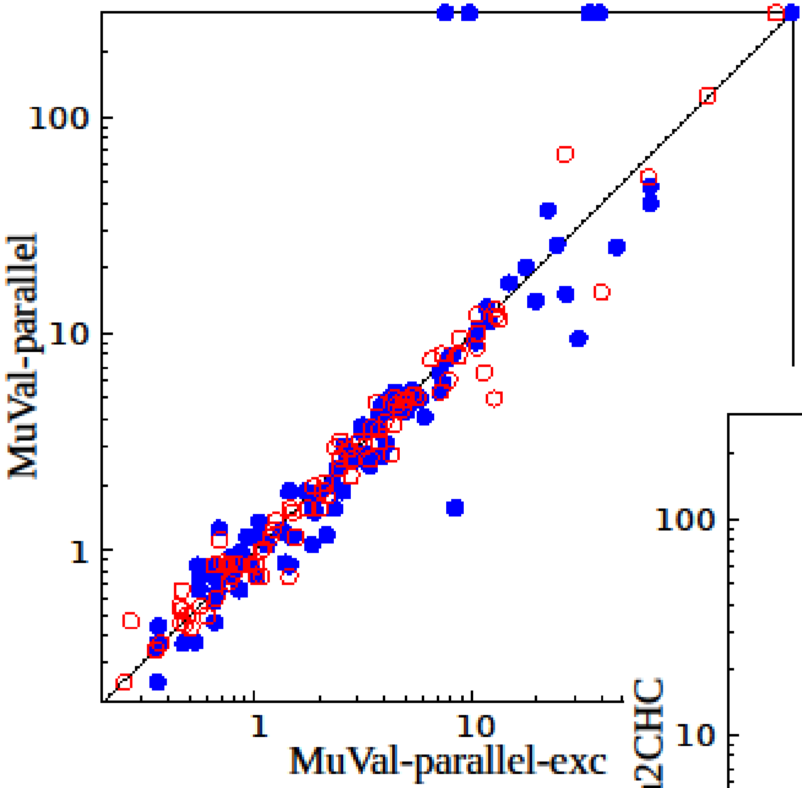
Evaluation with TermComp Benchmarks

- Mu2CHC: A **MuArith** validity checker based on a reduction to **CHCs** [Kobayashi+'19]
- MuVal-parallel-exc: MuVal-parallel + exchange of learned upper-bounds

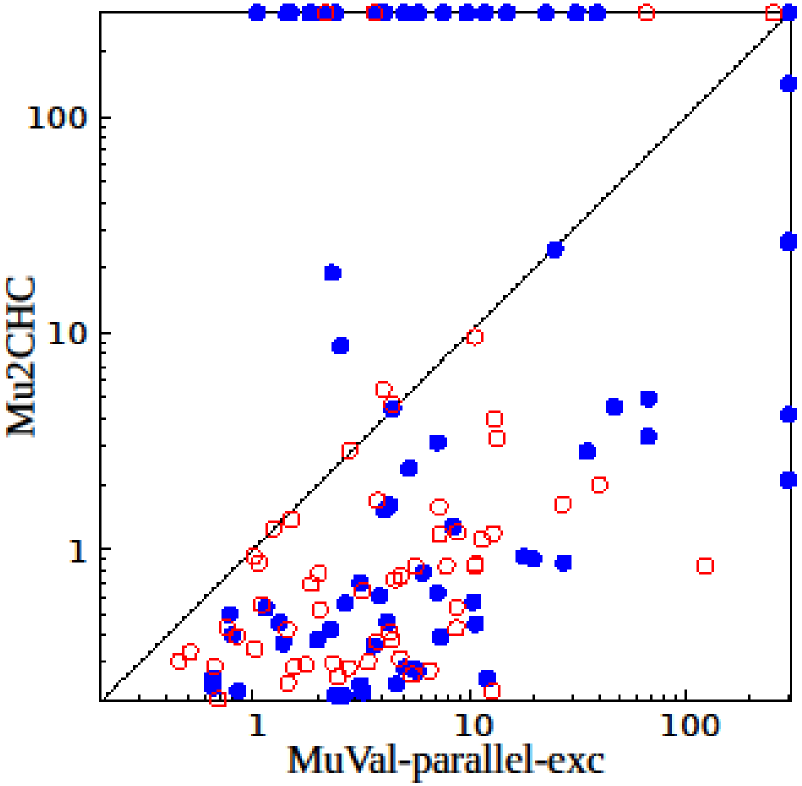


Evaluation with Temporal Verification Benchmarks





The paper also presents a comparison with UltimateLTLAutomizer [Dietsch+‘15]



Summary

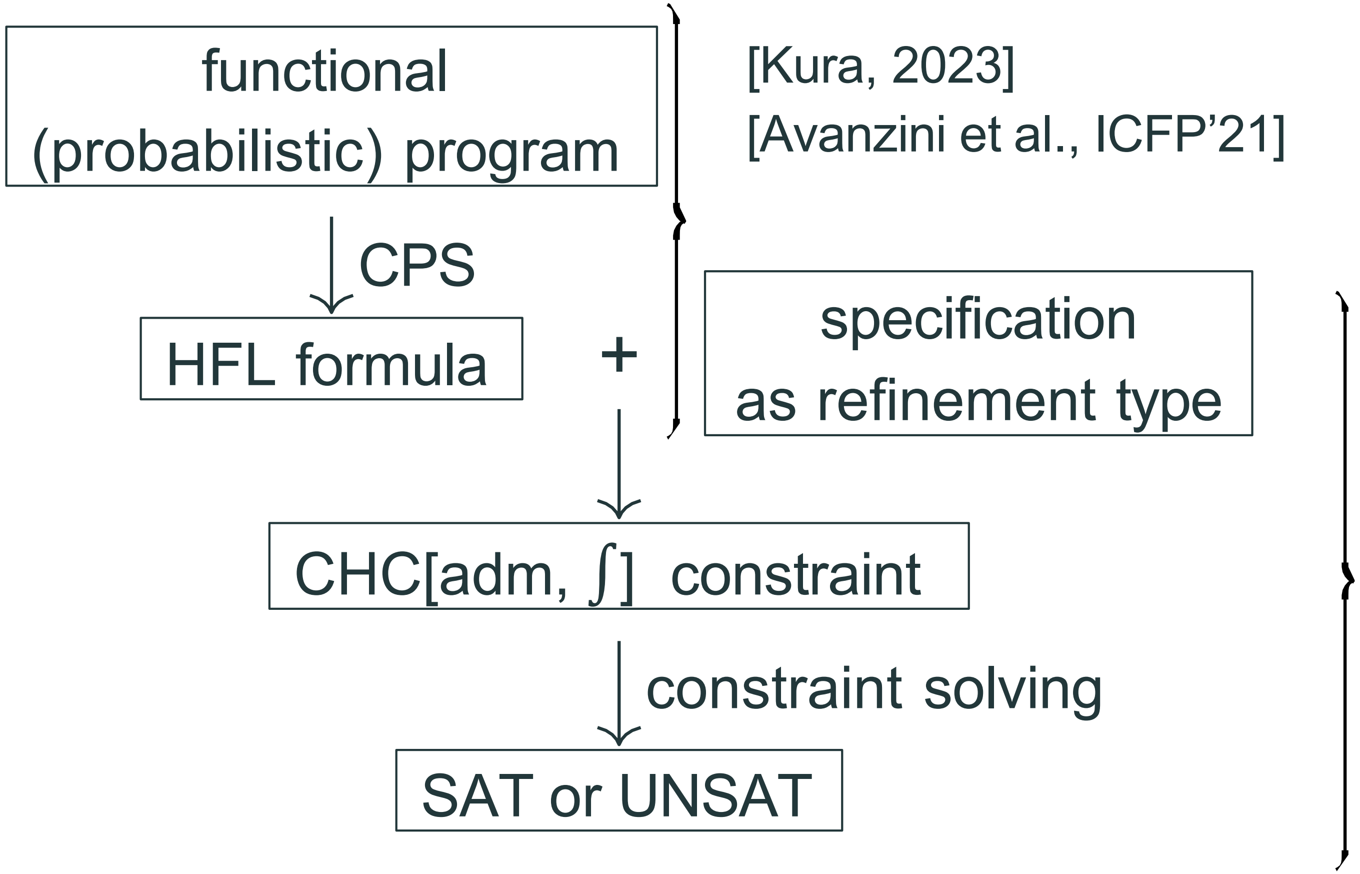
- **μCLP**: A first-order fixpoint logic modulo background theories
- **MuVal**: A ***modular primal-dual*** method for checking ***μ*CLP** validity
  - Reduce ***μ*CLP** validity to **pfwCSP** satisfiability
  - Solve the ***primal*** **pfwCSP** and the ***dual*** **pfwCSP** in parallel
    by exchanging each others' ***bounds*** to reduce each others' solution spaces
- Implementation and evaluation with a wide variety of temporal verification problems
  - Obtained competitive results to the state-of-the-art tools:
    AProVE and UltimateLTLAutomizer

Outline

- Classes of predicate constraint solving problems
- Reduction from validity checking for Mu-Arithmetic and μCLP [POPL 2023mod]
- **Reduction from validity checking for the quantitative variant of HFL [ICFP 2024]**
- CounterExample Guided Inductive Synthesis (CEGIS)
  for predicate constraint solving [AAAI 2020, CAV 2021rel, CAV 2021dt, ICFP 2024]

[POPL 2023mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.
[AAAI 2020] Satake et al. Probabilistic Inference for Predicate Constraint Satisfaction.
[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[POPL 2023opt] Gu et al. Optimal CHC Solving via Termination Proofs.

Overview of Our HFL-based Framework [ICFP 2024]

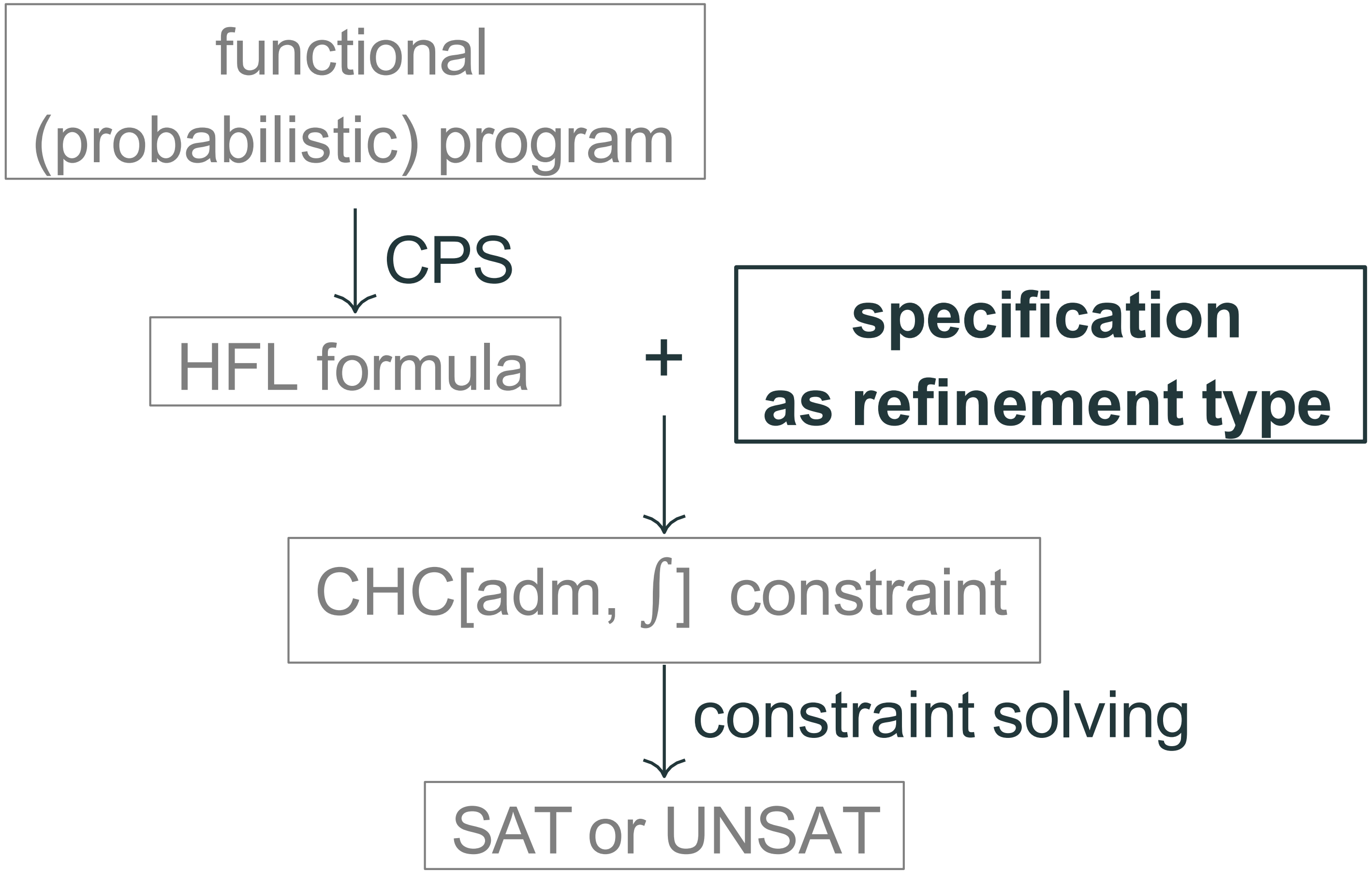


Our contributions:

1. Refinement type system for HFL
2. Implementation of type checking and inference

CPS = Continuation-Passing Style, HFL = (generalized) Higher-order Fixed-point Logic

Specification



Expected Cost Analysis via CPS Transformation

functional (probabilistic) program

$\downarrow$ CPS

HFL formula

$\mathbf{rw} : \mathbf{real} \rightarrow \mathbf{unit}$

$\mathbf{let\ rec\ rw}\ x = \ \mathbf{if}\ x \geq 0$

$\quad \mathbf{then}\ y \leftarrow \mathbf{uniform}_{[0,1]}; (\mathbf{rw}\ (x + \ 3 \cdot y - 2))^{\checkmark}$

$\quad \mathbf{else}\ \ ()$

$\mathbf{rw}' : \mathbf{real} \rightarrow (\mathbf{unit} \rightarrow [0,\infty]) \rightarrow [0,\infty]$

$\mathbf{let\ fix\ rw}'\ x\ k = \ \mathbf{if}\ x \geq 0$

$\quad \mathbf{then\ unif}(\lambda y.1 + \ \mathbf{rw}'(x + \ 3 \cdot y - 2)\ k)$

$\quad \mathbf{else}\ k\ ()$

(expected cost of $\mathbf{rw}\ x$) $=$ $\mathbf{rw}'\ x(\lambda r.0)$ [Avanzini et al., ICFP’21]

Specification

HFL formula + specification as refinement type

Specification: “(the expected cost of $\mathbf{rw}\ 1) \leq 6$.”

$\shortparallel$

“If $x = 1$ and $k = \lambda r.0$, then $\mathbf{rw}'\ x\ k \leq 6$.” ($\mathbf{rw}'$ = CPSed $\mathbf{rw}$)

$\shortparallel$

Refinement Type:

$$\mathbf{rw}' : \{x : \mathbf{real} \mid x = 1\} \to (\mathbf{unit} \to \{r : \mathbf{Prop} \mid r = 0\})$$
$$\to \{r : \mathbf{Prop} \mid r \leq 6\}$$

Type Checking and Inference

functional
(probabilistic) program

↓ CPS

HFL formula + **specification as refinement type**

↓

CHC[adm, ∫] constraint

↓ constraint solving

SAT or UNSAT

Type Checking and Inference

We extend the standard algorithm [Unno and Kobayashi, PPDP'09]

HFL formula + specification as refinement type

↓

CHC[adm, $\int$] constraint

1. Infer simple types.
2. Generate templates for refinement types.
3. Generate CHC constraints using typing rules.

Step 1: Inferring Simple Types

Example

$$(\lambda x. x + 1)\ 42 \quad : \quad \{ y : \mathbf{int} \mid y \geq 0 \}$$

Apply the Hindley–Milner type inference algorithm to obtain

$$\mathbf{sty} : \text{(subterms)} \to \text{(simple types)}$$

For example,

$$\lambda x. x + 1 : \mathbf{int} \to \mathbf{int}, \qquad 42 : \mathbf{int}, \qquad \ldots$$

Step 2: Generating Refinement Type Templates

Replace simple types with refinement type templates

For example,

$$\lambda x.x + 1 : (x : \{x : \mathtt{int} \mid P_1(x)\}) \rightarrow \{y : \mathtt{int} \mid P_2(x, y)\}$$

where P_1 and P_2 are fresh predicate variables

Step 3: Generating CHC Constraints

$$(\lambda x.x + 1)\ 42 \quad : \quad \{y : \mathtt{int} \mid y \geq 0\}$$

is well-typed if

$$P_1(x) \Rightarrow P_2(x, x + 1)$$

pre-/post-condition of $\lambda x.x + 1$

$$x = 42 \Rightarrow P_1(x)$$

argument of the function application

$$P_2(42, y) \Rightarrow y \geq 0$$

return value of the function application

is satisfiable.

$$\lambda x.x + 1 \quad : \quad (x : \{x : \mathtt{int} \mid P_1(x)\}) \rightarrow \{y : \mathtt{int} \mid P_2(x, y)\}$$

Refinement Type System for HFL: Theory

- A uniform framework applicable to
  - **effectful programs** in general
    e.g. probabilistic programs,
  - specifications described by the **generic weakest precondition**
    e.g. expected cost, cost moment, ...
- Soundness theorem using category theory [Kura, FoSSaCS'21].
- Key typing rules:
  - Fixed points ($\neq$ recursion)
  - Integration operators

Typing Rule for Fixed Points

We use the Scott induction

Admissibility is explicitly required because **`Prop`** is a non-flat domain

$$\frac{\dot{\Gamma}, f : (x : \dot{\sigma}) \to \{\, v : \mathtt{Prop} \mid \phi\} \vdash M \ : (x : \dot{\sigma}) \to \{\, v : \mathtt{Prop} \mid \phi\} \qquad \phi \text{ is admissible w.r.t. } v}{\dot{\Gamma} \ \vdash \mathrm{fix}\ f.\, M \ : (x : \dot{\sigma}) \to \{\, v : \mathtt{Prop} \mid \phi\}}$$

Definition

A subset $A \subseteq X$ of an ωcpo X is **admissible** if

$\bot \in A$ For any ω-chain $x_0 \leq x_1 \leq \cdots, (\forall i.\, x_i \in A) \Rightarrow \sup_i x_i \in A$

Fixed Points in HFL (1/3)

Program: coin flip

$$\text{let rec coin } x = \text{ if bern}(1/2) \text{ then } (\text{coin } ())^{\checkmark} \text{ else } ()$$

Expected cost (CPS):

$$\text{let fix coin}' \; x \, k = 1/2 \cdot (1 + \text{coin}' \; () \; k + k \; ()) \text{ in coin}' \; () \; (\lambda r.0)$$

Expected cost (simplified):

$$\text{let fix coin}'' = 1/2 \cdot (1 + \text{coin}'') \text{ in coin}''$$

This is the lfp of $F(c) := 1/2 \cdot (1 + c)$ w.r.t. $([0, \infty], \leq)$

Fixed Points in HFL (2/3)

The least fixed point is 1

$$
\begin{aligned}
0.5 \cdot (1 + 0) &= 0.5 \\
0.5 \cdot (1 + 0.5) &= 0.75 \\
0.5 \cdot (1 + 0.75) &= 0.875 \\
&\vdots \\
0.5 \cdot (1 + 1) &= 1
\end{aligned}
$$

Fixed Points in HFL (3/3)

We have

$$\mathrm{coin}'' : \{r : \mathrm{Prop} \mid r < 1\} \vdash 1/2 \cdot (1 + \mathrm{coin}'') : \{r : Prop \mid r < 1\}$$

but we can reject

$$\vdash \mathrm{fix}\ \mathrm{coin}''.\, 1/2 \cdot (1 + \mathrm{coin}'') : \{r : Prop \mid r < 1\}$$

because $r < 1$ is not admissible (not closed under $\mathbf{sup}$)

Typing Rule for Integration Operators

Our HFL has **integration operators** to reason about **continuous distributions** (e.g. uniform distribution)

$$\mathbf{unif} : (\mathbf{real} \to \mathbf{Prop}) \to \mathbf{Prop} \qquad\qquad (= \lambda f.\int_0^1 f\,x\ \mathrm{d}x)$$

The following rule can reason about upper bounds.

$$\frac{\Gamma^{\cdot} \vdash M : (x : \{x : \mathbf{real} \mid 0 \le x \le 1\}) \to \{v : \mathbf{Prop} \mid v \le N\ x\}}{\Gamma^{\cdot} \vdash \mathbf{unif}(M) : \{v : \mathbf{Prop} \mid v \le \mathbf{unif}(N)\}}$$

N should be simple so that we can easily compute $\mathbf{unif}(N)$

Refinement Type System for HFL: Implementation

We extend RCaml(type checking and inference) and PCSat¶

λ-calculus & refinement type — RCaml — CHC constraint — PCSat — SAT / UNSAT

- fixed points
- integration

generalized HFL & refinement type

- **admissible** predicate variables
- **integrable** predicate variables

CHC[adm, ∫] constraint

¶Available from https://github.com/hiroshi-unno/coar

Experimental Results

Problem	Benchmark	Time (sec)
Weakest pre-expectation	lics16_rec3	timeout
	lics16_rec3_ghost	1.270
	lics16_coins	3.110
Expected cost analysis	random_walk	2.761
	random_walk_unif	7.508
	coin_flip	0.718
	coin_flip_unif	0.884
	icfp21_walk	3.532
	icfp21_coupons	timeout
	lics16_fact	3.383
Cost moment analysis	coin_flip_ord2	1.135
	coin_flip_ord3	4.040
Conditional weakest pre-expectation	toplas18_ex4.4	timeout
	two_coin_conditioning	1.079

Invariant Synthesis

Benchmark: random_walk_unif

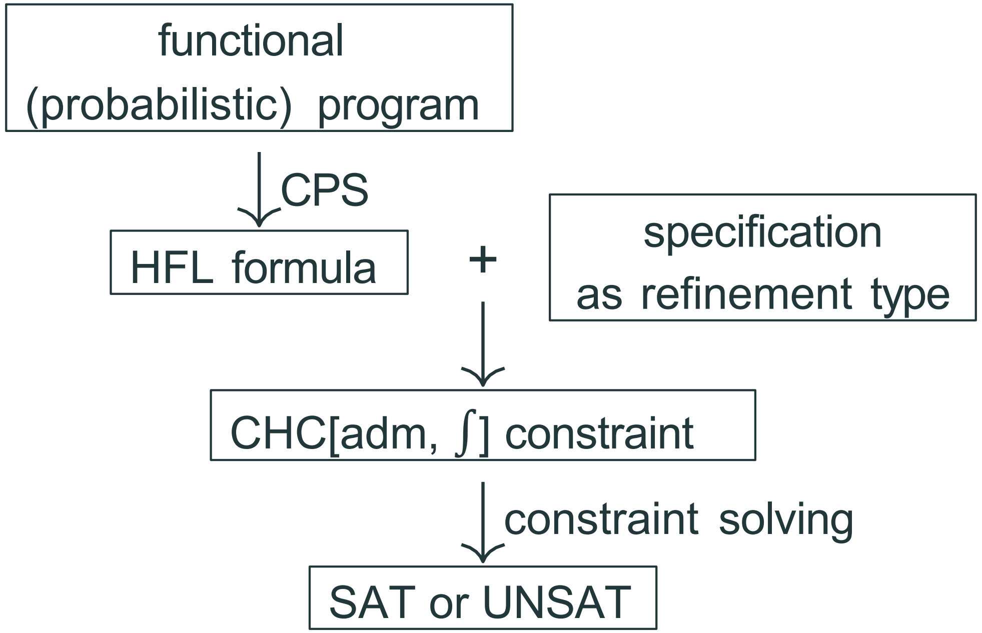
Program:

$$\begin{aligned}&\texttt{let rec rw } x = \texttt{ if } x \geq 0\\ &\quad \texttt{then } y \leftarrow \texttt{uniform}_{[0,1]};\\ &\quad\quad (\texttt{rw } (x + 3 \cdot y - 2))^{\checkmark}\\ &\quad \texttt{else } ()\end{aligned}$$

RCaml inferred an invariant automatically:

$$\begin{aligned}\texttt{rw}' : \{x : \texttt{real} \mid \texttt{true}\} &\rightarrow (\texttt{unit} \rightarrow \{r : \texttt{Prop} \mid r = 0\})\\ &\rightarrow \{r : \texttt{Prop} \mid r \leq |2x + 4|\}\end{aligned}$$

Summary



Our tool:

- We proposed a **uniform verification framework**
- We used existing results about **CPS ≅ WP**
- We proposed a refinement type system for the (generalized) HFL
- We extended the class of CHC and a CHC solver
- We implemented a type checker

Outline

- Classes of predicate constraint solving problems
- Reduction from validity checking for Mu-Arithmetic and μCLP [POPL 2023mod]
- Reduction from validity checking for the quantitative variant of HFL [ICFP 2024]
- **CounterExample Guided Inductive Synthesis (CEGIS) for predicate constraint solving [AAAI 2020, CAV 2021rel, CAV 2021dt, ICFP 2024]**

[POPL 2023mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[ICFP 2024] Kura and Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.
[AAAI 2020] Satake et al. Probabilistic Inference for Predicate Constraint Satisfaction.
[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[POPL 2023opt] Gu et al. Optimal CHC Solving via Termination Proofs.

Challenges in Predicate Constraint Solving

- Undecidable in general even for decidable theories
- The search space of solutions is often very large (or unbounded), high-dimensional, and non-smooth

To address these challenges, we integrate ***deductive & inductive reasoning*** techniques within the framework of ***CounterExample Guided Inductive Synthesis (CEGIS)*** [ASPLOS 2006], with an expectation that ***a general solution form*** can be ***inductively learned*** and then the ***details*** are ***deductively completed***

[ASPLOS 2006] Solar-Lezama et al. Combinatorial Sketching for Finite Programs.

CounterExample Guided Inductive Synthesis (CEGIS) [ASPLOS 2006]

- Iteratively accumulate example instances $\mathcal{E}$ of the given $\mathcal{C}$ through the two phases for each iteration:
  - ***Synthesis Phase*** *by* ***Learner***
    - Find a candidate solution ρ that satisfies $\mathcal{E}$
  - ***Validation Phase*** *by* ***Teacher***
    - Check if the candidate ρ also satisfies $\mathcal{C}$ (with an SMT solver)
      - If yes, return ρ as a genuine solution of $\mathcal{C}$
      - If no, repeat the procedure with new example instances witnessing non-satisfaction of $\mathcal{C}$ by ρ (i.e., counterexamples) added

[ASPLOS 2006] Solar-Lezama et al. Combinatorial Sketching for Finite Programs.

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$\emptyset$

Starting from the empty set ($\mathcal{C}$ is a black b

Teacher

Constraints $\mathcal{C}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \vee X(x+1) \Leftarrow X(x) \wedge x \neq 0$
- $\bot \Leftarrow X(x) \wedge x = 0$

Is the candidate $\{X(x) \mapsto \top\}$ genuine?

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$$\bot \Leftarrow X(0) \land 0 = 0$$

Teacher

Constraints $\mathcal{C}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \lor X(x+1)$
  $\Leftarrow X(x) \land x \neq 0$
- $\bot \Leftarrow X(x) \land x = 0$

No. $\{X(x) \mapsto \mathrm{T}\}$ is not.
The 3rd clause is violated when $x = 0$

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$\neg X(0)$

Teacher

Constraints $\mathcal{C}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \vee X(x+1)$
  $\Leftarrow X(x) \wedge x \neq 0$
- $\bot \Leftarrow X(x) \wedge x = 0$

Is the cand. $\{X(x) \mapsto x < 0\}$ genuine?

Example Run of CEGIS

Learner

Example Instances $\boldsymbol{\mathcal{E}}$:

$\neg X(0)$

$X(1) \Leftarrow 1 > 0$

Teacher

Constraints $\boldsymbol{\mathcal{C}}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \lor X(x+1) \Leftarrow X(x) \land x \neq 0$
- $\bot \Leftarrow X(x) \land x = 0$

No. $\{X(x) \mapsto x < 0\}$ is not.
The 1st clause is violated when $x = 1$

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$\neg X(0)$

$X(1)$

Teacher

Constraints $\mathcal{C}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \lor X(x+1) \Leftarrow X(x) \land x \neq 0$
- $\bot \Leftarrow X(x) \land x = 0$

Is the cand. $\{X(x) \mapsto x = 1\}$ genuine?

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$$\neg X(0)$$
$$X(0) \lor X(2) \Leftarrow X(1)$$
$$X(1)$$

Teacher

Constraints $\mathcal{C}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \lor X(x+1) \Leftarrow X(x) \land x \neq 0$
- $\bot \Leftarrow X(x) \land x = 0$

No. $\{X(x) \mapsto x = 1\}$ is not.
The 2nd clause is violated when $x = 1$

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$$\neg X(0)$$
$$X(0) \lor X(2) \Leftarrow X(1)$$
$$X(1)$$

Teacher

Constraints $\mathcal{C}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \lor X(x+1)$
  $\Leftarrow X(x) \land x \neq 0$
- $\bot \Leftarrow X(x) \land x = 0$

Is the cand. $\{X(x) \mapsto x \geq 1\}$ genuine?

Example Run of CEGIS

Learner

Example Instances $\boldsymbol{\mathcal{E}}$:

$$\neg X(0)$$
$$X(0) \vee X(2) \Leftarrow X(1)$$
$$X(1)$$

Teacher

Constraints $\boldsymbol{\mathcal{C}}$:

- $X(x) \Leftarrow x > 0$
- $X(x-1) \vee X(x+1)$
  $\Leftarrow X(x) \wedge x \neq 0$
- $\bot \Leftarrow X(x) \wedge x = 0$

Yes. $\{X(x) \mapsto x \geq 1\}$ satisfies $\boldsymbol{\mathcal{C}}$!

CEGIS vs. Online Supervised Learning

- Similarities
  - **Learner** trains a model on the examples $\mathcal{E}$ to obtain ρ
  - ρ is required to generalize to $\mathcal{C}$ (ρ shouldn't overfit $\mathcal{E}$)
- Differences
  - $\mathcal{E}$ is usually assumed to have no noise & $\mathcal{C}$ is ***hard*** constraints
  - ρ is required to ***exactly*** satisfy $\mathcal{E}$ (or has no chance to satisfy $\mathcal{C}$)
  - ρ should be ***efficiently*** handled by **Teacher** (i.e., an SMT solver)
  - Sampling of $\mathcal{E}$ from $\mathcal{C}$ is not i.i.d (depends on ρ and **Teacher**)
  - $\mathcal{E}$ may contain not only positive/negative examples but also arbitrary clause ones (cf. weakly-supervised learning)

Despite the differences, machine learning techniques turned out to be quite useful!

Machine Learning for CEGIS

- Adapt ML models and learning algorithms to implement **Learner**
  - Piecewise affine classifiers (or templates) [Garg+'14, CAV 2021rel]
  - Decision trees [Saha+'15, Garg+'16, Champion+'18, Ezudheen+'18, Zhu+'18, CAV 2021dt]
  - Support vector machines [Sharma+'12, Zhu+'18, CAV 2021dt]
  - Neural networks [Chang+'19, Zhao+'20, Ryan'20, Abate+'21, SAS 2021]
  - Geometric concept learning [Sharma+'13, Padhi+'16]
  - Graphical models and probabilistic inference
    - Metropolis Hastings MCMC sampler [Sharma+'14]
    - Survey propagation [AAAI 2020]
  - Reinforcement learning [Si'18, arXiv 2021]
  - Ensemble learning [Padhi+'20]

Our solver **PCSat** is based on the blue-highlighted references

Template-based Synthesis

1. Prepare a solution template with unknown coefficients,
2. Generate constraints on them, and
3. Solve them using an **SMT solver**

Examples: $\mathcal{E} = \{X(0), (X(0) \Rightarrow X(1)), \neg X(-1)\}$

Solution Template: $X(x) \mapsto c_1 \cdot x + c_2 \geq 0$

Coeff. Constraints: $\{c_2 \geq 0, (c_2 \geq 0 \Rightarrow c_1 + c_2 \geq 0), -c_1 + c_2 < 0\}$

Satisfying Assignment: $\{c_1 \mapsto 1, c_2 \mapsto 0\}$

A Candidate Solution: $\rho = \{X(x) \mapsto x \geq 0\}$

Templates used in Synthesis

- Design predicate templates to ensure they characterize well-founded relations, total functions, admissible predicates, and integrable predicates, satisfying their respective definitional conditions
  - For well-founded predicates, lexicographic piecewise affine ranking function templates are used
  - For ordinary, functional, admissible, and integrable predicates, piecewise affine (in)equality templates are used but their form is restricted to satisfy their respective definitional conditions

Stratified Template Families

- Our method combines CEGIS with ***stratified template families*** of ***ordinary/functional/well-founded/admissible/integrable*** predicates
  - Search for solutions in a stratified manner: Starting from simple templates, iteratively update them to expressive ones, if needed, according to the family
  - To achieve not only ***efficiency*** but also ***relatively completeness***
  - The ***data-driven nature*** of CEGIS is a good match: the stratified search accelerates the convergence by ***avoiding the overfitting problem*** [Padhi+ '19] of expressive templates to counterexamples

Decision Tree Learning of Ordinary Predicates

1. Consistently label atoms in $\mathcal{E}$ with $+/-$ using a **SAT solver**
2. Generate a set Q of predicates used in classification
3. Classify atoms in $\mathcal{E}$ with Q using a decision tree learner

Examples: $\boldsymbol{\mathcal{E}} = \{X(0), \big(X(0) \Rightarrow X(1)\big), \neg X(-1)\}$

Labeling: $\{X(0) \mapsto +, X(1) \mapsto +, X(-1) \mapsto -\}$

Predicates: $Q = \begin{Bmatrix} x \geq 0, x \leq 0, x \geq 1, \\ x \geq -1, x \leq 1, x \leq -1 \end{Bmatrix}$

Classifier: $\rho = \{X(x) \mapsto x \geq 0\}$

Decision Tree Learning of Other Predicates

- Dedicated and sophisticated techniques are required for
  - Well-founded predicates [CAV 2021dt]
  - Functional predicates [CAV 2015, TACAS 2016]

[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[CAV 2015] Saha et al. Learning Guarded Affine Functions.
[TACAS 2016] Neider et al. Synthesizing Piece-Wise Functions by Learning Classifiers.

Template-based Synthesis vs. Decision Tree Learning

- Template-based Synthesis (TB)
  - ☹ Fixes the ***shape*** of solution (updated upon failure)
  - ☺ Flexibly find necessary ***predicates*** via SMT solving
  - ☺ Atoms in $\mathcal{E}$ are consistently ***labeled*** using $\mathcal{E}$ as an SMT formula
- Decision Tree Learning (DT)
  - ☹ Fixes the ***predicates*** of solution (updated upon failure)
  - ☺ Flexibly adjust the ***shape*** by heuristics based on information gain
  - ☹ Atoms in $\mathcal{E}$ are consistently ***labeled*** using $\mathcal{E}$ as a SAT formula
    - The information about the arguments of predicate variables is not sufficiently utilized

Results of CHC-COMP 2025

LIA-Lin	LIA	LIA-Lin-Arrays**	LIA-Arrays	ADT-LIA	ADT-LIA-Arrays	BV	LRA-Lin
Golem	Golem	Eldarica	Eldarica	Catalia	Eldarica	Eldarica	Golem
MuCyc	Eldarica	Unihorn	PCSat	Eldarica	PCSat	Theta	Eldarica
LoAT	PCSat	PCSat	Unihorn	PCSat	--	PCSat	Theta

- Cited from: https://chc-comp.github.io/2025/CHC-COMP%202025%20Report%20-%20SPIN.pdf

Summary

- CEGIS-based predicate constraint solving methods integrate ***deductive*** and ***inductive*** reasoning for efficiency
  - ***Deductive*** reasoning by ***theorem proving*** (e.g., SAT, SMT)
  - ***Inductive*** reasoning by ***machine learning*** (e.g., decision tree learning)
- Our **pfwCSP** and **CHC[adm, ∫]** solver **PCSat** is available from: https://github.com/hiroshi-unno/coar

Course Schedule

- Wed. 21 May (8:50-10:30)
  1. Reduction from software verification to fixed-point logic validity checking
  2. Predicate constraint solving for validity checking
- Thu. 22 May (11:20-12:20)
  3. **Cyclic-proof search for validity checking**
  4. Game solving for validity checking

3. Cyclic-Proof Search for Validity Checking

Outline

- Software model-checking (that is an instance of μCLP validity checking) as cyclic-proof search [POPL 2022, PLDI 2024]
  - Various existing software model-checking techniques can be interpreted as different strategies for proof search in a cyclic proof system
- Relational verification (that is another instance of μCLP validity checking) as cyclic-proof search [CAV 2017]
  - New and powerful approach to relational verification

[POPL 2022] Tsukada and Unno. Software Model-Checking as Cyclic-Proof Search.
[PLDI 2024] Tsukada and Unno. Inductive Approach to Spacer.
[CAV 2017] Unno et al. Automating Induction for Solving Horn Clauses.

Outline

- **Software model-checking (that is an instance of μCLP validity checking) as cyclic-proof search [POPL 2022, PLDI 2024]**
  - **Various existing software model-checking techniques can be interpreted as different strategies for proof search in a cyclic proof system**
- Relational verification (that is another instance of μCLP validity checking) as cyclic-proof search [CAV 2017]
  - New and powerful approach to relational verification

[POPL 2022] Tsukada and Unno. Software Model-Checking as Cyclic-Proof Search.
[PLDI 2024] Tsukada and Unno. Inductive Approach to Spacer.
[CAV 2017] Unno et al. Automating Induction for Solving Horn Clauses.

This work

A precise connection between **software model checking** and **cyclic proof search**

[Ball+ 2001] [Henzinger+ 2002, 2004]
[McMillan 2006] [Cimatti&Griggio 2012]
[Hoder&Bjørner 2012] [Cimatti+ 2014]
[Birgmeier+ 2014] [Komuravelli+ 2013, 2014] …

[Brotherston and Simpson 2011]
[Sprenger and Dam 2003] …

Known **Software model-checking problem**
$\longleftrightarrow$ **CLP validity / CHC satisfiability problem**

New

Software model-checking algorithms

= **proof search heuristics**

(**Internal states of algorithms** = **partially constructed proofs**)

Our aim from the viewpoint of software model-checking

Providing a **unified account for model-checking algorithms** in terms of **logic**

- To **understand** behaviors of many algorithms from simple **declarative principles** based on a **single common structure**

  partially constructed proofs

- To **compare** different algorithms
  - Property-directed reachability (PDR) ≈ Efficient game solving algorithm

    [Bradley 2011] [Een+ 2011] [Cimatti&Griggio 2012] — [Farzan&Kincaid 2017]

- To **develop** new algorithms
  - Refutationally complete variant of PDR

Our aim from the viewpoint of cyclic proof search

Importing ideas and techniques of software model-checking to cyclic proof search

- **Finding an appropriate cut formula is crucial** for cyclic proof search
  - **Cut-elimination fails** for cyclic proof systems [Kimura+ 2020] [Oda+ 2025] …

- Software model-checking community has developed
  highly-efficient algorithms to find an appropriate cut formula
  - Existing proof search strategies for cyclic proof system ≈ bounded model-checking + covering
    E.g. [Brotherston+ 2011] [Chu+ 2015] [Ta+ 2016] for entailment checking in separation logic
    [Unno+ 2017] for relational verification
    [Tellez&Brotherston 2020] for temporal verification
    [Itzhaky+ 2021] for program synthesis

Outline

- **Background**
  - **Software model-checking**
  - Proof systems for inductive definitions
- Key observation
- Software model-checking as cyclic proof search

Software model-checking

Algorithmic analysis of programs to prove properties of their executions

[Jhala&Majumdar 2009]

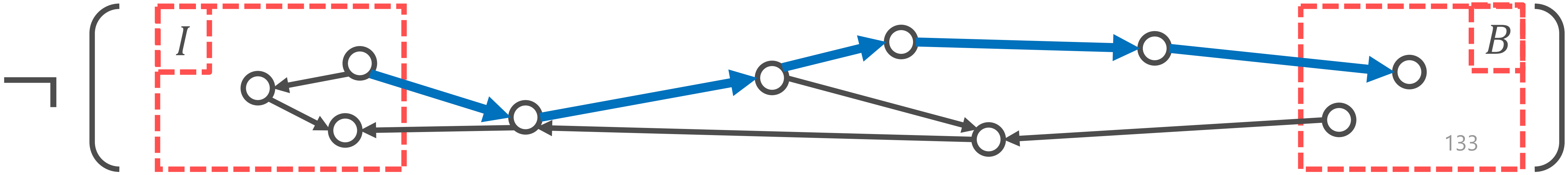
Let us focus on **safety verification of a while program**

Input	Set of **states**	D
	Initial states	$I \subseteq D$
	Bad states	$B \subseteq D$
	Transition relation	$T \subseteq D \times D$

Usually **infinite**, e.g. $D = \mathbb{Z}^n$

Output Whether B is **unreachable** from I via T

- $\neg \exists s_0 s_1 \dots s_n \in D . I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{n-1}, s_n) \wedge B(s_n)$



Inductive invariant

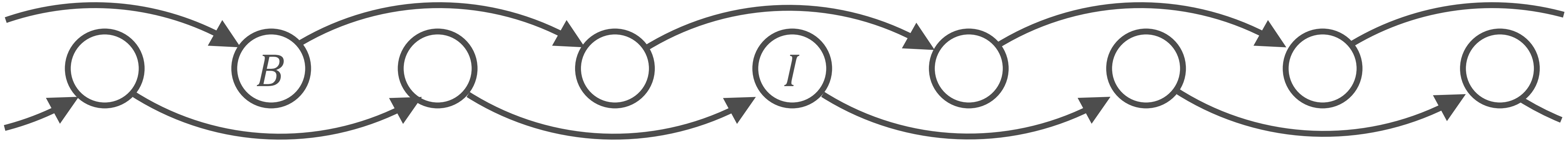
Set of states	D
Initial states	$I \subseteq D$
Bad states	$B \subseteq D$
Transition relation	$T \subseteq D \times D$

A witness of the safety of a given system

Def A subset $P \subseteq D$ is a (safe) **inductive invariant** if

- all initial states are P — $I(x) \Longrightarrow P(x)$
- P contains no bad state — $P(x) \Longrightarrow \neg B(x)$
- P is closed under the transition relation — $P(x) \wedge T(x,y) \Longrightarrow P(y)$

Example $D = \mathbb{Z}$, $I = \{0\}$, $B = \{-3\}$, $T = \{\, (n, n+2) \mid n \in \mathbb{Z} \,\}$



- $P_1 = \{\ 2n \mid n \in \mathbb{Z}, n \geq 0\ \}$ is an inductive invariant
- $P_2 = \{\ n \in \mathbb{Z} \mid n \geq 0\ \}$ is an inductive invariant

Inductive invariant

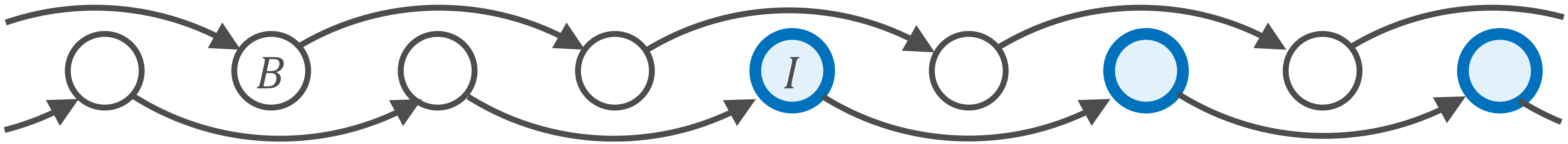
Set of states	D
Initial states	$I \subseteq D$
Bad states	$B \subseteq D$
Transition relation	$T \subseteq D \times D$

A witness of the safety of a given system

Def A subset $P \subseteq D$ is a (safe) **inductive invariant** if

- all initial states are P $\quad I(x) \Longrightarrow P(x)$
- P contains no bad state $\quad P(x) \Longrightarrow \neg B(x)$
- P is closed under the transition relation $\quad P(x) \wedge T(x,y) \Longrightarrow P(y)$

Example $\quad D = \mathbb{Z},\ I = \{0\},\ B = \{-3\},\ T = \{\ (n, n+2) \mid n \in \mathbb{Z}\ \}$



- $\boldsymbol{P_1 = \{\ 2n \mid n \in \mathbb{Z}, n \geq 0\ \}}$ **is an inductive invariant**
- $P_2 = \{\ n \in \mathbb{Z} \mid n \geq 0\ \}$ is an inductive invariant

Inductive invariant

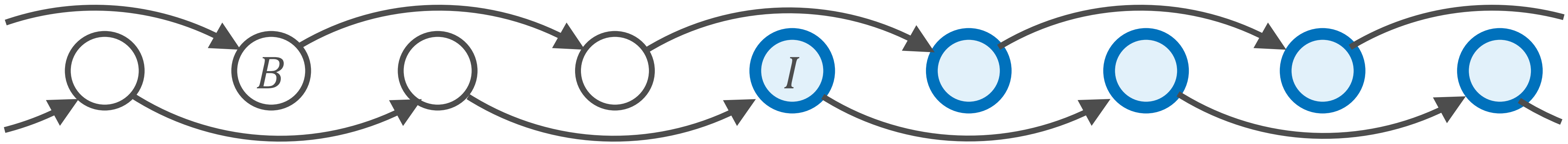
Set of states	D
Initial states	$I \subseteq D$
Bad states	$B \subseteq D$
Transition relation	$T \subseteq D \times D$

A witness of the safety of a given system

Def A subset $P \subseteq D$ is a (safe) **inductive invariant** if

- all initial states are P $\qquad I(x) \Longrightarrow P(x)$
- P contains no bad state $\qquad P(x) \Longrightarrow \neg B(x)$
- P is closed under the transition relation $\qquad P(x) \wedge T(x,y) \Longrightarrow P(y)$

Example $D = \mathbb{Z}$, $I = \{0\}$, $B = \{-3\}$, $T = \{\, (n, n+2) \mid n \in \mathbb{Z} \,\}$



- $P_1 = \{\ 2n \mid n \in \mathbb{Z}, n \geq 0\ \}$ is an inductive invariant
- $\boldsymbol{P_2 = \{\ n \in \mathbb{Z} \mid n \geq 0\ \}}$ **is an inductive invariant**

Inductive invariant

Set of states	D
Initial states	$I \subseteq D$
Bad states	$B \subseteq D$
Transition relation	$T \subseteq D \times D$

A witness of the safety of a given system

Def A subset $P \subseteq D$ is a (safe) **inductive invariant** if

- all initial states are P $\quad I(x) \Longrightarrow P(x)$
- P contains no bad state $\quad P(x) \Longrightarrow \neg B(x)$
- P is closed under the transition relation $\quad P(x) \wedge T(x, y) \Longrightarrow P(y)$

Prop If an inductive invariant $P \subseteq D$ exists, the system never reaches a bad state

Model-checkers search for inductive invariants in a variety of clever ways

- It is **relatively easy to check** if a given $P \subseteq D$ is indeed an inductive invariant

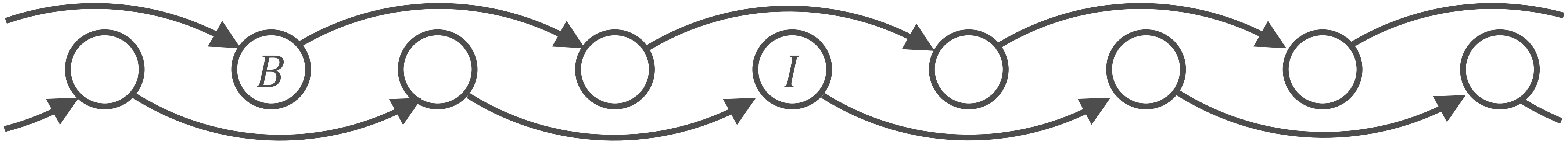
A Logical Formalization

The **set of reachable states** is the **least solution μR** for P in

$$P(x) \quad \overset{\mu}{\Longleftrightarrow} \quad I(x) \lor \big(\exists y. P(y) \land T(y,x)\big)$$

- Defining a property as the least solution of an equation = **inductive definition**

Example $D = \mathbb{Z}$, $I = \{0\}$, $T = \{\, (n, n+2) \mid n \in \mathbb{Z} \,\}$



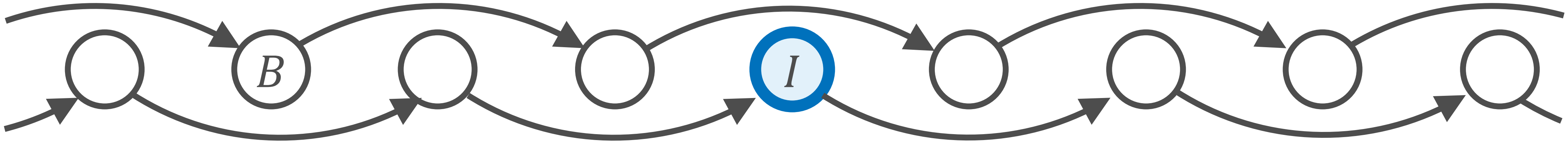
A Logical Formalization

The **set of reachable states** is the **least solution μR** for P in

$$P(x) \quad \overset{\mu}{\Longleftrightarrow} \quad I(x) \vee \big(\exists y. P(y) \wedge T(y,x)\big)$$

- Defining a property as the least solution of an equation = **inductive definition**

Example $D = \mathbb{Z}$, $I = \{0\}$, $T = \{\, (n, n+2) \mid n \in \mathbb{Z} \,\}$



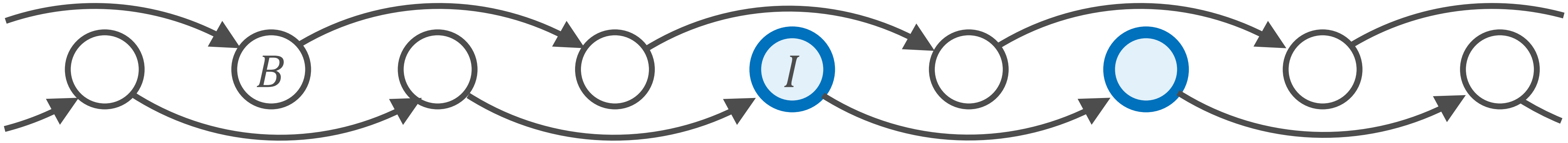
A Logical Formalization

The **set of reachable states** is the **least solution μR** for P in

$$P(x) \quad \stackrel{\mu}{\Longleftrightarrow} \quad I(x) \lor \big(\exists y. P(y) \land T(y,x)\big)$$

- Defining a property as the least solution of an equation = **inductive definition**

Example $D = \mathbb{Z}$, $I = \{0\}$, $T = \{\,(n, n+2) \mid n \in \mathbb{Z}\,\}$



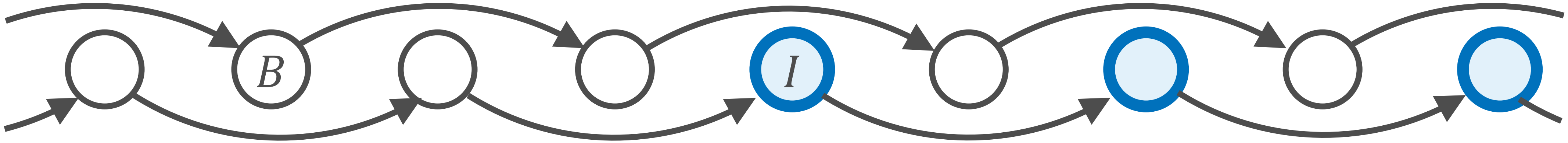
A Logical Formalization

The **set of reachable states** is the **least solution μR** for P in

$$P(x) \quad \overset{\mu}{\Longleftrightarrow} \quad I(x) \vee \big(\exists y. P(y) \wedge T(y,x)\big)$$

- Defining a property as the least solution of an equation = **inductive definition**

Example $D = \mathbb{Z}$, $I = \{0\}$, $T = \{\, (n, n+2) \mid n \in \mathbb{Z} \,\}$



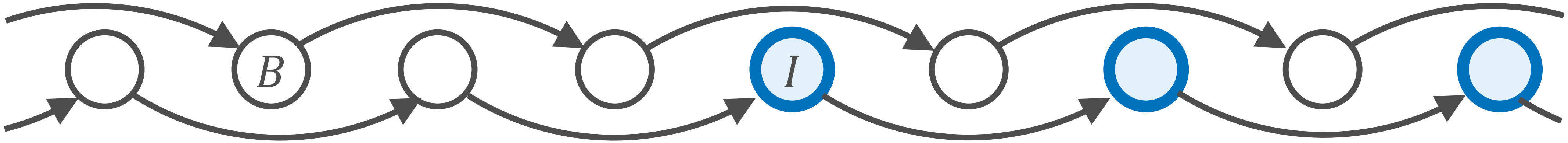
A Logical Formalization

The **set of reachable states** is the **least solution μR** for P in

$$P(x) \overset{\mu}{\iff} I(x) \lor \left(\exists y. P(y) \land T(y,x)\right)$$

- Defining a property as the least solution of an equation = **inductive definition**

<u>**Example**</u> $D = \mathbb{Z}$, $I = \{0\}$, $T = \{\, (n, n+2) \mid n \in \mathbb{Z} \,\}$



$$\mu R = \{2n \mid n \in \mathbb{Z}, n \geq 0\}$$

A Logical Formalization

The **set of reachable states** is the **least solution μR** for P in

$$P(x) \quad \overset{\mu}{\iff} \quad I(x) \lor \left(\exists y. P(y) \land T(y,x)\right)$$

- Defining a property as the least solution of an equation = **inductive definition**

Prop The system never reaches a bad state if and only if $\boldsymbol{\mu R(x)} \vDash \neg \boldsymbol{B(x)}$

- Simply because μR is the set of reachable states

Proof systems for inductive definitions are usable to prove $\mu R(x) \vdash \neg B(x)$

Outline

- **Background**
  - Software model-checking
  - **Proof systems for inductive definitions**
- Key observation
- Software model-checking as cyclic proof search

Classical proof rule for inductive definitions

Due to Martin-Löf (1972)

$$\boxed{\mu R(x) \overset{\mu}{\iff} I(x) \vee \big(\exists y.\mu R(y) \wedge T(y,x)\big)}$$

$$\frac{I(x) \vee (\exists y.\varphi(y) \wedge T(y,x)) \vdash \varphi(x) \qquad \varphi(x) \vdash \neg B(x)}{\mu R(x) \vdash \neg B(x)}$$

The premises require that **$\varphi(x)$ is an inductive invariant**

- Initial states satisfy φ — $I(x) \vdash \varphi(x)$
- φ is closed under the transition — $\exists y.\, \varphi(y) \wedge T(y,x) \vdash \varphi(x)$
- φ has no bad state — $\varphi(x) \vdash \neg B(x)$

This rule cannot be used to describe processes searching for inductive invariants

- This rule is **applicable only after an inductive invariant φ is found**

Cyclic proof system

[Brotherston&Simpson 2011]
[Sprenger&Dam 2003] …

A proof system in which **proofs may have cycles**

- **Cycle ≈ use of induction hypothesis**

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ I(x) \vdash \varphi(x)\end{array} \qquad \dfrac{\dfrac{\boxed{\mu R(y) \vdash \varphi(y)} \qquad \begin{array}{c}\vdots\\ \varphi(y) \vdash T(y,x) \Rightarrow \varphi(x)\end{array}}{\mu R(y) \vdash T(y,x) \Rightarrow \varphi(x)}}{\exists y.\mu R(y) \wedge T(y,x) \vdash \varphi(x)}}{\dfrac{I(x) \vee (\exists y.\mu R(y) \wedge T(y,x)) \vdash \varphi(x)}{\mu R(x) \vdash \varphi(x)}} \qquad \begin{array}{c}\vdots\\ \varphi(x) \vdash \neg B(x)\end{array}}{\mu R(x) \vdash \neg B(x)}$$

A rule for inductive definition just expands the definition

- **Applicable without knowing an inductive invariant**

$$\dfrac{I(x) \vee (\exists y.\mu R(y) \wedge T(y,x)) \vdash \varphi(x)}{\mu R(x) \vdash \varphi(x)}$$

$$\boxed{\mu R(x) \overset{\mu}{\Longleftrightarrow} I(x) \vee (\exists y.\mu R(y) \wedge T(y,x))}$$

Outline

- Background
  - Software model-checking
  - Proof systems for inductive definitions
- **Key observation**
- Software model-checking as cyclic proof search

Key observation

To establish a precise connection between model-checking and proof search,

- **"all reachable states are not bad"** is inappropriate,

$$\mu R(x) \vdash \neg B(x) \qquad\qquad \mu R(x) \overset{\mu}{\iff} I(x) \vee \big(\exists y. \mu R(y) \wedge T(y,x)\big)$$

  - A state x is **reachable** if $\exists y_0 y_1 \ldots y_{n-1}. I(y_0) \wedge T(y_0, y_1) \wedge \cdots \wedge T(y_{n-1}, x)$
    (cf. **strongest post-condition**, **backward reachability checking**)

- but the **dual** formalization **"all initial states are safe"** should be used

$$I(x) \vdash \nu S(x) \qquad\qquad \nu S(x) \overset{\nu}{\iff} \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow S(y)\big)$$

greatest solution

  - A state x is **safe** if $\neg\exists y_1 \ldots y_n. T(x, y_1) \wedge \cdots \wedge T(y_{n-1}, y_n) \wedge B(y_n)$
    (cf. **weakest pre-condition**, **forward reachability checking**)

Outline

- Background
  - Software model-checking
  - Proof systems for inductive definitions
- Key observation
- **Software model-checking as cyclic proof search**

Goal-oriented proof search

A bottom-up proof-search

- An intermediate state is a **proof with unproved leaves**

1. Start from the tree consisting only of the **goal sequent**

$$I(x) \overset{?}{\vdash} \nu S(x)$$

2. Choose an unproved leaf and **select an appropriate proof rule** for it

$$\frac{I(x) \overset{?}{\vdash} \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}{I(x) \vdash \nu S(x)}$$

3. Iterate this process until there are no unproved leaves

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$\frac{I(x) \overset{?}{\vdash} \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}{I(x) \vdash \nu S(x)}$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$\dfrac{\dfrac{I(x) \overset{?}{\vdash} \neg B(x) \qquad\qquad I(x) \overset{?}{\vdash} \forall y. T(x,y) \Rightarrow \nu S(y)}{I(x) \vdash \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}}{I(x) \vdash \nu S(x)}$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

An SMT solver can automatically (dis)prove

$$\dfrac{\dfrac{I(x) \overset{?}{\vdash} \neg B(x) \qquad\qquad I(x) \overset{?}{\vdash} \forall y. T(x,y) \Rightarrow \nu S(y)}{I(x) \vdash \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}}{I(x) \vdash \nu S(x)}$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$\dfrac{\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \overset{\textbf{?}}{I(x) \vdash \forall y. T(x,y) \Rightarrow \nu S(y)}}{I(x) \vdash \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}}{I(x) \vdash \nu S(x)}$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$
\dfrac{\dfrac{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\dfrac{?}{I(x) \vdash T(x,y) \Rightarrow \nu S(y)}}{I(x) \vdash \forall y. T(x,y) \Rightarrow \nu S(y)}}{\dfrac{I(x) \vdash \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}{I(x) \vdash \nu S(x)}}
$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$
\dfrac{
\overset{\text{SMT}}{I(x) \vdash \neg B(x)}
\qquad
\dfrac{\dfrac{I(x), T(x,y) \overset{?}{\vdash} \nu S(y)}{I(x) \vdash T(x,y) \Rightarrow \nu S(y)}}{I(x) \vdash \forall y. T(x,y) \Rightarrow \nu S(y)}
}{
\dfrac{I(x) \vdash \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}{I(x) \vdash \nu S(x)}
}
$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$\dfrac{\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\dfrac{\dfrac{\exists x. I(x) \wedge T(x,y) \overset{?}{\vdash} \nu S(y)}{\overline{I(x), T(x,y) \vdash \nu S(y)}}}{I(x) \vdash T(x,y) \Rightarrow \nu S(y)}}{I(x) \vdash \forall y. T(x,y) \Rightarrow \nu S(y)}}{I(x) \vdash \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)}}{I(x) \vdash \nu S(x)}$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

One-step transition

?

$$\dfrac{\begin{array}{c}\text{SMT}\\ I(x) \vdash \neg B(x)\end{array} \qquad \dfrac{\dfrac{\dfrac{\boxed{\exists x. I(x) \wedge T(x,y)} \vdash \nu S(y)}{I(x), T(x,y) \vdash \nu S(y)}}{I(x) \vdash T(x,y) \Rightarrow \nu S(y)}}{I(x) \vdash \forall y. T(x,y) \Rightarrow \nu S(y)}}{\dfrac{I(x) \vdash \neg B(x) \wedge \left(\forall y. T(x,y) \Rightarrow \nu S(y)\right)}{\boxed{I(x)} \vdash \nu S(x)}}$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Next states of *I* are safe

?

$$\boxed{\exists x. I(x) \wedge T(x,y)} \vdash \nu S(y)$$

$$I(x), T(x,y) \vdash \nu S(y)$$

$$I(x) \vdash T(x,y) \Rightarrow \nu S(y)$$

One-step transition

Initial states are not bad

SMT

$$I(x) \vdash \neg B(x) \qquad\qquad I(x) \vdash \forall y. T(x,y) \Rightarrow \nu S(y)$$

$$I(x) \vdash \neg B(x) \wedge \left(\forall y. T(x,y) \Rightarrow \nu S(y)\right)$$

$$\boxed{I(x)} \vdash \nu S(x)$$

Symbolic execution

<u>**Heuristic 1**</u> Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Next states of φ are safe

Current states are not bad

One-step transition

$$
\dfrac{
\dfrac{}{\varphi(x) \vdash \neg B(x)}\,\text{SMT}
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{\boxed{\exists x.\varphi(x) \wedge T(x,y)} \vdash \nu S(y)\ ?}{\varphi(x), T(x,y) \vdash \nu S(y)}
}{\varphi(x) \vdash T(x,y) \Rightarrow \nu S(y)}
}{\varphi(x) \vdash \forall y.T(x,y) \Rightarrow \nu S(y)}
}{}
}{
\dfrac{\varphi(x) \vdash \neg B(x) \wedge \left(\forall y.T(x,y) \Rightarrow \nu S(y)\right)}{\boxed{\varphi(x)} \vdash \nu S(x)}
}
$$

Generalize to arbitrary set φ

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$\dfrac{\dfrac{\overset{\text{SMT}}{\varphi(x) \vdash \neg B(x)} \qquad \dfrac{\dfrac{\dfrac{\exists x.\varphi(x) \land T(x,y) \overset{?}{\vdash} \nu S(y)}{\varphi(x), T(x,y) \vdash \nu S(y)}}{\varphi(x) \vdash T(x,y) \Rightarrow \nu S(y)}}{\varphi(x) \vdash \forall y.T(x,y) \Rightarrow \nu S(y)}}{\varphi(x) \vdash \neg B(x) \land \left(\forall y.T(x,y) \Rightarrow \nu S(y)\right)}}{\varphi(x) \vdash \nu S(x)}$$

A derived rule: $$\dfrac{\varphi(x) \vdash \neg B(x) \qquad \exists x.\varphi(x) \land T(x,y) \vdash \nu S(y)}{\varphi(x) \vdash \nu S(x)}\ (\textsc{SymbolicExecution})$$

Symbolic execution

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

$$
\dfrac{
\dfrac{\text{SMT}}{\varphi(x) \vdash \neg B(x)}
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{\exists x.\varphi(x) \land T(x,y) \overset{?}{\vdash} \nu S(y)}{\varphi(x), T(x,y) \vdash \nu S(y)}
}{\varphi(x) \vdash T(x,y) \Rightarrow \nu S(y)}
}{\varphi(x) \vdash \forall y.T(x,y) \Rightarrow \nu S(y)}
}{}
}{
\dfrac{\varphi(x) \vdash \neg B(x) \land \big(\forall y.T(x,y) \Rightarrow \nu S(y)\big)}{\varphi(x) \vdash \nu S(x)}
}
$$

A derived rule:
$$
\dfrac{\varphi(x) \vdash \neg B(x) \qquad \exists x.\varphi(x) \land T(x,y) \vdash \nu S(y)}{\varphi(x) \vdash \nu S(x)}\,(\mathrm{SE})
$$

Bounded model-checking [Biere+ 1999]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

***k* iterations of (SE)** rule coincide with **model-checking within *k* steps**

$$\cfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \cfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \cfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \begin{matrix} \cfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \overset{?}{\varphi_{k+1}(x) \vdash \nu S(x)}}{\varphi_k(x) \vdash \nu S(x)}\ (\mathrm{SE}) \\ \vdots \end{matrix}}{\varphi_2(x) \vdash \nu S(x)}\ (\mathrm{SE})}{\varphi_1(x) \vdash \nu S(x)}\ (\mathrm{SE})}{I(x) \vdash \nu S(x)}\ (\mathrm{SE})$$

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$\dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \varphi_{k+1}(x) \overset{?}{\vdash} \nu S(x)}{\varphi_k(x) \vdash \nu S(x)}\ (\text{SE})$$

$$\vdots$$

$$\dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \vdots}{\varphi_2(x) \vdash \nu S(x)}\ (\text{SE})$$

$$\dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \varphi_2(x) \vdash \nu S(x)}{\varphi_1(x) \vdash \nu S(x)}\ (\text{SE})$$

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \varphi_1(x) \vdash \nu S(x)}{I(x) \vdash \nu S(x)}\ (\text{SE})$$

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

if $\varphi_{k+1}(x) = \varphi_2(x)$

$$\dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \varphi_{k+1}(x) \overset{?}{\vdash} \nu S(x)}{\varphi_k(x) \vdash \nu S(x)}\,(\text{SE})$$

$$\vdots$$

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \vdots}{\varphi_2(x) \vdash \nu S(x)}\,(\text{SE})}{\varphi_1(x) \vdash \nu S(x)}\,(\text{SE})}{I(x) \vdash \nu S(x)}\,(\text{SE})$$

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

if $\varphi_{k+1}(x) = \varphi_2(x)$

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \begin{matrix} \dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \varphi_{k+1}(x) \vdash \nu S(x)}{\varphi_k(x) \vdash \nu S(x)}\,(\mathrm{SE}) \\ \vdots \end{matrix}}{\varphi_2(x) \vdash \nu S(x)}\,(\mathrm{SE})}{\varphi_1(x) \vdash \nu S(x)}\,(\mathrm{SE})}{I(x) \vdash \nu S(x)}\,(\mathrm{SE})$$

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$\dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \varphi_{k+1}(x) \overset{?}{\vdash} \nu S(x)}{\varphi_k(x) \vdash \nu S(x)}\,(\text{SE})$$

$$\vdots$$

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \vdots}{\varphi_2(x) \vdash \nu S(x)}\,(\text{SE})}{\varphi_1(x) \vdash \nu S(x)}\,(\text{SE})}{I(x) \vdash \nu S(x)}\,(\text{SE})$$

Induction hypothesis

Induction hypothesis

Induction hypothesis

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$
\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \vdots}{\varphi_2(x) \vdash \nu S(x)}\,(\mathrm{SE})}{\varphi_1(x) \vdash \nu S(x)}\,(\mathrm{SE})}{I(x) \vdash \nu S(x)}\,(\mathrm{SE})
$$

$$
\vdots
$$

$$
\dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \dfrac{\boxed{\varphi_{k+1}(x) \overset{?}{\vdash} \bigvee_{i=1}^{k} \varphi_i(x)} \qquad ?}{\varphi_{k+1}(x) \vdash \nu S(x)}}{\varphi_k(x) \vdash \nu S(x)}\,(\mathrm{SE})
$$

Induction hypothesis

Induction hypothesis

Induction hypothesis

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$
\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \begin{matrix} \dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \dfrac{\boxed{\overset{\text{SMT}}{\varphi_{k+1}(x) \vdash \bigvee_{i=1}^{k} \varphi_i(x)}} \qquad ?}{\varphi_{k+1}(x) \vdash \nu S(x)}}{\varphi_k(x) \vdash \nu S(x)}\,(\mathrm{SE}) \\ \vdots \end{matrix}}{\varphi_2(x) \vdash \nu S(x)}\,(\mathrm{SE})}{\varphi_1(x) \vdash \nu S(x)}\,(\mathrm{SE})}{I(x) \vdash \nu S(x)}\,(\mathrm{SE})
$$

Induction hypothesis

Induction hypothesis

Induction hypothesis

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \dfrac{\vdots}{}}{\varphi_2(x) \vdash \nu S(x)}\,(\mathrm{SE})}{\varphi_1(x) \vdash \nu S(x)}\,(\mathrm{SE})}{I(x) \vdash \nu S(x)}\,(\mathrm{SE})$$

$$\dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \dfrac{\boxed{\overset{\text{SMT}}{\varphi_{k+1}(x) \vdash \bigvee_{i=1}^{k} \varphi_i(x)}} \qquad \bigvee_{i=1}^{k} \varphi_i(x) \overset{?}{\vdash} \nu S(x)}{\varphi_{k+1}(x) \vdash \nu S(x)}\,(\textsc{Cut})}{\varphi_k(x) \vdash \nu S(x)}\,(\mathrm{SE})$$

Induction hypothesis

Induction hypothesis

Induction hypothesis

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$
\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \begin{matrix} \dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \dfrac{\boxed{\overset{\text{SMT}}{\varphi_{k+1}(x) \vdash \bigvee_{i=1}^{k} \varphi_i(x)}} \qquad \dfrac{\varphi_k(x) \overset{?}{\vdash} \nu S(x) \quad \cdots \quad \varphi_1(x) \overset{?}{\vdash} \nu S(x)}{\bigvee_{i=1}^{k} \varphi_i(x) \vdash \nu S(x)}}{\varphi_{k+1}(x) \vdash \nu S(x)}\ (\textsc{Cut})}{\varphi_k(x) \vdash \nu S(x)}\ (\mathrm{SE}) \\ \vdots \end{matrix}}{\varphi_2(x) \vdash \nu S(x)}\ (\mathrm{SE})}{\varphi_1(x) \vdash \nu S(x)}\ (\mathrm{SE})}{I(x) \vdash \nu S(x)}\ (\mathrm{SE})
$$

Induction hypothesis

Induction hypothesis

Induction hypothesis

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$\dfrac{\boxed{\overset{\text{SMT}}{\varphi_{k+1}(x) \vdash \bigvee_{i=1}^{k} \varphi_i(x)}} \qquad \dfrac{\varphi_k(x) \vdash \nu S(x) \quad \cdots \quad \varphi_1(x) \overset{?}{\vdash} \nu S(x)}{\bigvee_{i=1}^{k} \varphi_i(x) \vdash \nu S(x)}}{\varphi_{k+1}(x) \vdash \nu S(x)}\ (\textsc{Cut})$$

$$\dfrac{\overset{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)} \qquad \varphi_{k+1}(x) \vdash \nu S(x)}{\varphi_k(x) \vdash \nu S(x)}\ (\text{SE})$$

Induction hypothesis

$$\vdots$$

$$\dfrac{\overset{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)} \qquad \vdots}{\varphi_2(x) \vdash \nu S(x)}\ (\text{SE})$$

Induction hypothesis

$$\dfrac{\overset{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)} \qquad \varphi_2(x) \vdash \nu S(x)}{\varphi_1(x) \vdash \nu S(x)}\ (\text{SE})$$

Induction hypothesis

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \varphi_1(x) \vdash \nu S(x)}{I(x) \vdash \nu S(x)}\ (\text{SE})$$

Forward criterion [Sheeran+ 2000]

Heuristic 1 Try to fit the shape of unproved sequents into the form $\varphi(x) \vdash \nu S(x)$

Trying to make cycles after k-th iteration of (SE) rule

$$
\dfrac{
\dfrac{\text{SMT}}{I(x) \vdash \neg B(x)}
\qquad
\dfrac{
\dfrac{\text{SMT}}{\varphi_1(x) \vdash \neg B(x)}
\qquad
\dfrac{
\dfrac{\text{SMT}}{\varphi_2(x) \vdash \neg B(x)}
\qquad
\dfrac{
\dfrac{
\dfrac{\text{SMT}}{\varphi_k(x) \vdash \neg B(x)}
\qquad
\dfrac{
\boxed{\dfrac{\text{SMT}}{\varphi_{k+1}(x) \vdash \bigvee_{i=1}^{k} \varphi_i(x)}}
\qquad
\dfrac{\varphi_k(x) \vdash \nu S(x) \quad \ldots \quad \varphi_1(x) \vdash \nu S(x)}{\bigvee_{i=1}^{k} \varphi_i(x) \vdash \nu S(x)}
}{\varphi_{k+1}(x) \vdash \nu S(x)}\,(\textsc{Cut})
}{\varphi_k(x) \vdash \nu S(x)}\,(\mathrm{SE})
}{\vdots}
}{\varphi_2(x) \vdash \nu S(x)}\,(\mathrm{SE})
}{\varphi_1(x) \vdash \nu S(x)}\,(\mathrm{SE})
}{I(x) \vdash \nu S(x)}\,(\mathrm{SE})
$$

Induction hypothesis

Induction hypothesis

Induction hypothesis

More aggressive use of (Cut)

$$\frac{\varphi(x) \vdash \neg B(x) \qquad \exists x.\varphi(x) \wedge T(x,y) \vdash \psi(y) \qquad \psi(y) \vdash \nu S(y)}{\varphi(x) \vdash \nu S(x)}\ (\text{SE+}\textsc{Cut})$$

$$\left(\frac{\varphi(x) \vdash \neg B(x) \qquad \dfrac{\exists x.\varphi(x) \wedge T(x,y) \vdash \psi(y) \qquad \psi(y) \vdash \nu S(y)}{\exists x.\varphi(x) \wedge T(x,y) \vdash \nu S(y)}\ (\textsc{Cut})}{\varphi(x) \vdash \nu S(x)}\ (\text{SE}) \right)$$

Question **How to select the cut formula ψ?**

Let Ξ be a finite set of formulas (closed under certain logical operations)

Heuristic 2 Let the cut formula be the **strongest** $\psi \in \Xi$ s.t. $\exists x.\, \varphi(x) \wedge T(x,y) \vdash \psi(y)$

Predicate abstraction [Ball+2001] [Graf&Saïdi 1997]

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose $\top$ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \wedge Q_i$** and **solve the constraints on Q_i**

$$I(x) \overset{?}{\vdash} \nu S(x)$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose $\top$ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \wedge Q_i$** and **solve the constraints on Q_i**

$$\dfrac{I(x) \overset{?}{\vdash} \neg B(x) \qquad \exists x.I(x) \wedge T(x,y) \overset{?}{\vdash} \boxed{\top} \qquad \boxed{\top} \overset{?}{\vdash} \nu S(y)}{I(x) \vdash \nu S(x)}\ (\text{SE+Cut})$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose ⊤ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \wedge Q_i$** and **solve the constraints on Q_i**

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \exists x. I(x) \wedge T(x,y) \vdash \overset{?}{\boxed{\top}} \qquad \overset{?}{\boxed{\top}} \vdash \nu S(y)}{I(x) \vdash \nu S(x)}\ (\textsc{SE+Cut})$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose $\top$ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \land Q_i$** and **solve the constraints on Q_i**

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \overset{\text{SMT}}{\exists x. I(x) \land T(x,y) \vdash \boxed{\top}} \qquad \overset{?}{\boxed{\top} \vdash \nu S(y)}}{I(x) \vdash \nu S(x)} \;(\text{SE+Cut})$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose ⊤ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \wedge Q_i$** and **solve the constraints on Q_i**

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \overset{\text{SMT}}{\exists x. I(x) \wedge T(x,y) \vdash \top} \qquad \overset{?}{\top \vdash \nu S(y)}}{I(x) \vdash \nu S(x)}\ (\text{SE+Cut})$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose ⊤ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \wedge Q_i$** and **solve the constraints on Q_i**

$$
\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \overset{\text{SMT}}{\exists x. I(x) \wedge T(x,y) \vdash \top} \qquad \dfrac{\overset{\textbf{False}}{\top \vdash \neg B(y)} \qquad \cdots\cdots}{\top \vdash \nu S(y)}\,(\mathrm{SE}{+}\textsc{Cut})}{I(x) \vdash \nu S(x)}\,(\mathrm{SE}{+}\textsc{Cut})
$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose $\top$ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \land Q_i$** and **solve the constraints on Q_i**

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \exists x. I(x) \land T(x,y) \overset{?}{\vdash} \boxed{\top \land Q_1(y)} \qquad \dfrac{\boxed{\top \land Q_1(y)} \overset{?}{\vdash} \neg B(y) \qquad \cdots\cdots}{\boxed{\top \land Q_1(y)} \vdash \nu S(y)}\,(\text{SE+Cut})}{I(x) \vdash \nu S(x)}\,(\text{SE+Cut})$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose ⊤ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \wedge Q_i$** and **solve the constraints on Q_i**

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \exists x.I(x) \wedge T(x,y) \overset{?}{\vdash} \boxed{\top \wedge Q_1(y)} \qquad \dfrac{\boxed{\top \wedge Q_1(y)} \overset{?}{\vdash} \neg B(y) \quad \cdots\cdots}{\boxed{\top \wedge Q_1(y)} \vdash \nu S(y)}\,(\text{SE+Cut})}{I(x) \vdash \nu S(x)}\,(\text{SE+Cut})$$

$$\text{Constraints: } \{\exists x.I(x) \wedge T(x,y) \vdash \top \wedge Q_1(y), \quad \top \wedge Q_1(y) \vdash \neg B(y)\}$$

IMPACT [McMillan 2006]

Heuristic 3 Tentatively choose ⊤ **as the cut formula**

Heuristic 4 **When the proof attempt fails, strengthen the cut formulas** as follows

- **Replace cut formula φ_i with $\varphi_i \wedge Q_i$** and **solve the constraints on Q_i**

$$\dfrac{\overset{\text{SMT}}{I(x) \vdash \neg B(x)} \qquad \exists x. I(x) \wedge T(x,y) \overset{?}{\vdash} \boxed{\top \wedge Q_1(y)} \qquad \dfrac{\boxed{\top \wedge Q_1(y)} \overset{?}{\vdash} \neg B(y) \quad \cdots\cdots}{\boxed{\top \wedge Q_1(y)} \vdash \nu S(y)}\,(\mathrm{SE{+}C{\scriptsize UT}})}{I(x) \vdash \nu S(x)}\,(\mathrm{SE{+}C{\scriptsize UT}})$$

$$\text{Constraints: } \{\exists x. I(x) \wedge T(x,y) \vdash \top \wedge Q_1(y), \quad \top \wedge Q_1(y) \vdash \neg B(y)\}$$

A solution of this constraint set is called an **interpolant**

Property-directed reachability [Bradley 2011] [Een+ 2011] [Cimatti&Griggio 2012] ...

<u>**Heuristic 5**</u> In strengthening, **keep as many cut formulas unchanged as possible**

In the paper we discuss

- **how to obtain a PDR-like process** from Heuristic 5
- **how to derive a refutationally complete variant** of PDR using MBP
- **unexpected connection** between Heuristic 5 and **a game solving algorithm** [Farzan&Kincaid 2017]

Property-directed reachability

[Bradley 2011] [Een+ 2011] [Cimatti&Griggio 2012] …

Heuristic 5 In strengthening, **keep as many cut formulas unchanged as possible**

maximally conservative

In the paper we discuss

- **how to obtain a PDR-like process** from Heuristic 5
- **how to derive a refutationally complete variant** of PDR using MBP
- **unexpected connection** between Heuristic 5 and **a game solving algorithm**

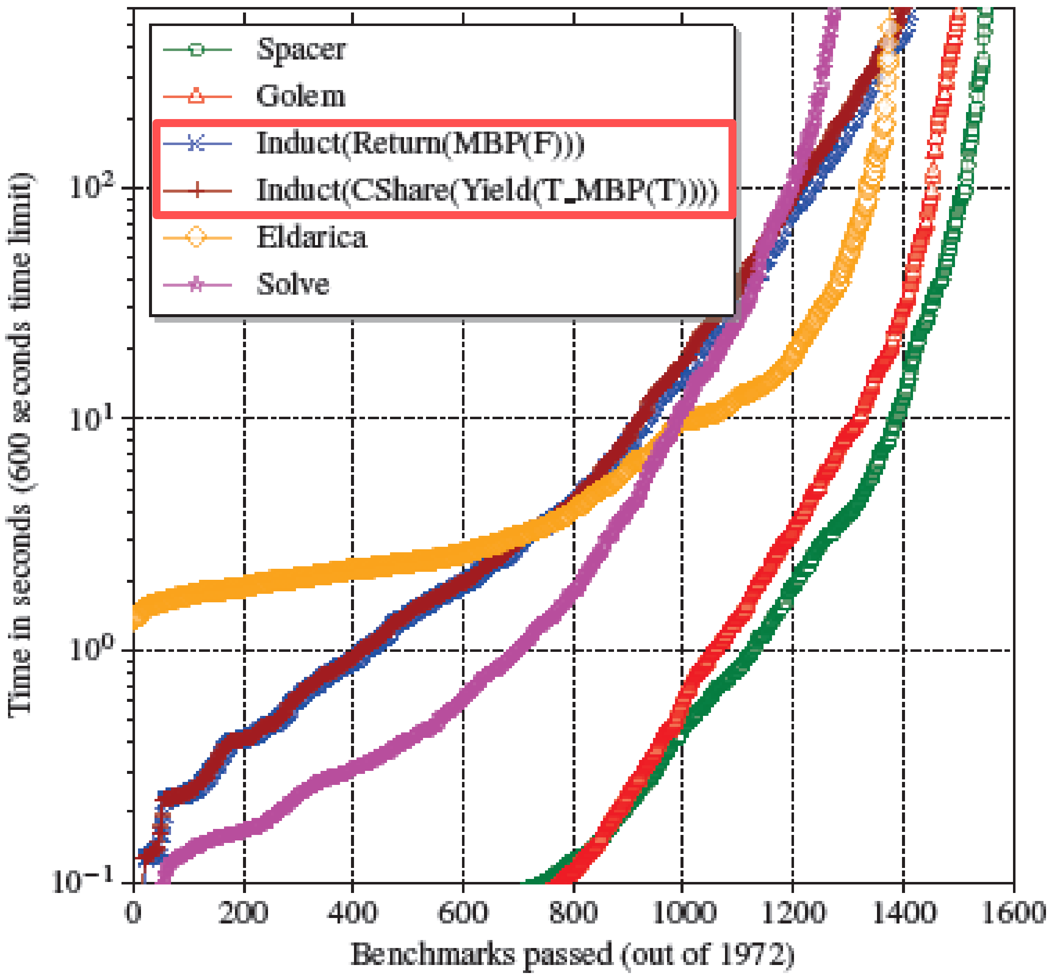
[Farzan&Kincaid 2017]

MuCyc [PLDI 2024]

- Implementation of an extension of the **refutationally complete variant of PDR**
- Available from: https://github.com/hiroshi-unno/coar

Results of CHC-COMP 2025

LIA-Lin	LIA	LIA-Lin-Arrays**	LIA-Arrays	ADT-LIA	ADT-LIA-Arrays	BV	LRA-Lin
Golem	Golem	Eldarica	Eldarica	Catalia	Eldarica	Eldarica	Golem
MuCyc	Eldarica	Unihorn	PCSat	Eldarica	PCSat	Theta	Eldarica
LoAT	PCSat	PCSat	Unihorn	PCSat	--	PCSat	Theta



Summary

Software model-checking algorithms can be seen as **cyclic proof search strategies**

- The connection is rather straightforward

  once the goal sequent is appropriately set

  "All initial states are safe"

  $$I(x) \vdash \nu S(x)$$

  where $\nu S(x) \overset{\nu}{\Leftrightarrow} \neg B(x) \wedge \big(\forall y. T(x,y) \Rightarrow \nu S(y)\big)$

- Several algorithms can be **reconstructed from simple proof-search heuristics**
- The usefulness of the connection is demonstrated by
  - revealing an unexpected connection: **PDR $\approx$ an efficient game solving algorithm**
  - developing a **refutationally complete variant of PDR**

Outline

- Software model-checking as cyclic-proof search
  - Interpretation of various existing software model-checking techniques as different strategies for proof search in a cyclic proof system [POPL 2022]
  - (If time permits) Proof refinement for Spacer [PLDI 2024]
- **Relational verification via cyclic-proof search [CAV 2017]**

[POPL 2022] Tsukada and Unno. Software Model-Checking as Cyclic-Proof Search.
[PLDI 2024] Tsukada and Unno. Inductive Approach to Spacer.
[CAV 2017] Unno et al. Automating Induction for Solving Horn Clauses.

Relational Program Verification

- Verification of properties that relate **multiple executions** of one or more programs
- Clarkson and Schneider formalized such properties as **sets of sets of program traces** and coined the term **hyperproperties** [CSF 2008]
- An important trend in formal methods with wide applications including **security**

[CSF 2008] Clarkson, Schneider. Hyperproperties.

Verification of Algebraic Specifications

Verification of an implementation of an **abstract data type** with **algebraic specs.**

- Arithmetic operations with algebraic specifications:
  - equivalence, associativity, commutativity, distributivity, idempotency, monotonicity, invertibility, symmetry, transitivity, …
- List operations with algebraic specifications such as:
  - append (take xs n) (drop xs n) = xs
- Try out a web interface of our relational verifier from http://lfp.dip.jp/rcaml/

Variants of Program Equivalence

- Functional (i.e., input-output) equivalence
  - Termination-insensitive: $f =_{DTI} g \triangleq \forall x, y_1, y_2. (f(x) \Downarrow y_1) \wedge (g(x) \Downarrow y_2) \Longrightarrow y_1 = y_2$

    An execution of $f(x)$ terminates and returns y_1
  - Termination-sensitive:

    $f(x)$ has a diverging execution

    $f =_{DTS} g \triangleq (f =_{DTI} g) \wedge \forall x. \big((f(x) \Uparrow) \Longrightarrow \neg\exists y. (g(x) \Downarrow y)\big) \wedge \forall x. \big((g(x) \Uparrow) \Longrightarrow \neg\exists y. (f(x) \Downarrow y)\big)$
  - Non-det. & Termination-*sensitive*: $f =_{NdTS} g \triangleq \forall x. \{y \mid f(x) \Downarrow y\} = \{y \mid g(x) \Downarrow y\}$
  - Probabilistic & Termination-*sensitive*: $f =_{PrTS} g \triangleq \forall x, y. \Pr[f(x) \Downarrow y] = \Pr[g(x) \Downarrow y]$
- Trace equivalence: $p =_{Tr} q \triangleq Tr(p) = Tr(q)$

  The set of finite and infinite execution traces of q
- Bisimilarity: $p \sim_{bis} q \triangleq$ there is a strong bisimulation R such that $(p, q) \in R$
- Observational equivalence: $p =_{Obs} q \triangleq \forall C, y. (C[p] \Downarrow y) \Longleftrightarrow (C[q] \Downarrow y)$
  - Captures non-trivial interactions between contexts C and higher-order, object-oriented, and effectful (e.g., non-det., probabilistic, stateful, exception-raising, …) programs
  - In security applications, attackers' capabilities are reflected in the definition of contexts C

Variants of Program Refinement

Useful to transfer properties and proofs!

- Functional (i.e., input-output) refinement:
  - Termination-insensitive: $f \leq_{TI} g \triangleq \forall x, y. (f(x) \Downarrow y) \Longrightarrow (g(x) \Downarrow y)$
    - If $f \leq_{TI} g$, then $\vDash \{Pre\}\ g\ \{Post\}$ implies $\vDash \{Pre\}\ f\ \{Post\}$
  - Termination-sensitive: $f \leq_{TS} g \triangleq f \leq_{TI} g \land \forall x. (f(x) \Uparrow) \Longrightarrow (g(x) \Uparrow)$
    - If $f \leq_{TS} g$, then $\vDash [Pre]\ g\ [Post]$ implies $\vDash [Pre]\ f\ [Post]$ (i.e., termination is also transferred)
- Trace refinement: $p \leq_{Tr} q \triangleq Tr(p) \subseteq Tr(q)$
  - If $p \leq_{Tr} q$, then trace properties of q can be transferred to p
- Similarity: $p \leq_{sim} q \triangleq$ there is a strong simulation R such that $(p, q) \in R$
  - If $p \leq_{sim} q$, then trace (but branching-time) properties of q can be migrated to p
  - If $p \sim_{bis} q$, then branching-time (but hyper-) properties of q can be migrated to p

Program Refinement as *Generalized* Model Checking

- Program refinement verification $\models p \le q$ generalizes ordinary model checking $p \models \phi$
  - **A specification of p is given as a program q** instead of a logical formula ϕ
  - q can encode the given ϕ (if the programming language is expressive enough)
  - q can be **a reference implementation** (cf. seL4 Project) or **an abstract model** represented as a highly non-deterministic program
- This motivates me to investigate entailment checking problems $\psi_1 \models \psi_2$ in a first-order fixpoint logic modulo theories we call μCLP [CAV 2017, LICS 2018, POPL 2023]
- **Relational verification boils down to entailment checking in μCLP**

[CAV 2017] Unno et al. Automating Induction for Solving Horn Clauses.
[LICS 2018] Nanjo et al. A Fixpoint Logic and Dependent Effects for Temporal Property Verification.
[POPL 2023] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.

Example: Functional Program & Relational Spec.

```
(* recursive function to compute "x × y" *)
let rec mult x y =
  if y = 0 then 0 else x + mult x (y - 1)

(* tail recursive function to compute "x × y + a" *)
let rec mult_acc x y a =
  if y = 0 then a else mult_acc x (y - 1) (a + x)

(* functional equivalence of mult and mult_acc *)
let main x y a = assert (mult x y + a = mult_acc x y a)
```

CHCs Constraint Generation based on Dependent Refinement Types [PPDP 2009]

```
let rec mult x y =
  if y = 0 then 0
  else x + mult x (y - 1)
let rec mult_acc x y a =
  if y = 0 then a
  else mult_acc x (y - 1) (a + x)
```

```
let main x y a =
  assert (mult x y + a
           = mult_acc x y a)
```

$$P(x, 0, 0)$$

$$P(x, y, x + r) \Leftarrow P(x, y - 1, r) \wedge y \neq 0$$

$$Q(x, 0, a, a)$$

$$Q(x, y, a, r) \Leftarrow Q(x, y - 1, a + x, r) \wedge y \neq 0$$

$$s_1 + a = s_2 \Leftarrow P(x, y, s_1) \wedge Q(x, y, a, s_2)$$

[PPDP 2009] Unno, Kobayashi. Dependent Type Inference with Interpolants.

CHC Solving via Entailment Checking in $\boldsymbol{\mu}$CLP

The CHCs on the right is satisfiable if and only if the following $\boldsymbol{\mu}$**CLP** entailment holds

$$P(x, y, s_1), Q(x, y, a, s_2) \vDash s_1 + a = s_2$$

where

$$P(x, y, z) =_\mu \begin{pmatrix} y = 0 \land z = 0 \lor \\ y \neq 0 \land P(x, y-1, r) \land z = x + r \end{pmatrix}$$

$$Q(x, y, a, r) =_\mu \begin{pmatrix} y = 0 \land r = a \lor \\ y \neq 0 \land Q(x, y-1, a+x, r) \end{pmatrix}$$

$$\boxed{\begin{array}{l} P(x, 0, 0) \\ P(x, y, x + r) \Leftarrow P(x, y-1, r) \land y \neq 0 \\ Q(x, 0, a, a) \\ Q(x, y, a, r) \Leftarrow Q(x, y-1, a+x, r) \land y \neq 0 \\ s_1 + a = s_2 \Leftarrow P(x, y, s_1) \land Q(x, y, a, s_2) \end{array}}$$

$\boldsymbol{\mu}$CLP: A First-Order *Fixed-Point Logic* Modulo Background Theories T [POPL 2023]

predicate symbols of T

(*formulas*) $\phi ::= \bot \mid \top \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall x.\phi \mid \exists x.\phi \mid P(\vec{t}) \mid p(\vec{t})$

(*predicates*) $P ::= X \mid \boldsymbol{\mu X.\lambda\vec{x}.\phi} \mid \boldsymbol{\nu X.\lambda\vec{x}.\phi}$ (*terms*) $t ::= x \mid f(\vec{t})$

predicate variables

Least fixed-point (X occurs only positively in ϕ)

Greatest fixed-point (X occurs only positively in ϕ)

term variables

constant and function symbols of T

- We assume that formulas, predicates, and terms are well-sorted
- Least fixpoints $\boldsymbol{\mu X.\lambda\vec{x}.\phi}$ represent *inductive predicates*, and greatest fixpoints $\boldsymbol{\nu X.\lambda\vec{x}.\phi}$ represent *co-inductive predicates*
- We also use (hierarchical) equational form: $X(\vec{x}) =_\mu \phi$ and $X(\vec{x}) =_\nu \phi$

$$P(x,y,z) =_\mu \begin{pmatrix} y = 0 \wedge z = 0 \vee \\ y \neq 0 \wedge P(x, y-1, r) \wedge z = x + r \end{pmatrix}$$

$$Q(x,y,a,r) =_\mu \begin{pmatrix} y = 0 \wedge r = a \vee \\ y \neq 0 \wedge Q(x, y-1, a+x, r) \end{pmatrix}$$

$$P(x,y,s_1), Q(x,y,a,s_2) \overset{?}{\vdash} s_1 + a = s_2$$

$$P(x,y,z) =_\mu \begin{pmatrix} y = 0 \wedge z = 0 \vee \\ y \neq 0 \wedge P(x, y-1, r) \wedge z = x + r \end{pmatrix}$$

$$Q(x,y,a,r) =_\mu \begin{pmatrix} y = 0 \wedge r = a \vee \\ y \neq 0 \wedge Q(x, y-1, a+x, r) \end{pmatrix}$$

$$\dfrac{\dfrac{\dfrac{}{y = 0 \wedge s_1 = 0, y = 0 \wedge s_2 = a \vdash s_1 + a = s_2}\text{SMT} \quad \overset{?}{\vdots}}{y = 0 \wedge s_1 = 0, Q(x,y,a,s_2) \vdash s_1 + a = s_2} \qquad \overset{?}{\cdots, Q(x,y,a,s_2) \vdash s_1 + a = s_2}}{P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$P(x,y,z) =_\mu \begin{pmatrix} y = 0 \wedge z = 0 \vee \\ y \neq 0 \wedge P(x, y-1, r) \wedge z = x + r \end{pmatrix}$$

$$Q(x,y,a,r) =_\mu \begin{pmatrix} y = 0 \wedge r = a \vee \\ y \neq 0 \wedge Q(x, y-1, a+x, r) \end{pmatrix}$$

$$\dfrac{\dfrac{\overset{\text{SMT}}{\cdots} \quad \overset{\text{SMT}}{y = 0 \wedge s_1 = 0, y \neq 0 \wedge \cdots \vdash s_1 + a = s_2}}{y = 0 \wedge s_1 = 0, Q(x,y,a,s_2) \vdash s_1 + a = s_2} \qquad \overset{?}{\cdots, Q(x,y,a,s_2) \vdash s_1 + a = s_2}}{P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$P(x,y,z) =_\mu \begin{pmatrix} y = 0 \land z = 0 \lor \\ y \neq 0 \land P(x, y-1, r) \land z = x + r \end{pmatrix}$$

$$Q(x,y,a,r) =_\mu \begin{pmatrix} y = 0 \land r = a \lor \\ y \neq 0 \land Q(x, y-1, a+x, r) \end{pmatrix}$$

$$\dfrac{\vdots \qquad \dfrac{\dfrac{\text{SMT}}{y \neq 0 \land P(x,y-1,r) \land s_1 = x + r, y = 0 \land s_2 = a \vdash s_1 + a = s_2} \quad \cdots}{y \neq 0 \land P(x,y-1,r) \land s_1 = x + r, Q(x,y,a,s_2) \vdash s_1 + a = s_2}}{P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$P(x,y,z) =_\mu \begin{pmatrix} y = 0 \wedge z = 0 \vee \\ y \neq 0 \wedge P(x, y-1, r) \wedge z = x + r \end{pmatrix}$$

$$Q(x,y,a,r) =_\mu \begin{pmatrix} y = 0 \wedge r = a \vee \\ y \neq 0 \wedge Q(x, y-1, a+x, r) \end{pmatrix}$$

$$\dfrac{\vdots \qquad \dfrac{\vdots \quad y \neq 0 \wedge P(x,y-1,r) \wedge s_1 = x + r, y \neq 0 \wedge Q(x, y-1, a+x, s_2) \overset{?}{\vdash} s_1 + a = s_2}{y \neq 0 \wedge P(x,y-1,r) \wedge s_1 = x + r, Q(x,y,a,s_2) \vdash s_1 + a = s_2}}{P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$P(x,y,z) =_\mu \begin{pmatrix} y = 0 \land z = 0 \lor \\ y \neq 0 \land P(x, y-1, r) \land z = x + r \end{pmatrix}$$

$$Q(x,y,a,r) =_\mu \begin{pmatrix} y = 0 \land r = a \lor \\ y \neq 0 \land Q(x, y-1, a+x, r) \end{pmatrix}$$

$$\dfrac{\vdots \qquad \dfrac{\vdots \qquad \dfrac{P(x, y-1, r), Q(x, y-1, a+x, s_2) \vdash (y \neq 0 \land s_1 = x + r) \Rightarrow s_1 + a = s_2 \;\; ?}{y \neq 0 \land P(x, y-1, r) \land s_1 = x + r, y \neq 0 \land Q(x, y-1, a+x, s_2) \vdash s_1 + a = s_2}}{y \neq 0 \land P(x, y-1, r) \land s_1 = x + r, Q(x, y, a, s_2) \vdash s_1 + a = s_2}}{P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$P(x,y,z) =_\mu \begin{pmatrix} y = 0 \land z = 0 \lor \\ y \neq 0 \land P(x, y-1, r) \land z = x + r \end{pmatrix}$$

$$Q(x,y,a,r) =_\mu \begin{pmatrix} y = 0 \land r = a \lor \\ y \neq 0 \land Q(x, y-1, a+x, r) \end{pmatrix}$$

QED

?

SMT

$$\dfrac{\vdots \quad \dfrac{\vdots \quad \dfrac{\dfrac{\dfrac{P(x,y-1,r), Q(x,y-1,a+x,s_2) \vdash r + (a+x) = s_2 \qquad r + (a+x) = s_2 \vdash \cdots}{P(x,y-1,r), Q(x,y-1,a+x,s_2) \vdash (y \neq 0 \land s_1 = x + r) \Rightarrow s_1 + a = s_2}}{y \neq 0 \land P(x,y-1,r) \land s_1 = x + r, y \neq 0 \land Q(x,y-1,a+x,s_2) \vdash s_1 + a = s_2}}{y \neq 0 \land P(x,y-1,r) \land s_1 = x + r, Q(x,y,a,s_2) \vdash s_1 + a = s_2}}{P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$\{y \mapsto y - 1, s_1 \mapsto r, a \mapsto a + x\}$$

Proof-Search Heuristics

- Use the following rule application strategy:
  - Select some $P(\vec{t})$ and apply UNFOLD
  - Try to make a cycle whenever a new sequent is added
  - If failed, apply VALID
- VALID rule uses
  - SMT solvers: provide efficient and powerful reasoning about **data structures** (e.g., integers, reals, algebraic data structures) but predicates are abstracted as uninterpreted ones
  - CHC solvers: provide bit costly but powerful reasoning about **inductive predicates**

A Prototype Entailment Checker **MuCyc** http://lfp.dip.jp/rcaml/

- Use **Z3** and **SPACER** respectively as the backend SMT and CHC solvers
- Integrated with a dependent refinement type based CHC generation tool **RCaml** for OCaml
- Currently support entailments in
  - The fragment corresponding to CHCs: $P_1(\overrightarrow{x_1}), \ldots, P_n(\overrightarrow{x_n}) \vDash \phi$ and
  - $P_1(\overrightarrow{x_1}), \ldots, P_n(\overrightarrow{x_n}) \vDash Q(\vec{y})$, which is useful for program refinement verification and proving lemmas to prove entailments in the above fragment (cf. commutativity proof of mult)
- Can prove and then exploit lemmas which are:
  - User-supplied,
  - Heuristically conjectured from the given constraints, or
  - Automatically generated by an abstract interpreter
- Can generate a counterexample (if any)

Experiments on IsaPlanner Benchmark Set

- 85 (mostly) relational verification problems of total functions on inductively defined data structures

Inductive Theorem Prover	#Successfully Proved
RCaml	68
Zeno	82 [Sonnex+ ’12]
HipSpec	80 [Claessen+ ’13]
CVC4	80 [Reynolds+ ’15]
ACL2s	74 (according to [Sonnex+ ’12])
IsaPlanner	47 (according to [Sonnex+ ’12])
Dafny	45 (according to [Sonnex+ ’12])

Support automatic lemma discovery & goal generalization

Experiments on Benchmark Programs with Advanced Language Features & Side-Effects

- 30 (mostly) relational verification problems for:
  - Complex integer functions: Ackermann, McCarthy91
  - Nonlinear real functions: dyn_sys
  - Higher-order functions: fold_left, fold_right, repeat, find, ...
  - Exceptions: find
  - Non-terminating functions: mult, sum, ...
  - Non-deterministic functions: randpos
  - Imperative procedures: mult_Ccode

ID	specification	kind	features	result	time (sec.)
1	$\texttt{mult}\ x\ y + a = \texttt{mult\_acc}\ x\ y\ a$	equiv	P	✓	0.378
2	$\texttt{mult}\ x\ y = \texttt{mult\_acc}\ x\ y\ 0$	equiv	P	✓†	0.803
3	$\texttt{mult}\ (1+x)\ y = y + \texttt{mult}\ x\ y$	equiv	P	✓	0.403
4	$y \geq 0 \Rightarrow \texttt{mult}\ x\ (1+y) = x + \texttt{mult}\ x\ y$	equiv	P	✓	0.426
5	$\texttt{mult}\ x\ y = \texttt{mult}\ y\ x$	comm	P	✓‡	0.389
6	$\texttt{mult}\ (x+y)\ z = \texttt{mult}\ x\ z + \texttt{mult}\ y\ z$	dist	P	✓	1.964
7	$\texttt{mult}\ x\ (y+z) = \texttt{mult}\ x\ y + \texttt{mult}\ x\ z$	dist	P	✓	4.360
8	$\texttt{mult}\ (\texttt{mult}\ x\ y)\ z = \texttt{mult}\ x\ (\texttt{mult}\ y\ z)$	assoc	P	✗	n/a
9	$0 \leq x_1 \leq x_2 \land 0 \leq y_1 \leq y_2 \Rightarrow \texttt{mult}\ x_1\ y_1 \leq \texttt{mult}\ x_2\ y_2$	mono	P	✓	0.416
10	$\texttt{sum}\ x + a = \texttt{sum\_acc}\ x\ a$	equiv		✓	0.576
11	$\texttt{sum}\ x = x + \texttt{sum}\ (x-1)$	equiv		✓	0.452
12	$x \leq y \Rightarrow \texttt{sum}\ x \leq \texttt{sum}\ y$	mono		✓	0.593

- 28 (2 required lemmas) successfully proved by **MuCyc**
- 3 proved by CHC constraint solver ***μ*Z PDR**
- 2 proved by inductive theorem prover **CVC4** (if inductive predicates are encoded using uninterpreted functions)

ID	specification	kind	features	result	time (sec.)
24	$\texttt{noninter}\ h_1\ l_1\ l_2\ l_3 = \texttt{noninter}\ h_2\ l_1\ l_2\ l_3$	nonint	P	✓	1.203
25	$\texttt{try find\_opt}\ p\ l = \texttt{Some}\ (\texttt{find}\ p\ l)\ \texttt{with}$ $\texttt{Not\_Found} \rightarrow \texttt{find\_opt}\ p\ l = \texttt{None}$	equiv	H, E	✓	1.065
26	$\texttt{try mem}\ (\texttt{find}\ ((=)\ x)\ l)\ l\ \texttt{with Not\_Found} \rightarrow \neg(\texttt{mem}\ x\ l)$	equiv	H, E	✓	1.056
27	$\texttt{sum\_list}\ l = \texttt{fold\_left}\ (+)\ 0\ l$	equiv	H	✓	6.148
28	$\texttt{sum\_list}\ l = \texttt{fold\_right}\ (+)\ l\ 0$	equiv	H	✓	0.508
29	$\texttt{sum\_fun randpos}\ n > 0$	equiv	H,D	✓	0.319
30	$\texttt{mult}\ x\ y = \texttt{mult\_Ccode}(x, y)$	equiv	P, C	✓	0.303

† A lemma $P_{\texttt{mult\_acc}}(x, y, a, r) \Rightarrow P_{\texttt{mult\_acc}}(x, y, a - x, r - x)$ is used

‡ A lemma $P_{\texttt{mult}}(x, y, r) \Rightarrow P_{\texttt{mult}}(x - 1, y, r - y)$ is used

Used a machine with Intel(R) Xeon(R) CPU (2.50 GHz, 16 GB of memory).

Summary

- The integration of **SMT solving**, **CHC solving**, and **cyclic-proof search** resulted in an automated **relational verifier** across programs in various paradigms with **advanced language features** and **side-effects**
- Current limitations
  - Limited support for automatic lemma discovery and goal generalization
  - Does not support the full fragment of $\boldsymbol{\mu}$**CLP**

Course Schedule

- Wed. 21 May (8:50-10:30)
  1. Reduction from software verification to fixed-point logic validity checking
  2. Predicate constraint solving for validity checking
- Thu. 22 May (11:20-12:20)
  3. Cyclic-proof search for validity checking
  4. **Game solving for validity checking**

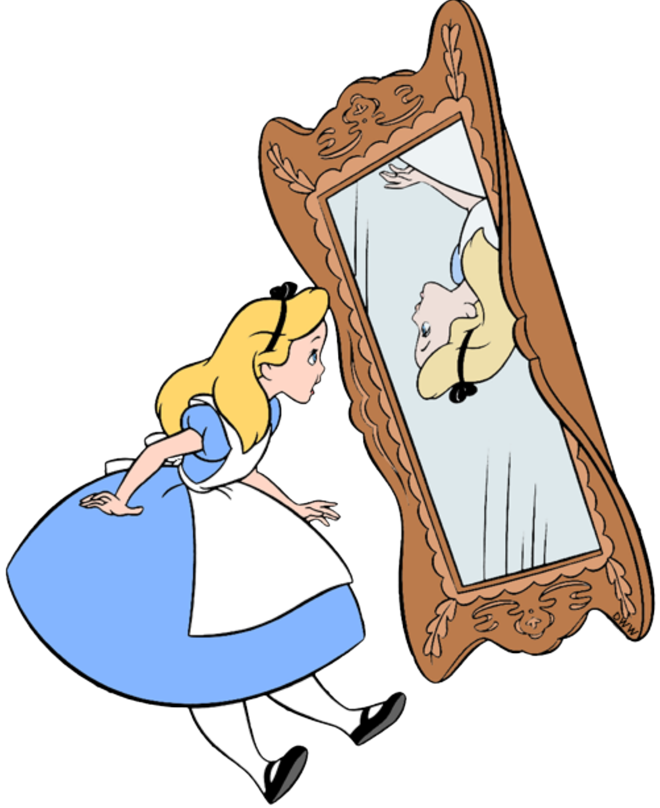
4. Game Solving for Validity Checking

Outline

- A unified primal-dual framework for verification methods based on the concept of Lagrangians [POPL 2025]
  - We derive a validity checking method for μCLP based on game solving by analyzing, organizing, and integrating existing software verification and game solving techniques within the framework

[POPL 2025] Tsukada et al. A Primal-Dual Perspective on Program Verification Algorithms.

Duality in verification algorithms



- Many algorithms in verification have a primal-dual "feel"
  - duality between "proofs" and "counterexamples"
    - CEGAR, ICE Learning, IC3/PDR
    - CDCL, CDCL(T), MBQI
- Recent algorithms exploit formal duality
  - [POPL'22] Oded Padon, James R. Wilcox, Jason R. Koenig, Kenneth L. McMillan, and Alex Aiken. Induction Duality: Primal-Dual Search for Invariants.
  - [POPL'23] Hiroshi Unno, Tachio Terauchi, Yu Gu, Eric Koskinen. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.

Our contribution

- Lagrangian-based unifying framework for primal-dual algorithms
  - Inspired by linear programming
  - Captures many verification algorithms
    - CEGAR, ICE-learning, primal-dual Houdini, termination verification algorithms, and quantified SMT solving
  - Interesting theoretical properties
- Interesting comparisons between existing algorithms
- Derivation of a new validity checking method for μCLP

Generalized Lagrangian duality

- Linear optimization:

$$L: \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$$
$$L(x, \lambda) = f(x) + \langle \lambda, g(x) \rangle$$
$$\sup_\lambda \inf_x L(x, \lambda) = L(x^*, \lambda^*) = \inf_x \sup_\lambda L(x, \lambda)$$

Strong duality
(May not hold)

- Generalization:
  - Let X, Y be general sets, and $(Z, \leq)$ a totally-ordered complete lattice

$$L: X \times Y \to Z$$
$$\sup_y \inf_x L(x, y) = \inf_x \sup_y L(x, y)$$
$$\sup_y \inf_x L(x, y) \leq \inf_x \sup_y L(x, y)$$

Weak duality
Always holds

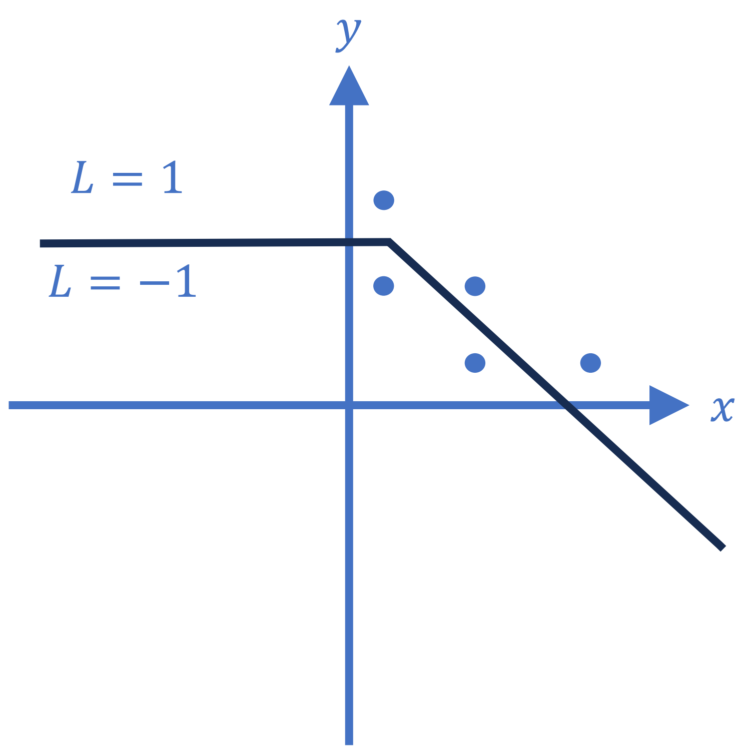
Lagrangian duality for program verification

$$L: X \times Y \to Z$$
$$\sup_{y \in Y} \inf_{x \in X} L(x, y) \leq \inf_{x \in X} \sup_{y \in Y} L(x, y)$$

- X – space of possible (partial, abstract) counterexamples
- Y – space of possible (partial) proofs
- $L(x, y) = \begin{cases} -1 & \text{if } x \text{ is a counterexample that shows } y \text{ isn't a valid proof} \\ 1 & \text{otherwise} \end{cases}$
- 2-player game: X player tries to minimize L with a good counterexample
  Y player tries to maximize L with a good proof

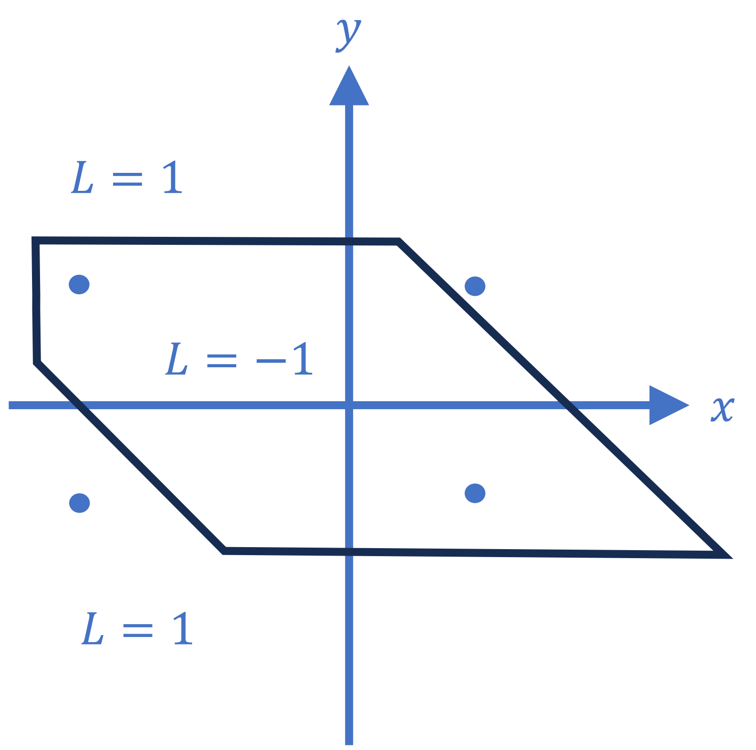
Primal-dual procedure

let $x \in X,\ y \in Y$
while T:
    if $\inf_{x' \in X} L(x', y) \geq 0$:
        $\boxed{\texttt{return (dual!, } y\texttt{)}}$
    update x s.t. $L(x, y) < 0$
    if $\sup_{y' \in Y} L(x, y') \leq 0$:
        return (primal!, x)
    update y s.t. $L(x, y) > 0$



Primal-dual procedure

let $x \in X,\ y \in Y$
while T:
    if $\inf\limits_{x' \in X} L(x', y) \geq 0$:
        return (dual!, y)
    update x s.t. $L(x, y) < 0$
    if $\sup\limits_{y' \in Y} L(x, y') \leq 0$:
        return (primal!, x)
    update y s.t. $L(x, y) > 0$



Monotonicity and progress

$$L: X \times Y \rightarrow Z$$

$$\sup_{y \in Y} \inf_{x \in X} L(x, y) \leq \inf_{x \in X} \sup_{y \in Y} L(x, y)$$

- X – space of possible (partial, abstract) counterexamples
- Y – space of possible (partial) proofs
- If X or Y (or both) have a lattice structure and L is monotone on Y or anti-monotone on X then we can ensure progress

Primal-dual procedure

```
let x ∈ X, y ∈ Y
while T:
    if inf_{x'∈X} L(x', y) ≥ 0:
        return (dual!, y)
    update x s.t. L(x, y) < 0
    if sup_{y'∈Y} L(x, y') ≤ 0:
        return (primal!, x)
    update y s.t. L(x, y) > 0
```

Primal-dual procedure

```
let x ∈ X, y ∈ Y
while T:
    if inf_{x′∈X} L(x′, y) ≥ 0:
        return (dual!, y)
    update x s.t. L(x, y) < 0
    if sup_{y′∈Y} L(x, y′) ≤ 0:
        return (primal!, x)
    update y s.t. L(x, y) > 0
```

monotonically

monotonically

Theorems

- (Partial) correctness
- Progress under monotonicity
- Termination via stratification

Example: CEGAR (counterexample guided abstraction refinement)

- Let $A \in \mathcal{P}_{fin}(P)$ be a finite set of predicates
- Partition the state space S to $2^{|A|}$ equivalence classes of states
- Check the abstract system for safety
  - Safe → terminate (original system is safe)
  - Not safe → Refine A or terminate (original system not safe)

$$I \ni s_0 \longrightarrow s_1 \approx_A s_2 \longrightarrow s_3 \underset{\not\approx_{A \cup \{p\}}}{\approx_A} s_4 \longrightarrow s_5 \approx_A s_6 \longrightarrow s_7 \in B$$

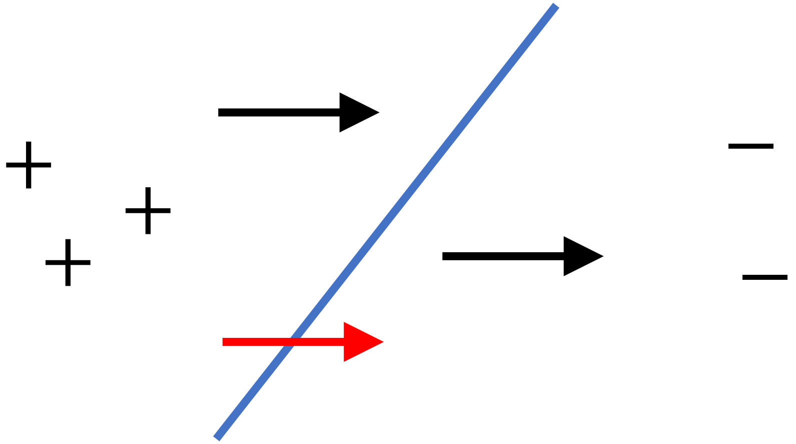
Example: CEGAR (counterexample guided abstraction refinement)

$$I \ni s_0 \longrightarrow s_1 \approx_A s_2 \longrightarrow s_3 \approx_A s_4 \longrightarrow s_5 \approx_A s_6 \longrightarrow s_7 \in B$$

$$\not\approx_{A \cup \{p\}}$$

- $X = S^*, Y = \mathcal{P}_{fin}(P)$ ordered by $\subseteq$
- $L_{\text{CEGAR}}(\langle s_0, s_1, \ldots, s_n \rangle, A) = \begin{cases} -1 & \text{if } \langle s_0, s_1, \ldots, s_n \rangle \text{ is an abs. cex. trace to A} \\ 1 & \text{otherwise} \end{cases}$

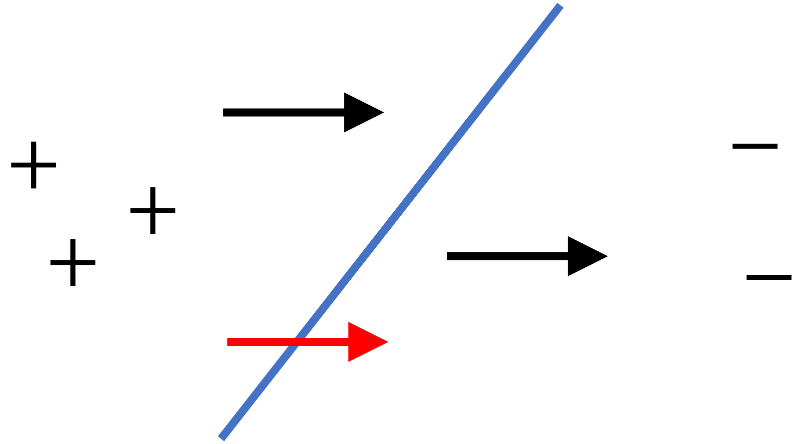
Example: ICE learning / CEGIS



- Let $S_+ \in \mathcal{P}_{fin}(S)$ be a finite set of initial states,
  $S_- \in \mathcal{P}_{fin}(S)$ a finite set of bad states, and
  $S_\to \in \mathcal{P}_{fin}(S \times S)$ a finite set of transitions
- Find a predicate $p \in P$ that satisfies $S_+, S_\to, S_-$
- Check if p is an inductive invariant for the original system
  - If yes, terminate (original system is safe)
  - If not, refine $S_+, S_\to, S_-$

[CAV 2021] Unno et al. Constraint-Based Relational Verification.
[CAV 2014] Garg et al. ICE: A Robust Framework for Learning Invariants.

Example: ICE learning / CEGIS



- $X = \mathcal{P}_{fin}(S) \times \mathcal{P}_{fin}(S \times S) \times \mathcal{P}_{fin}(S)$
  - ordered by $\subseteq \times \subseteq \times \subseteq$
- $Y = P$
- $L_{\text{ICE}}(\langle S_+, S_\rightarrow, S_- \rangle, p) = \begin{cases} 1 & \text{if } p \text{ is an inductive invariant for } \langle S_+, S_\rightarrow, S_- \rangle \\ -1 & \text{otherwise} \end{cases}$

Example: primal-dual Houdini

- Primal-dual Houdini [POPL 2022] uses a dual transition system that represents incremental induction proofs
- $X = \mathcal{P}_{fin}(S), Y = \mathcal{P}_{fin}(P)$
- $L_{\text{pdH}}(x, y) =$
$$\begin{cases} -1 & \text{if no subset of } y \text{ is a safe inductive invariant for the TS reduced to } x \\ 1 & \text{if no subset of } x \text{ is a safe inductive invariant for the dual TS reduced to } y \\ 0 & \text{otherwise} \end{cases}$$
- Well-definedness of this Lagrangian is non-trivial
- It uses three values and not just two
- The Lagrangian is symmetric, and the algorithm makes monotonic progress on both sides

[POPL 2022] Padon et al. Induction Duality: Primal-Dual Search for Invariants.

Lagrangians for Termination

- Termination is typically proven with ranking functions
- ICE for termination [CAV 2021rel, CAV2021dt]

$$X := \mathcal{P}_{\text{fin}}(I_{\mathbb{S}}) \times \mathcal{P}_{\text{fin}}(\rightsquigarrow_{\mathbb{S}}) \quad \text{and} \quad Y := \mathfrak{R} \subseteq (|\mathbb{S}| \to \mathbb{N})$$

$$L_{\text{T-ICE}}(S', r) = \begin{cases} 1 & r \text{ is a ranking function for the subsystem } S' \text{ of } \mathbb{S} \\ -1 & \text{otherwise.} \end{cases}$$

- CEGAR for termination? How to make progress on the side of proofs?
  - Disjunctive well-founded relations (transition invariants) [LICS 2004]

$$X := \{s_0 s_1 \ldots s_n \in |\mathbb{S}| \mid I_{\mathbb{S}} \ni s_0 \rightsquigarrow_{\mathbb{S}} \ldots \rightsquigarrow_{\mathbb{S}} s_n\}, Y := \mathcal{P}_{\text{fin}}(\mathfrak{R})$$

$$L_{\text{T-CEGAR}}(s_0 \ldots s_n, R) = \begin{cases} 1 & \forall i.\forall j.(i < j) \Rightarrow \exists r \in R. r(s) > r(s') \\ -1 & \text{otherwise.} \end{cases}$$

[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[LICS 2004] Podelski and Rybalchenko. Transition Invariants.

Lagrangian for Quantified Formulas

- Consider a formula $\forall a \in \mathbb{Q}. \exists b \in \mathbb{Q}. \forall c \in \mathbb{Q}. \varphi(a, b, c)$
- Define X and Y to be strategies (Skolem functions) for universal and existential quantifiers

$$X = (\text{Skolem functions for } a \text{ and } c) = (\mathbb{Z} \times (\mathbb{Z} \to \mathbb{Z}))$$
$$Y = (\text{Skolem function for } b) = (\mathbb{Z} \to \mathbb{Z}),$$

$L : X \times Y \longrightarrow \{-1, 1\}$ is given by

$$L((f_a, f_c), f_b) = 1 \quad :\Leftrightarrow \quad \varphi(f_a, f_b(f_a), f_c(f_b(f_a))) \text{ is true.}$$

- Strong duality: $\sup_{y \in Y} \inf_{x \in X} L(x, y) = \inf_{x \in X} \sup_{y \in Y} L(x, y) = \begin{cases} -1 & \text{if the formula is false} \\ 1 & \text{if the formula is true} \end{cases}$
- What about monotonicity and progress?

Strategy Skeletons [IJCAI 2016]

- Rather than Skolem functions, let X and Y represent sets of possible functions represented by Strategy Skeletons
- Roughly, a set of finite terms
- Then, the obtained Lagrangian is anti-monotone (on X) and monotone on Y, and the algorithm of [IJCAI 2016] can be seens as an instance of the primal-dual Lagrangian-based procedures
- Interestingly, strong duality holds

[IJCAI 2016] Farzan and Kincaid. Linear Arithmetic Satisfiability via Strategy Improvement.

Lagrangian for Fixed-Point Logic Formulas

- Consider a formula $\forall z. \Big(\nu A. \lambda x. \big(\mu B. \lambda y. y = 0 \vee B(y-1)\big)(x) \wedge A(x+1)\Big)(z)$
- Define X and Y to be strategies (ranking functions) for ν and μ operators, as well as strategies (Skolem functions) for ∀ and ∃ quantifiers)

$$X = (\text{Skolem funcion for } z) \times (\text{ranking function for } \nu A) = \mathbb{z} \times \mathfrak{N}$$
$$Y = (\text{ranking function for } \mu B) = \mathfrak{N}$$

$L: X \times Y \to \{-1,1\}$ is given by

$$L\big((s_z, r_A), r_B\big) = 1 \Leftrightarrow r_B \text{ strategy defeats } (s_z, r_A) \text{ strategy}$$

- What about monotonicity and progress?
  - Strategy skeletons for quantifiers and disjunctively well-founded relations for fixpoint operators

An example play of the game using

$$s_Z = 1, \qquad r_A(x) = \max(3-x, 0) \quad r_B(x) = \max(x, 0)$$

$$
\begin{aligned}
& \big(\nu A.\lambda x.\big(\mu B.\lambda y.y = 0 \vee B(y-1)\big)(x) \wedge A(x+1)\big)(1) \\
\to\ & \underline{\big(\mu B.\lambda y.y = 0 \vee B(y-1)\big)(1)} \wedge (\nu A.\cdots)(2) \\
\to\ & \underline{\big(\mu B.\lambda y.y = 0 \vee B(y-1)\big)(0)} \wedge (\nu A.\cdots)(2) \\
\to\ & 0 = 0 \wedge \underline{(\nu A.\cdots)(2)} \\
\to^*\ & \underline{(\nu A.\cdots)(3)} \\
\to^*\ & \underline{(\nu A.\cdots)(4)} \\
\to^*\ & \top \quad (X \text{ player, i.e., the refuter failed!})
\end{aligned}
$$

$r_B(1) > r_B(0)$

$r_A(2) > r_A(3)$

$r_A(2) > r_A(4)$
$r_A(3) \not> r_A(4)$

Implementation and Evaluation

- Implemented **MuStrat** in OCaml 5, using Z3 and SPACER as the backend SMT and CHC solvers

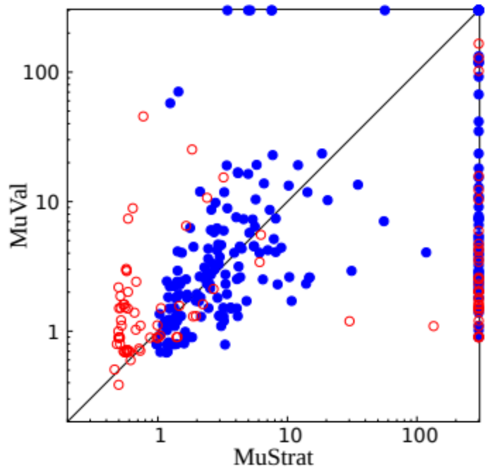


Fig. 1. **MuStrat** vs. **MuVal** on the (non-)term. benchmark set from termCOMP (C Integer).

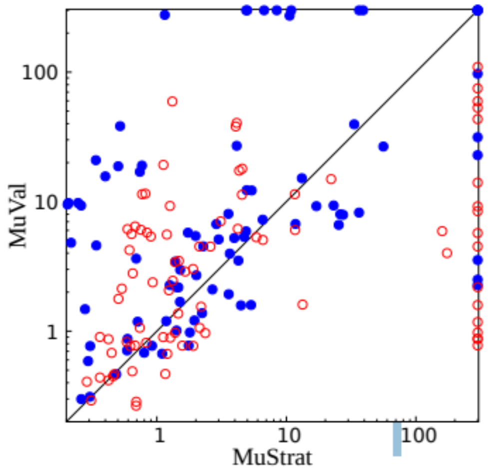


Fig. 2. **MuStrat** vs. **MuVal** on the fixed-point logic benchmark set from [Unno et al. 2023].

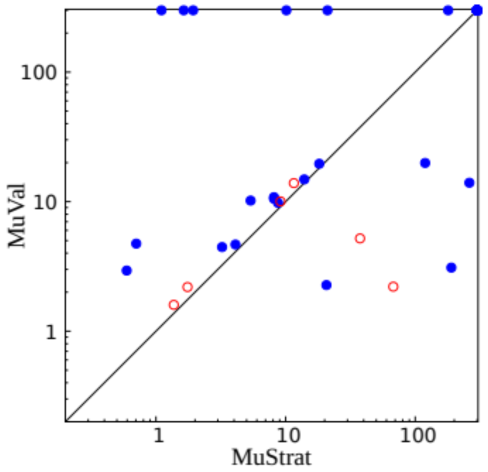


Fig. 3. **MuStrat** vs. **MuVal** on the game solving benchmark set from [Heim and Dimitrova 2024].

Primal-dual algorithm	Makes progress on
CEGAR	Proofs
ICE learning	Counterexamples
Primal-dual Houdini	Both (fully symmetric)
CEGAR for termination	Proofs • using disjunctive rankings [LICS 2004]
Ranking function synthesis [CAV2021dt, CAV 2021rel]	Counterexamples
Solving quantified formulas with strategy skeletons [IJCAI 2016]	Both • using strategy skeletons
MuStrat: New algorithm for quantified fixpoint logic over arithmetic	Both • strategy skeletons + disjunctive rankings

[CAV 2021rel] Unno et al. Constraint-based Relational Verification.
[CAV 2021dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[LICS 2004] Podelski and Rybalchenko. Transition Invariants.
[IJCAI 2016] Farzan and Kincaid. Linear Arithmetic Satisfiability via Strategy Improvement.

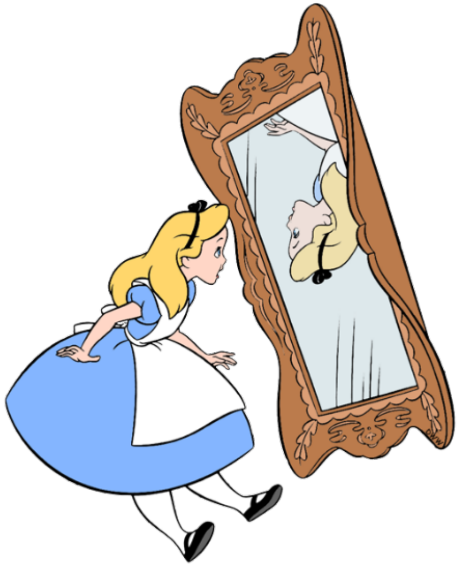
Weak duality vs strong duality

$$L: X \times Y \rightarrow Z$$

$$\sup_{y \in Y} \inf_{x \in X} L(x, y) \leq \inf_{x \in X} \sup_{y \in Y} L(x, y)$$

- X – space of possible (partial, abstract) counterexamples
- Y – space of possible (partial) proofs
- When does strong duality $\sup_{y \in Y} \inf_{x \in X} L(x, y) = \inf_{x \in X} \sup_{y \in Y} L(x, y)$ hold?
  - If there is either a valid proof or a counterexample
  - Holds in finite-state cases, or in idealized versions
  - Analogous to relative completeness

Summary



- Lagrangian Duality
  - General framework for primal-dual verification algorithms
  - Inspired by a well-known duality from linear programming
  - Captures several existing algorithms
  - Sheds new light on existing algorithms, can lead to new algorithms
  - Many more details in the paper
    - New formulation of primal-dual Houdini
    - Lagrangian-based design of new algorithm for quantified fixpoint logic

Conclusion

1. Reduction from software verification to fixed-point logic validity checking
2. Predicate constraint solving for validity checking
3. Cyclic-proof search for validity checking
4. Game solving for validity checking

- By analyzing, organizing, and integrating existing verification methods based on ***unified logical frameworks*** grounded in ***fixed-point logics***, ***predicate constraint solving***, ***cyclic-proof search***, and ***game solving***, we can derive ***new verification methods*** that are ***correct***, ***efficient***, and ***highly extensible***

Ongoing and Future Work

- Cyclic-proof search for the full fragment of μCLP
- Lower and upper bounds checking for the full fragment of the quantitative extension of HFL
  - Nice result for a first-order fragment without fixed-point alternation [arXiv 2025]

[arXiv 2025] Kura et al. Ranking and Invariants for Lower-Bound Inference in Quantitative Verification of Probabilistic Programs.