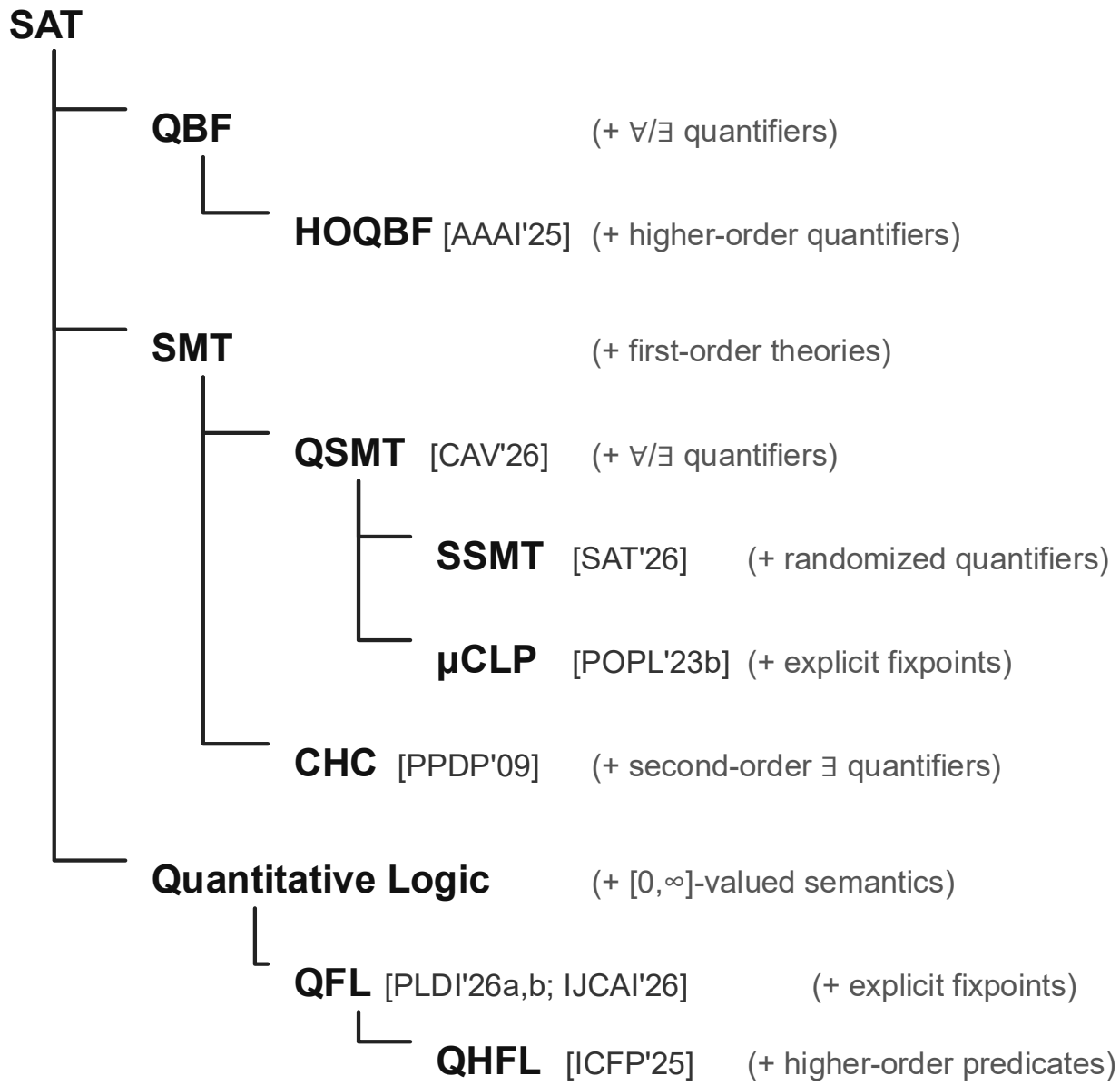
Fixpoint Logics and Applications to Software Verification

Hiroshi Unno (hiroshi.unno@acm.org)

Research Institute of Electrical Communication,
Tohoku University, Japan

My Research Overview

- Broadly interested in logic-based **symbolic reasoning** for formal verification and synthesis, particularly for software systems
- Explore systematic extensions of general-purpose SAT/SMT-style reasoning frameworks toward richer semantic structures: **quantifiers**, **fixpoints**, **higher-order functions**, and **quantitative semantics**
- Our solvers are available at https://github.com/hiroshi-unno/coar



This talk focuses on how these extensions enable richer forms of verification

Why Symbolic Reasoning Matters

Verification Faces Combinatorial Explosion

- This combinatorial explosion appears everywhere: hardware circuits, communication protocols, software systems, concurrent systems, etc.
  - E.g., 100 Boolean latches already induce 2^{100} states
- Formal verification aims to mathematically prove that the target system satisfies the given specification ***for all possible executions***
- The central challenge of verification: How can we reason about enormous state spaces ***without explicitly enumerating*** all reachable states?

Explicit vs Symbolic Representation

Explicit Representation

Suppose we want to represent: all states except $000 \ldots 000$ of the 100-bit system:

$000 \ldots 001$
$000 \ldots 010$
$000 \ldots 011$
…
$111 \ldots 111$

Need $2^{100} - 1$ states explicitly

Symbolic Representation

Instead, represent the same set by: $x_1 \vee x_2 \vee \cdots \vee x_{100}$

Compact symbolic structures can implicitly represent exponentially many states

Symbolic Boolean Reasoning

Symbolic Representations

- Boolean formulas
- BDDs / ZDDs
- spectral representations
  - e.g. Reed–Muller forms

Boolean reasoning tasks

- equivalence checking
- symbolic model checking
- symbolic search
- synthesis

SAT solving emerged as an extremely successful general-purpose symbolic solving paradigm

SAT Problem and Its Application

SAT problem

$$\exists x_1, \ldots, x_n.\ \phi(x_1, \ldots, x_n)$$

- Boolean variables:
  $x_1, \ldots, x_n \in \{\text{true}, \text{false}\}$
- Boolean formula: ϕ
- Question: Is there an assignment to $x_1, \ldots, x_n$ satisfying ϕ?

Application to bounded model checking of finite-state systems:

$$Init(s_0) \wedge Trans(s_0, s_1) \wedge \cdots \wedge Trans(s_{k-1}, s_k) \wedge Error(s_k)$$

is satisfiable
⇔ an error state is reachable after exactly k transitions

SAT solving made (bounded) verification scalable through symbolic constraint solving

Beyond Pure Boolean Reasoning

SAT succeeded because of:

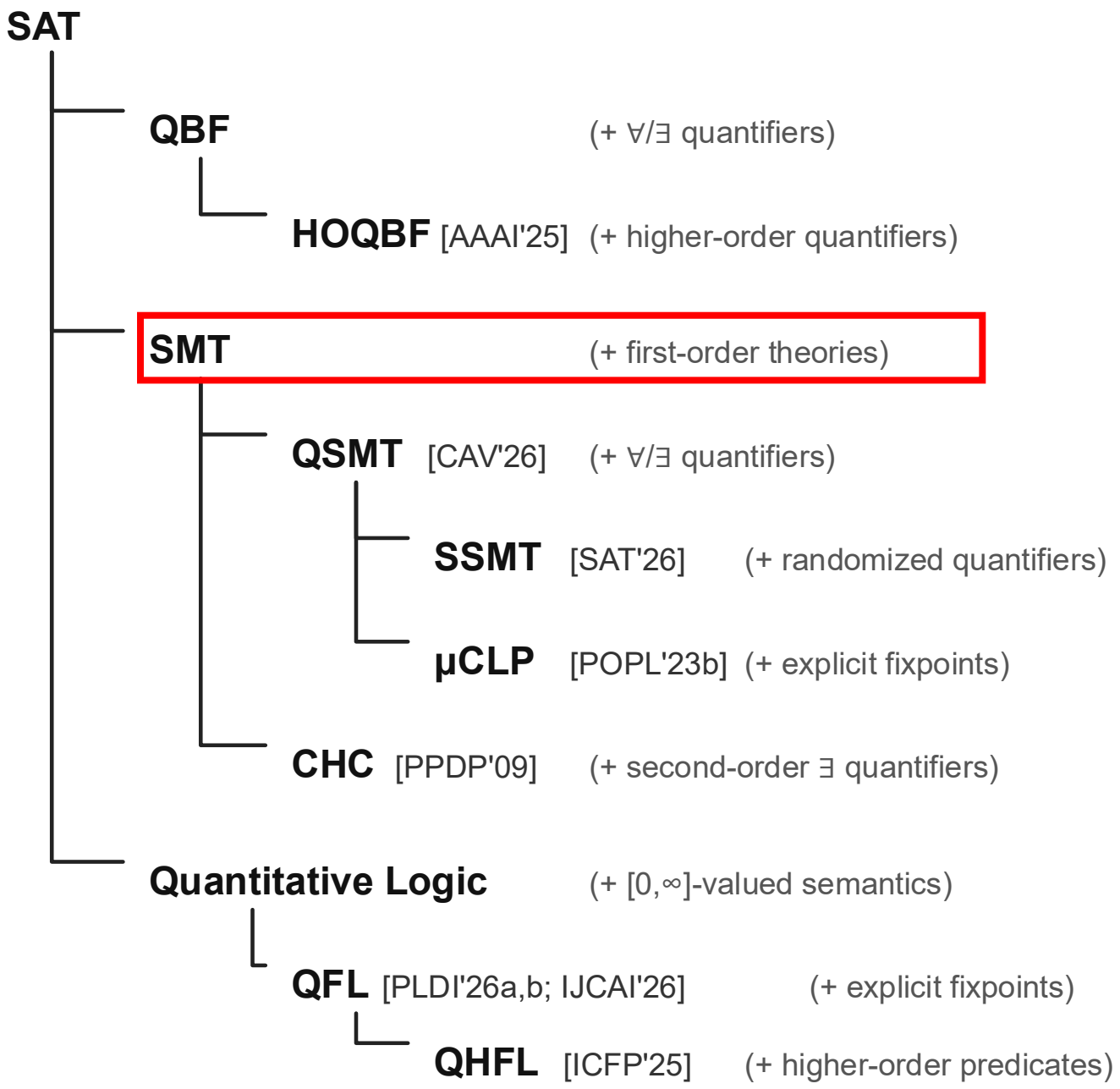
- compact Boolean encodings
- scalable symbolic reasoning
- powerful general-purpose solving

This motivated various extensions of SAT

But modern (particularly software) verification increasingly requires:

- richer semantic domains
- quantification
- higher-order abstraction
- induction (least fixpoint) and co-induction (greatest fixpoint)
- quantitative semantics
  - Probabilities, costs, and expectations

First-Order Theories


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

Satisfiability Modulo Theories (SMT) = SAT + First-Order Theories

- Enables reasoning about integer/real arithmetic, bit-vectors, arrays, lists, algebraic data types, etc.
- Enables compact symbolic representations of **infinitely many program states**, which is essential for software verification

Application to bounded model checking of infinite-state systems:

$$Init(s_0) \wedge Trans(s_0, s_1) \wedge \cdots \wedge Trans(s_{k-1}, s_k) \wedge Error(s_k)$$

is satisfiable
⇔ an error state is reachable after exactly k transitions

Application to deductive verification:

@precond: $x = 1 \wedge n = n_0$

```
while(n != 0) {
   x = n * x;
   n = n - 1; }
```

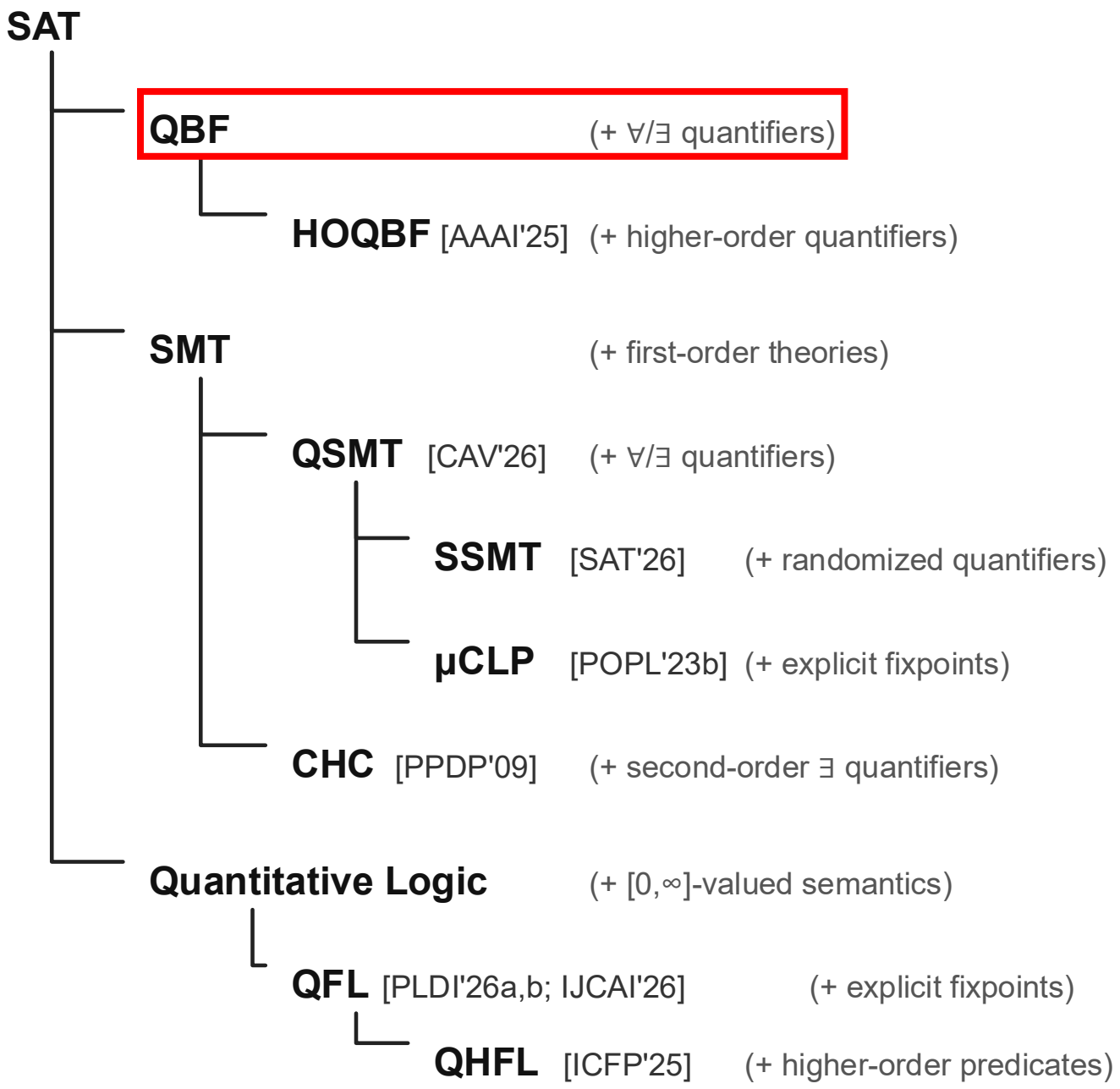
@invariant: $x \cdot n! = n_0!$

@postcond: $x = n_0!$

Generated verification condition:

$$1 \cdot n_0 = n_0! \wedge (n = 0 \wedge x \cdot n! = n_0! \Rightarrow x = n_0!) \wedge (x \cdot n! = n_0! \wedge n \neq 0 \Rightarrow (n \cdot x) \cdot (n-1)! = n_0!)$$

Quantifiers


SAT
QBF (+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT (+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic (+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

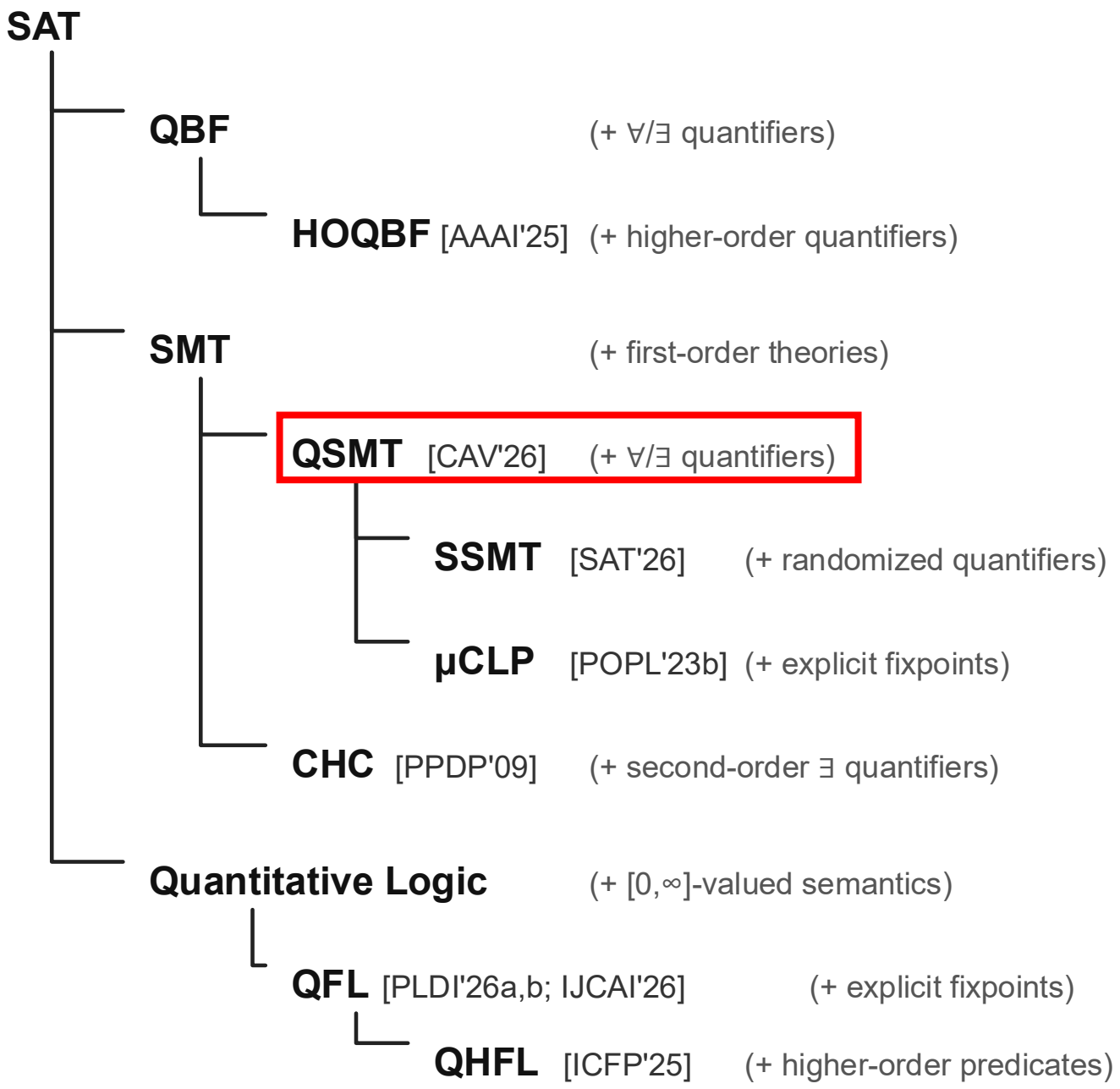
Quantified Boolean Formulas (QBF)
= SAT + ∀/∃ Quantifiers

$\varphi ::= \top \mid \bot \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Leftrightarrow \varphi_2 \mid x \mid \forall x.\varphi \mid \exists x.\varphi$

Def.1 QBF Satisfiability (QSAT)

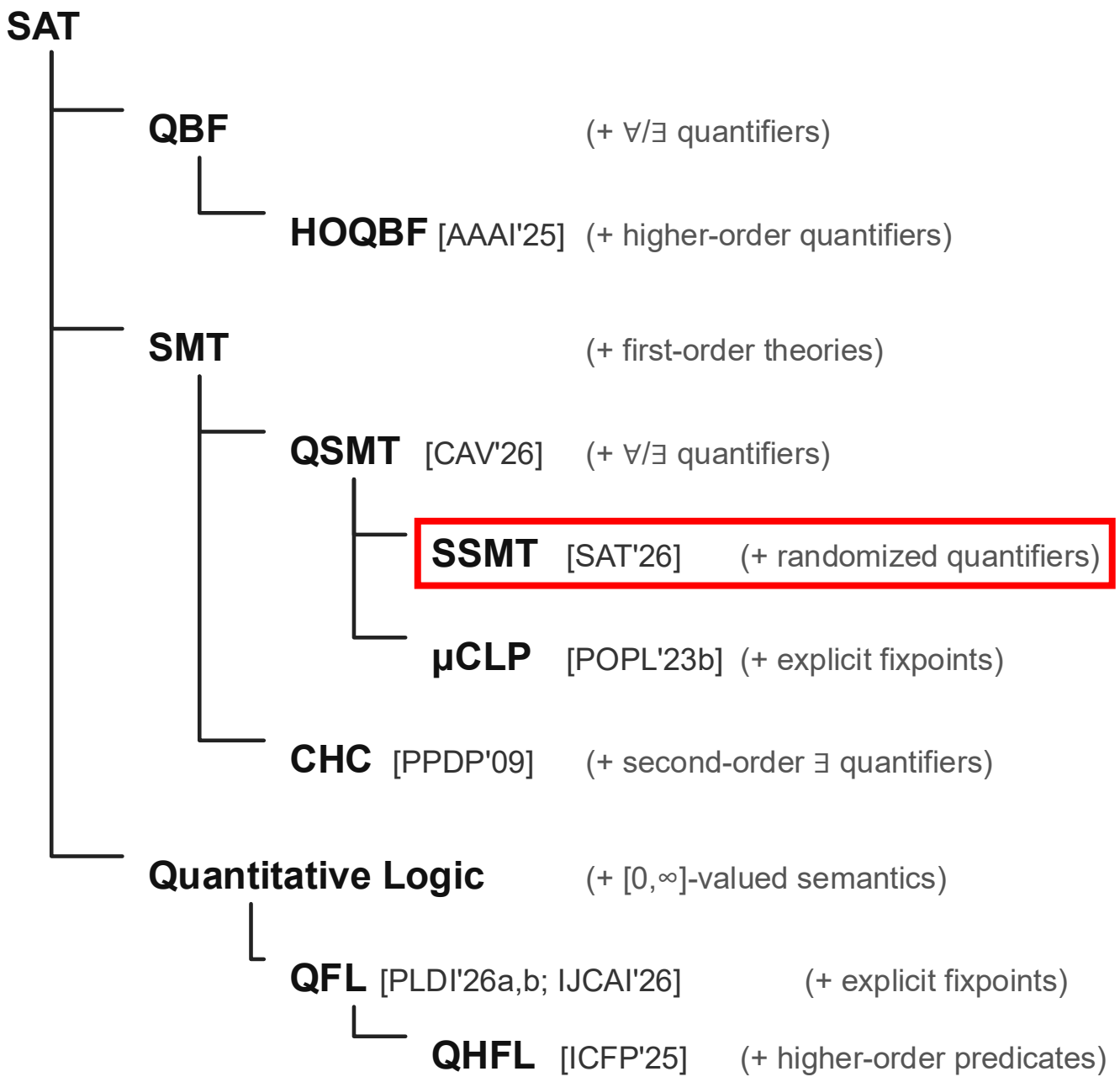
Ask if a closed formula φ is equivalent to $\top$.

- Ex. $\forall x.\exists y.\ x \Leftrightarrow \neg y$ is satisfiable because for any $x \in \{\top, \bot\}$, we can choose $y = \neg x$ to satisfy the quantifier-free part
- Enables symbolic reasoning about Boolean strategic interaction between a system (represented by ∃) and its adversarial environment (represented by ∀)
- Applications: reactive synthesis, adversarial planning, and game solving


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

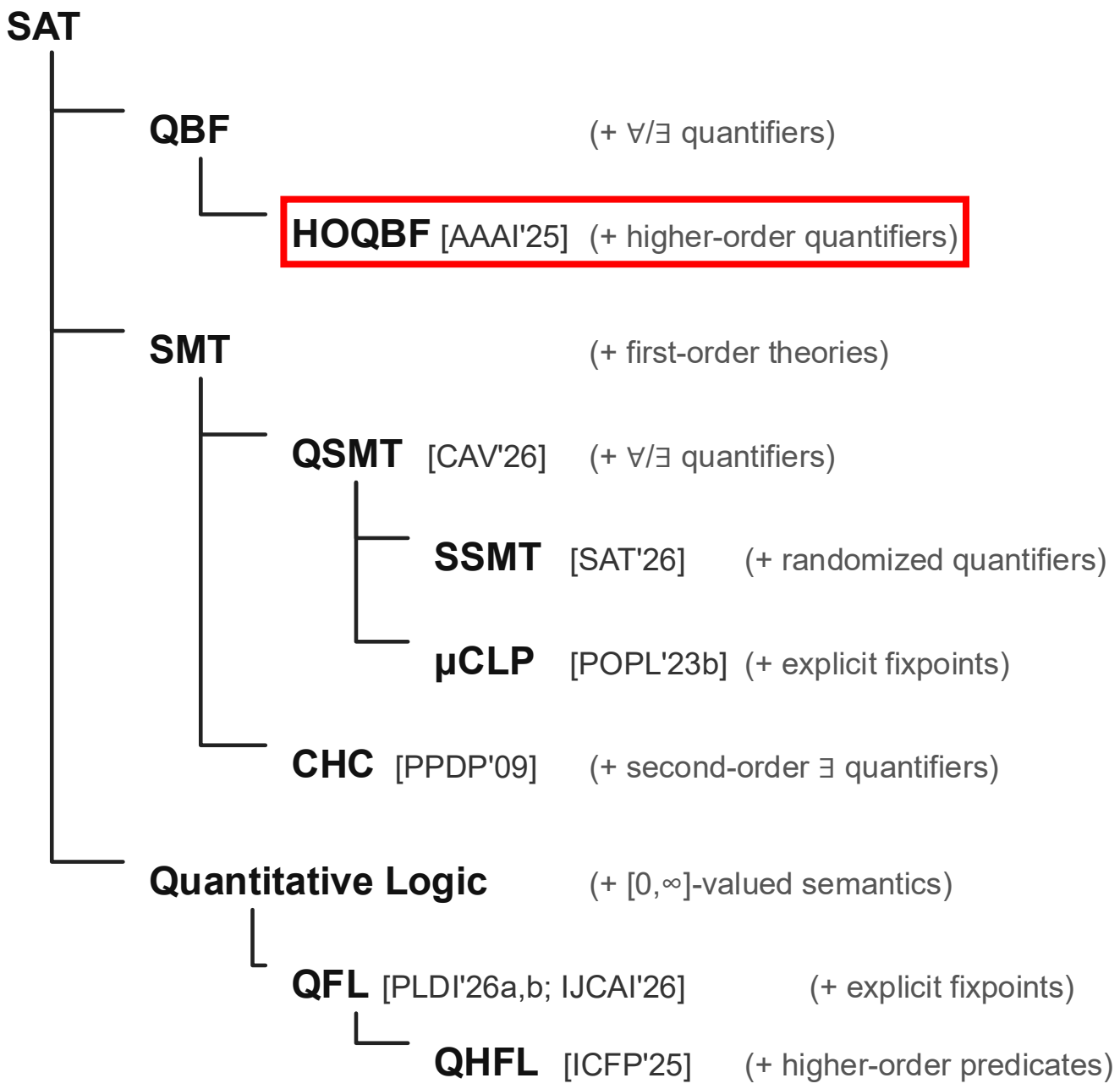
Quantified SMT (QSMT)
= SMT + ∀/∃ Quantifiers

- Ex. $\forall x \in \mathbb{Z}. \exists y \in \mathbb{Z}. y > x$ is satisfiable because for any $x \in \mathbb{Z}$, we can choose $y = x + 1$ to satisfy the quantifier-free part
- Enables symbolic reasoning about strategic interaction over rich theories (e.g., arithmetic, arrays, bit-vectors) between an ***infinite-state*** system (represented by ∃) and its adversarial ***infinite-state*** environment (represented by ∀)
- Applications: reactive synthesis, adversarial planning, and game solving


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

Stochastic SMT (SSMT)
= QSMT + Randomized Quantifiers

- Ex. $\mathcal{R}\ x \in [0,1].\exists y \in \left[0,\frac{1}{2}\right].\ y > x$
  - A random variable x is sampled uniformly from $[0,1]$
  - The ***satisfiability probability*** of the formula is $\frac{1}{2}$ because there exists $y \in \left[0,\frac{1}{2}\right]$ satisfying $y > x$ only when $x < \frac{1}{2}$
- Application: (bounded) verification of hybrid systems with both probabilistic and non-deterministic behaviors


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

Higher-Order QBF (HOQBF)
= QBF + Higher-Order ∀/∃ Quantifiers

$$\varphi ::= \top \mid \bot \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Leftrightarrow \varphi_2$$
$$\mid \forall x^A.\varphi \mid \exists x^A.\varphi \mid x_0(x_1, \ldots, x_n)$$
$$A ::= \mathrm{bool} \mid A_1 \to A_2$$

function type

Apply $x_0{}^{A_1 \to \ldots \to A_n \to \mathrm{bool}}$ to arguments $x_1^{A_1}, \ldots, x_n^{A_n}$

Def.2 HOQBF Satisfiability (HOSAT)

Ask if a closed formula φ is equivalent to $\top$.

Potential applications: memory consistency verification, planning for multi-agent systems, secure system synthesis, and solving quantified bit-vector arithmetic problems

Expressiveness of **HOQBF**

- **HOSAT** is TOWER-complete [Chistikov+'22]
  - Cf. **SAT** is NP-complete and **QSAT** is PSPACE-complete
- **QBF** is a strict fragment where type A is fixed to bool

  Ex.1 $\forall x^{\mathrm{bool}}.\exists y^{\mathrm{bool}}.\forall z^{\mathrm{bool}}.(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$
- **Dependency QBF (DQBF)** and **Second-Order QBF (SOQBF)** [Jiang'23] are strict fragments

  Ex.2 $\forall f^{\mathrm{bool}\to\mathrm{bool}}.\exists g^{\mathrm{bool}\to\mathrm{bool}}.\exists z^{\mathrm{bool}}.\Big(f\big(g(z)\big)\Big) \Leftrightarrow z$

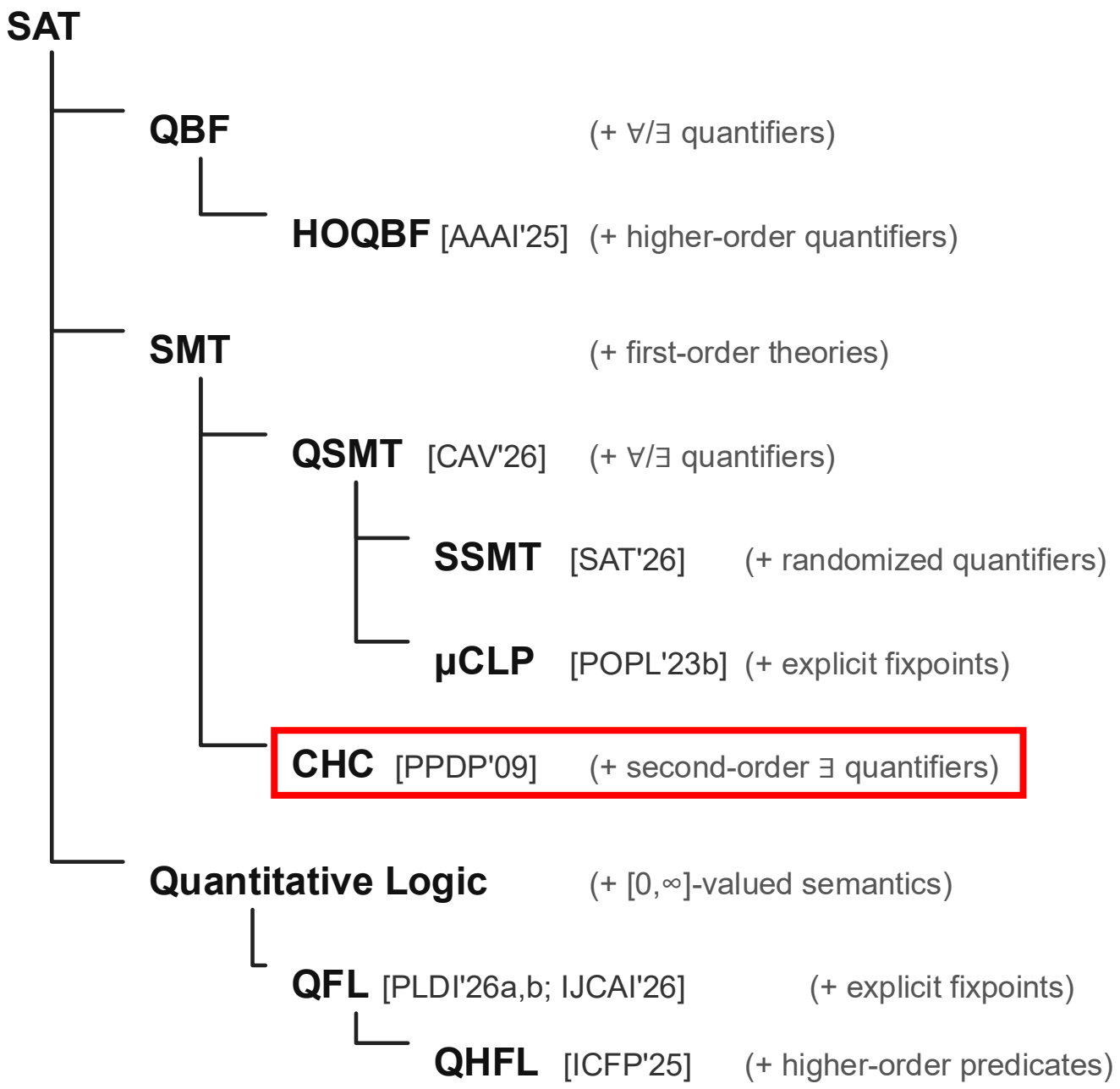
Implementation and Evaluation

- **HOMCSat** implemented using **HorSat2** as **HOMC**

problem	O	V	A	result	time (s)
example1	1	3	2	SAT	0.032
example2	2	3	1	SAT	0.024
aaai23_ex1	2	5	4	SAT	0.437
aaai23_ex2	2	7	4	UNSAT	12.274
sym-asym-b	2	5	0	UNSAT	0.190
sym-asym-bb	2	9	0	T/O	N/A
sym-asym-id	2	7	1	SAT	0.070
SB-theorem	2	16	3	SAT	0.152
left-total	2	3	1	SAT	0.035
right-total	2	3	1	UNSAT	0.035
left-uniq	2	4	0	UNSAT	0.035
right-uniq	2	4	0	SAT	0.032
refl-cl-exist	2	11	2	SAT	3.415
refl-cl-uniq	2	23	2	SAT	124.717
sym-cl-exist	2	13	2	SAT	7.717
sym-cl-uniq	2	27	2	M/O	N/A
tran-cl-exist	2	15	2	SAT	30.286
tran-cl-uniq	2	31	2	M/O	N/A
f_h-neq-g_h	3	3	2	SAT	0.111
cps-arity1	3	5	4	SAT	11.785
cps-arity2	4	6	4	M/O	N/A

O: order
V: number of quantified variables
A: number of quantifier alternations

- Successfully solved **HOQBFs** that express typical properties of Boolean functions and binary relations
- Revealed limitations and possible future direction of **HOMC**
  - **HorSat2** struggled to scale when handling cases with **extensive branching** due to numerous quantifiers, a situation uncommon in human-written programs where **HorSat2** has been applied so far


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

CHCs: Constrained Horn Clauses

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

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

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

  where $X_0, X_1 \ldots, X_m$ are unknown predicate variables,
  $\overrightarrow{t_0}, \ldots, \overrightarrow{t_m}$ 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 $\rho \vDash \bigwedge \mathcal{C}$

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

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

Verification intermediary independent of particular target and method ☺

Constraint Generation

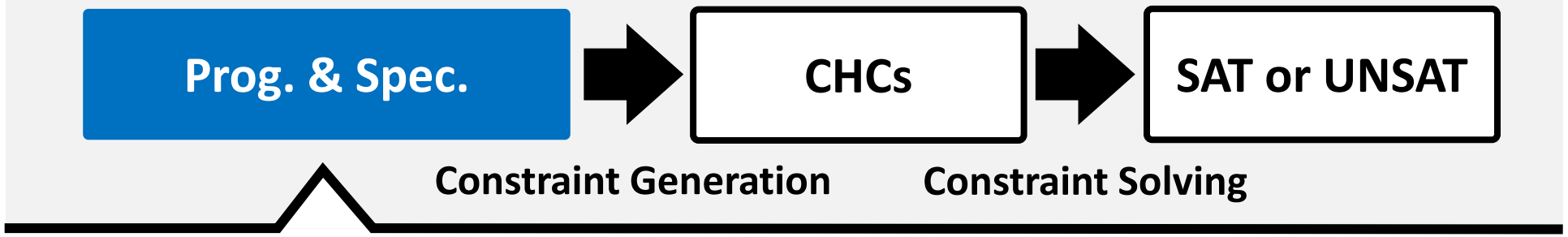
RustHorn [Matsushita+'20,...]
JayHorn [Kahsai+'16,...]
SeaHorn [Gurfinkel+'15,...]
RCaml [PPDP09,...]

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**



Example Program and *Partial Correctness* Specification:

Pre-condition

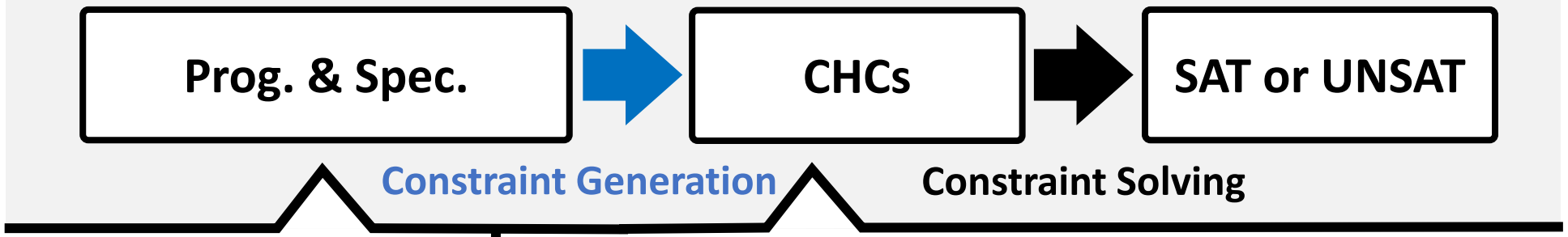
$\{x = x_0\}$
$y = 0;$
while $x \neq 0$ **do**
$y = y + 1;$
$x = x - 1$
done

Post-condition

$\{y = x_0\}$

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



Input:

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

Output $\mathcal{C}$:

represents a ***loop invariant***

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

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

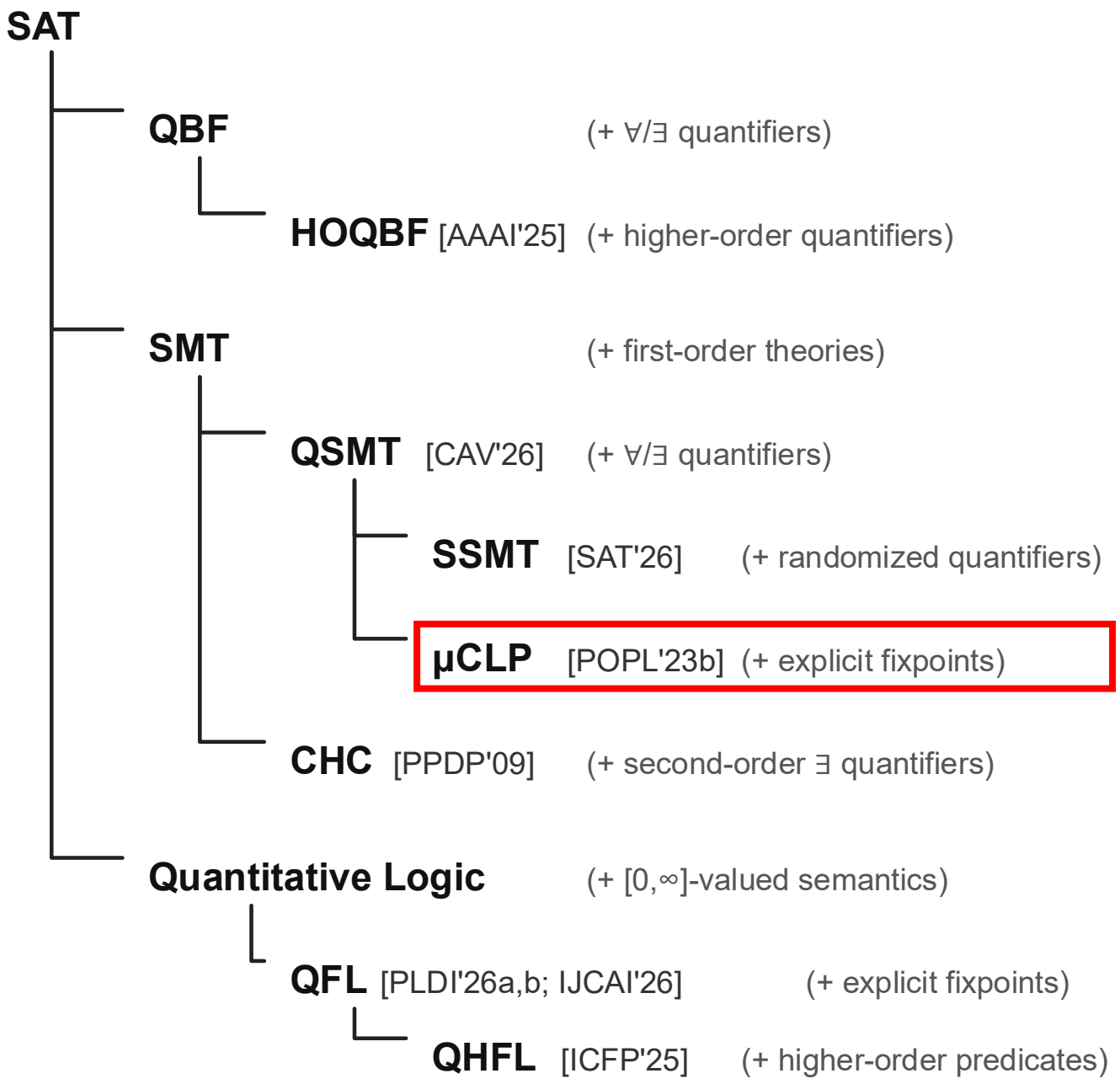
③ $y = x_0 \Leftarrow \underline{S}(x_0, x, y) \wedge x = 0$

$\mathcal{C}$ 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

Least and Greatest Fixpoints


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

μCLP: 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 Predicates of μCLP

- Equational form: $X(x) =_{\alpha} x = 0 \vee X(x-1)$
  - Has the least solution $\lambda x.\, x \geq 0$ (indicated by $\alpha = \mu$)
    - $\mu X.\, \lambda x.\, x = 0 \vee X(x-1)$ is equivalent to $\lambda x.\, x \geq 0$
  - Has the greatest solution $\lambda x.\, \top$ (indicated by $\alpha = \nu$)
    - $\nu X.\, \lambda x.\, x = 0 \vee X(x-1)$ is equivalent to $\lambda x.\, \top$
- Equational form: $X(x) =_{\alpha} x \geq 0 \wedge X(x+1)$
  - Has the least solution $\lambda x.\, \bot$ (indicated by $\alpha = \mu$)
    - $\mu X.\, \lambda x.\, x \geq 0 \wedge X(x+1)$ is equivalent to $\lambda x.\, \bot$
  - Has the greatest solution $\lambda x.\, x \geq 0$ (indicated by $\alpha = \nu$)
    - $\nu X.\, \lambda x.\, x \geq 0 \wedge X(x+1)$ is equivalent to $\lambda x.\, x \geq 0$

Example Formulas of μCLP

$$\left(\mu X.\, \lambda x.\, x = 0 \vee X(x-1)\right)(n)$$
$$\Leftrightarrow \left(\lambda x.\, x = 0 \vee \left(\mu X.\, \lambda x.\, x = 0 \vee X(x-1)\right)(x-1)\right)(n)$$
$$\Leftrightarrow n = 0 \vee \left(\mu X.\, \lambda x.\, x = 0 \vee X(x-1)\right)(n-1)$$
$$\Leftrightarrow n = 0 \vee n - 1 = 0 \vee \left(\mu X.\, \lambda x.\, x = 0 \vee X(x-1)\right)(n-2)$$
$$\Leftrightarrow n = 0 \vee n = 1 \vee \left(\mu X.\, \lambda x.\, x = 0 \vee X(x-1)\right)(n-2)$$
$$\Leftrightarrow n = 0 \vee n = 1 \vee n = 2 \vee \cdots \vee n = k-1 \vee \left(\mu X.\, \lambda x.\, x = 0 \vee X(x-1)\right)(n-k)$$
$$\Leftrightarrow n = 0 \vee n = 1 \vee n = 2 \vee \cdots \Leftrightarrow \exists k \geq 0.\, n = k \Leftrightarrow n \geq 0$$

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

Example Formulas of $\boldsymbol{\mu}$CLP

$$\left(\nu X.\, \lambda x.\, x \geq 0 \wedge X(x+1)\right)(n)$$
$$\Leftrightarrow n \geq 0 \wedge \left(\nu X.\, \lambda x.\, x \geq 0 \wedge X(x+1)\right)(n+1)$$
$$\Leftrightarrow n \geq 0 \wedge n+1 \geq 0 \wedge \cdots \wedge n+k-1 \geq 0 \wedge \left(\nu X.\, \lambda x.\, x \geq 0 \wedge X(x+1)\right)(n+k)$$
$$\Leftrightarrow n \geq 0 \wedge n+1 \geq 0 \wedge \cdots \Leftrightarrow \forall k \geq 0.\, n+k \geq 0 \Leftrightarrow n \geq 0$$

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

Example Predicates: Encoding Integer Quantifiers

- $\exists x.\phi$ can be encoded as $E(0)$ where

$$E(x) =_{\mu} \phi \vee [-x/x]\phi \vee E(x+1)$$

- $\forall x.\phi$ can be encoded as $A(0)$ where

$$A(x) =_{\nu} \phi \wedge [-x/x]\phi \wedge A(x+1)$$

Example Predicates: Fixpoint Alternation

- Equational form: X where $X =_\nu X \wedge Y$
  $Y =_\mu X \vee Y$
  - Equivalent to $\nu X.\, X \wedge (\mu Y.\, X \vee Y)$ and thus to $\nu X.\, X \wedge X$
  - Has the interpretation $\{X \mapsto \top, Y \mapsto \top\}$
- Equational form: Y where $Y =_\mu X \vee Y$
  $X =_\nu X \wedge Y$
  - Equivalent to $\mu Y.\, (\nu X.\, X \wedge Y) \vee Y$ and thus to $\mu Y.\, Y \vee Y$
  - Has the interpretation $\{X \mapsto \bot, Y \mapsto \bot\}$

Example: *Partial Correctness* Verification

Pre-condition

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

$\forall x_0, x, y.\, (R(x_0, x, y) \wedge x = 0 \Rightarrow y = x_0)$ 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

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

Post-condition

$\{y = x_0\}$

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 μCLP:

$\forall x_0, x, y.\, (R(x_0, x, y) \wedge x = 0 \Rightarrow y = x_0)$ 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

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

$\forall x_0, x, y. \big((x = x_0 \wedge y = 0) \Rightarrow S(x_0, x, y)\big)$ where

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

Example: *Total Correctness* Verification

Pre-condition

$[x \neq 0]$

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

  if $x > 0$ **then**

    $x = x - 1$

  else

    $x = x + 1$

done

Post-condition

$[\top]$

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 μCLP:

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

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

Example: *Total Correctness* Verification

Pre-condition

$[x \neq 0]$

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

  if $x > 0$ **then**

    $x = x - 1$

  else

    $x = x + 1$

done

Post-condition

$[\top]$

Tautology

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

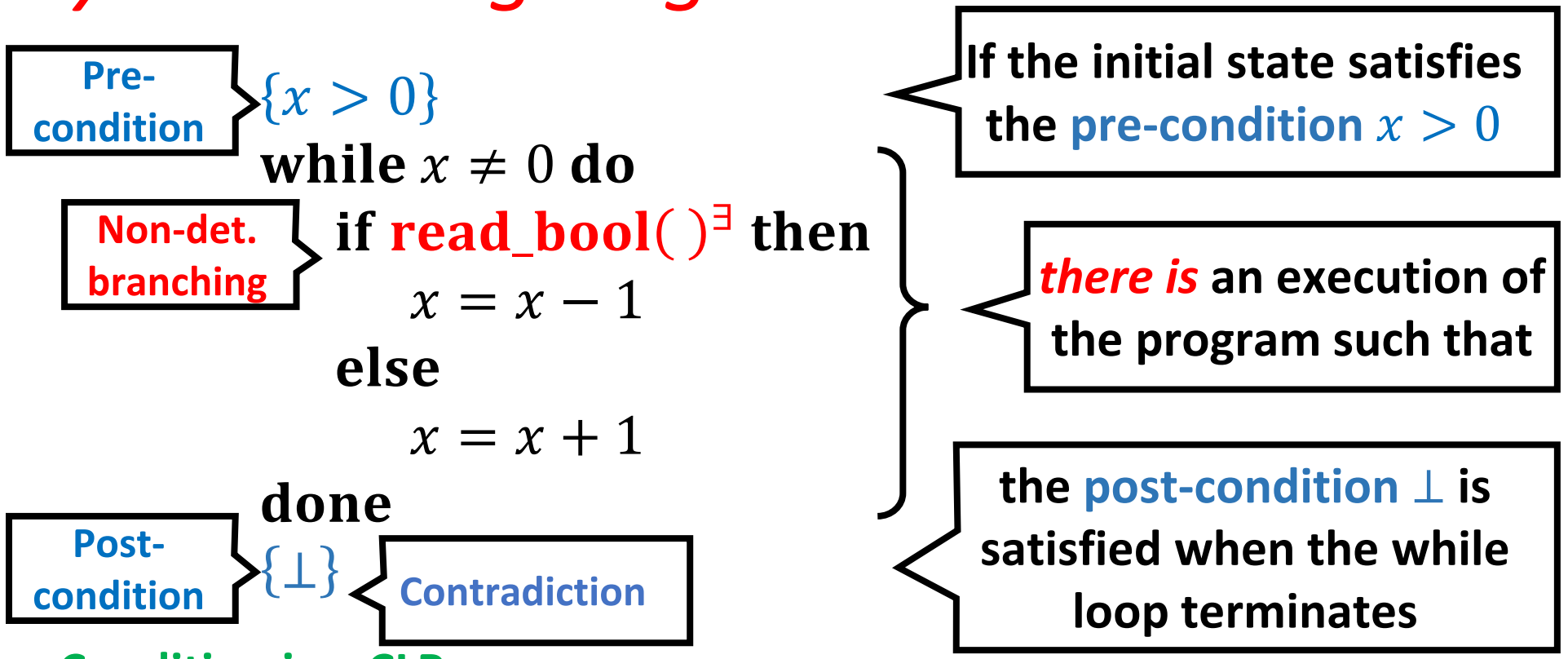
the loop always terminates and the post-condition $\top$ is satisfied by the resulting state

Verification Condition in μCLP:

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

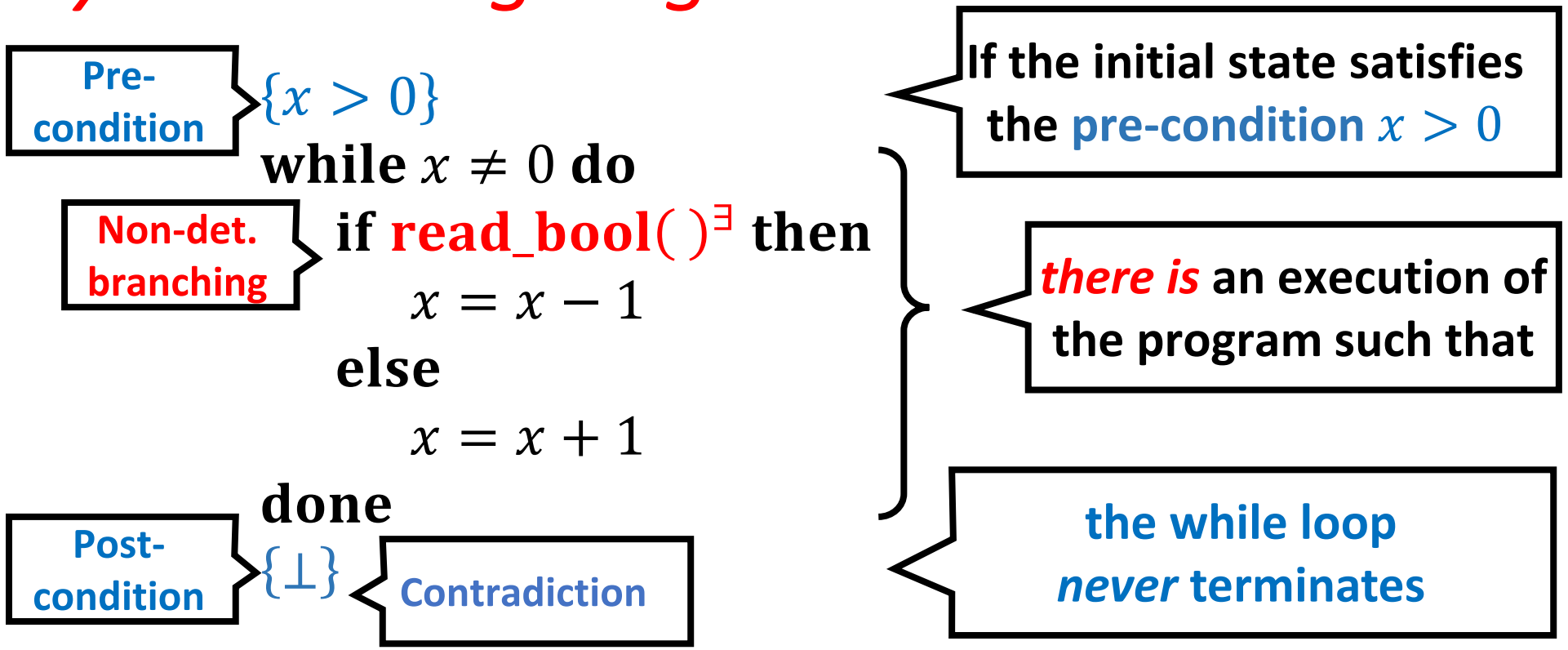
$S(x) =_{\mu} \big(x > 0 \Rightarrow S(x-1)\big) \wedge \big(x < 0 \Rightarrow S(x+1)\big)$

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



Verification Condition in μCLP:

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

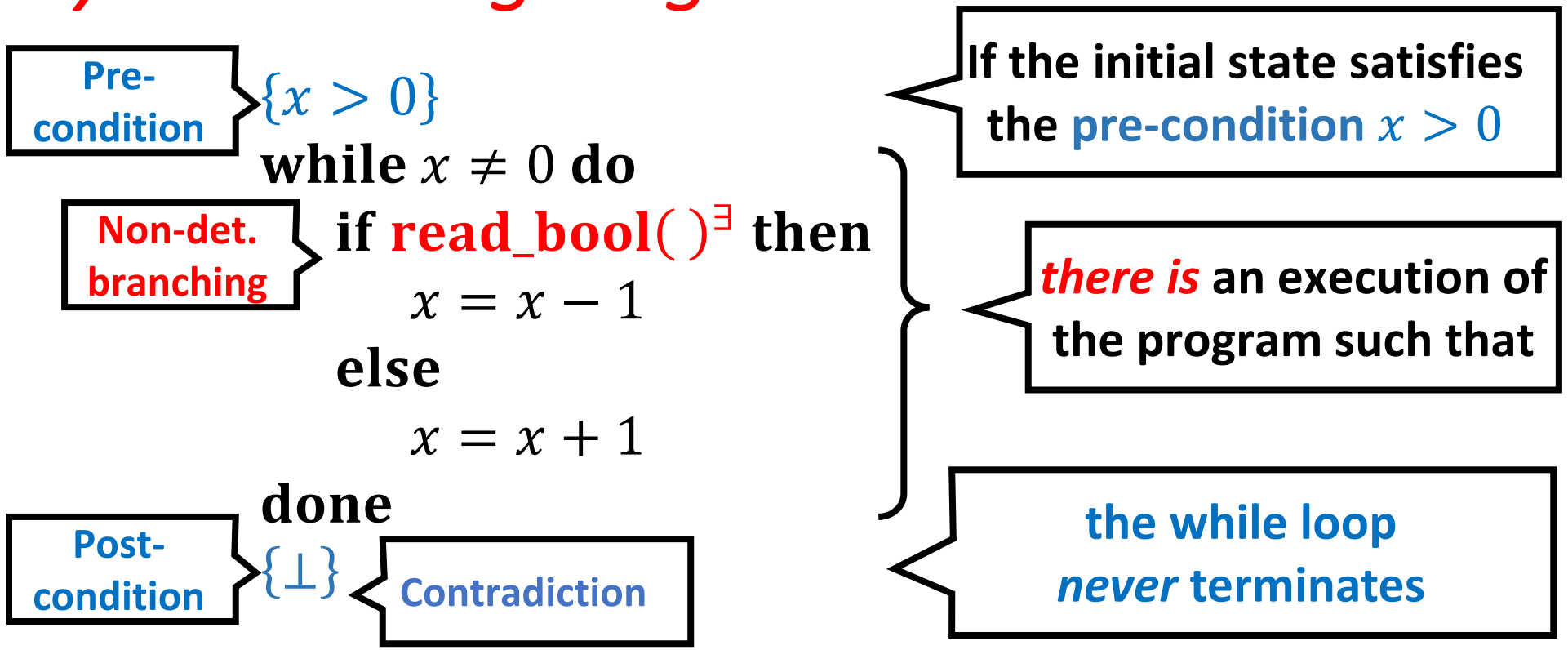


Verification Condition in μCLP:

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

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

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

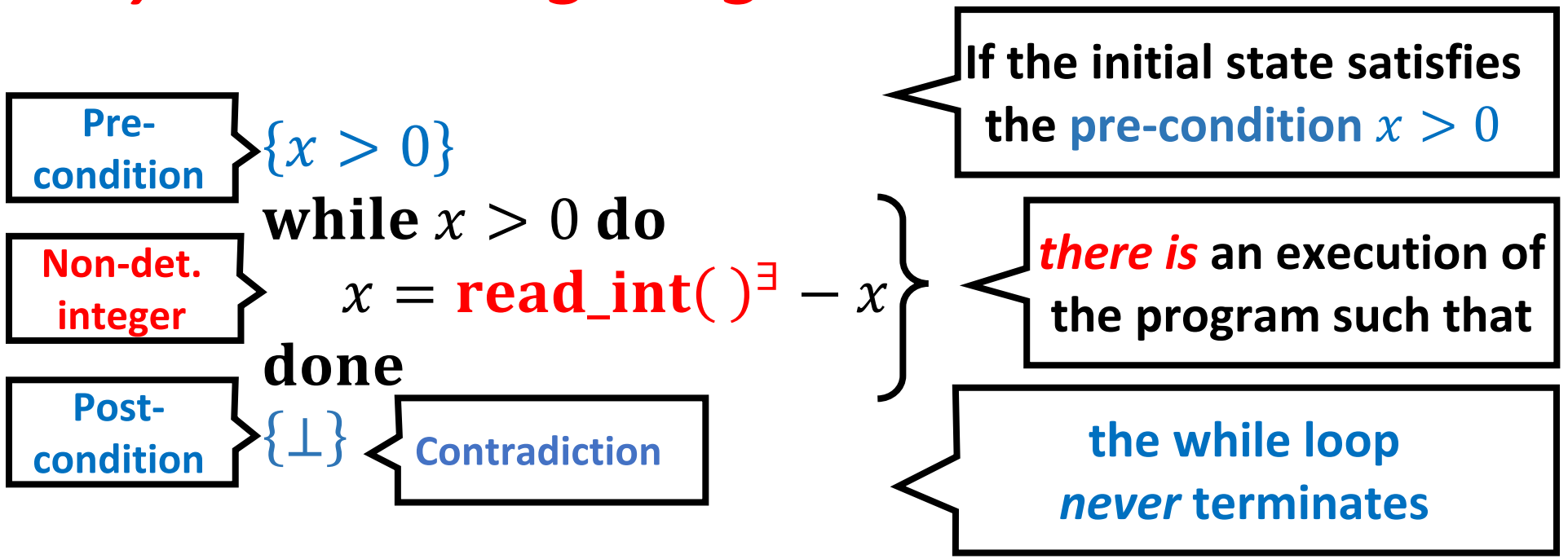


Verification Condition in μCLP:

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

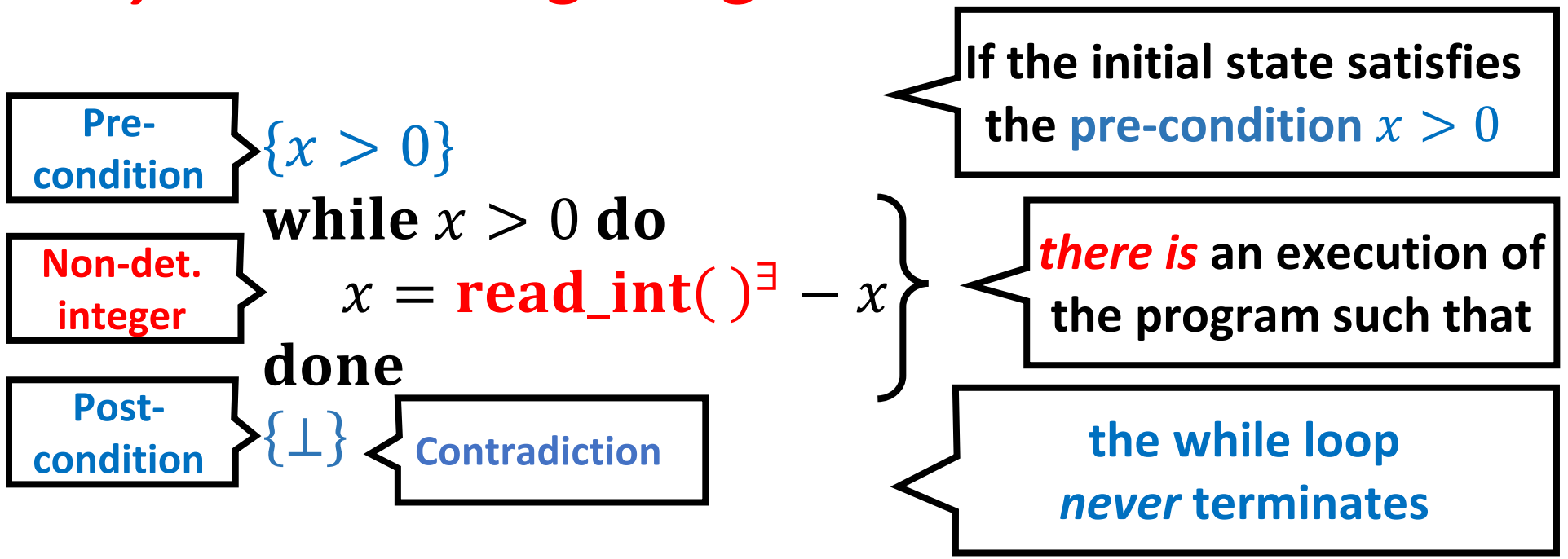


Verification Condition in μCLP:

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



Verification Condition in μCLP:

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

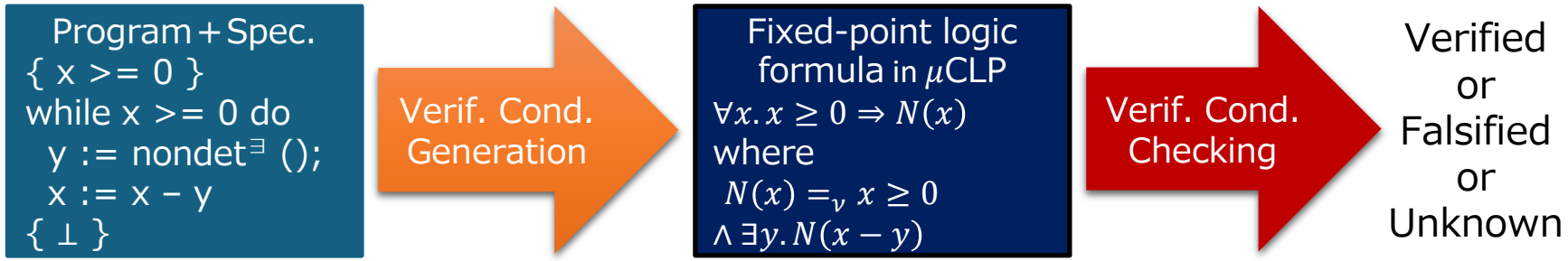
$S(x) =_{\nu} x > 0 \wedge \exists r.S\,(r - x)$

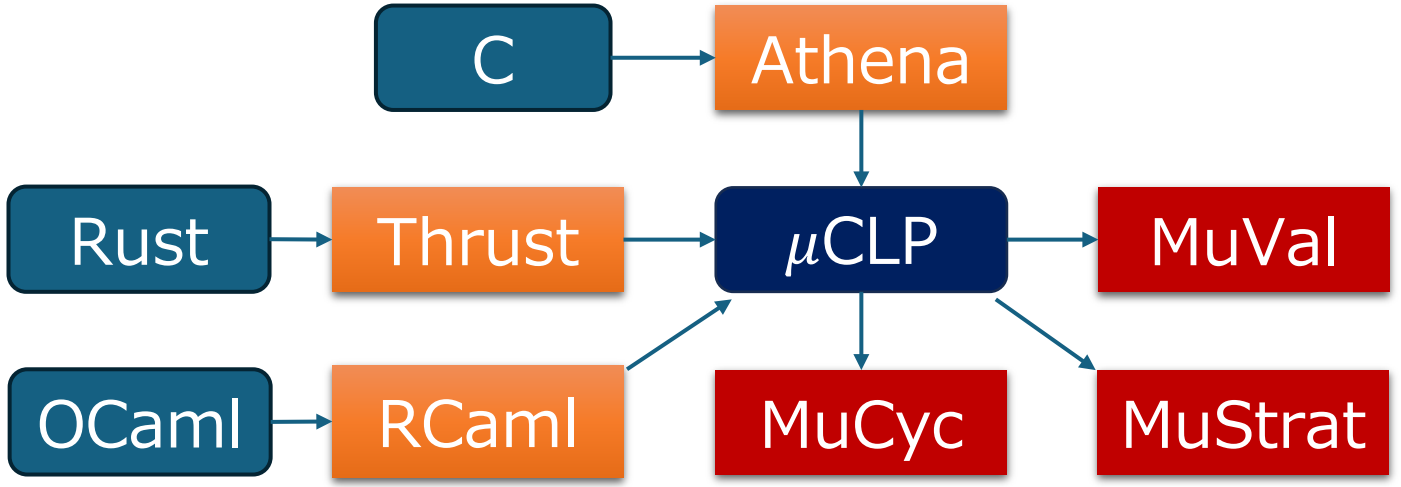
μCLP Encoding of Various Verification Problems

- 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
  - Modal μ-calculus model checking of imperative programs [SAS 2019]
  - Omega-regular model checking of first-order recursive programs [SAS 2019]
  - Modular (non-)termination verification of imperative programs [POPL 2023mod]
  - Omega-regular model checking of labeled transition systems [POPL 2023mod]
  - Modular and value-dependent temporal verification of *higher-order effectful programs* [LICS 2018] and further extensions to *delimited continuations* [POPL 2023aem] and *algebraic effects and handlers* [POPL 2024]

[SAS 2019] Kobayashi et al. Temporal Verification of Programs via First-Order Fixpoint Logic.
[POPL 2023mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[POPL23aem] Sekiyama, Unno. Temporal Verification with Answer-Effect Modification.
[POPL24] Kawamata et al. Answer Refinement Modification: Refinement Type System for Algebraic Effects and Handlers.

Software Verification based on Fixed-Point Logics





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

Frontends: Reduction to Fixed-Point Logic Validity Checking

- **Athena** for C [FSE26]
  - Support bit-precise integers (bit-operations, overflow, etc.), arrays, pointers, and structs
- **Thrust** for Rust [PLDI25]
  - Support mutable borrows in “safe” Rust
- **RCaml** for OCaml […, PPDP09, POPL13, SAS15, POPL18, LICS18, CAV18, POPL23aem, POPL24, POPL25ate, …]
  - Support algebraic effects & handlers, nondeterministic & probabilistic choices

[PPDP09] Unno, Kobayashi. Dependent Type Inference with Interpolants.
[POPL13] Unno et al. Automating Relatively Complete Verification of Higher-Order Functional Programs.
[SAS15] Hashimoto, Unno. Refinement Type Inference via Horn Constraint Optimization.
[POPL18] Unno et al. Relatively Complete Refinement Type System for Verification of Higher-Order Non-deterministic Programs.
[LICS18] Nanjo et al. A Fixpoint Logic and Dependent Effects for Temporal Property Verification
[CAV18] Satake, Unno. Propositional Dynamic Logic for Higher-Order Functional Programs
[POPL23aem] Sekiyama, Unno. Temporal Verification with Answer-Effect Modification.
[POPL24] Kawamata et al. Answer Refinement Modification: Refinement Type System for Algebraic Effects and Handlers.
[POPL25ate] Sekiyama, Unno. Algebraic Temporal Effects: Temporal Verification of Recursively Typed Higher-Order Programs.
[PLDI25] Ogawa et al. Thrust: A Prophecy-based Refinement Type System for Rust.
[FSE26] Fathi et al. Sound Termination and Non-Termination Analysis of C Programs with Bit-Precise Bounded Semantics and Advanced Constructs.

Backends: Complementary Approaches to Fixed-Point Logic Validity Checking

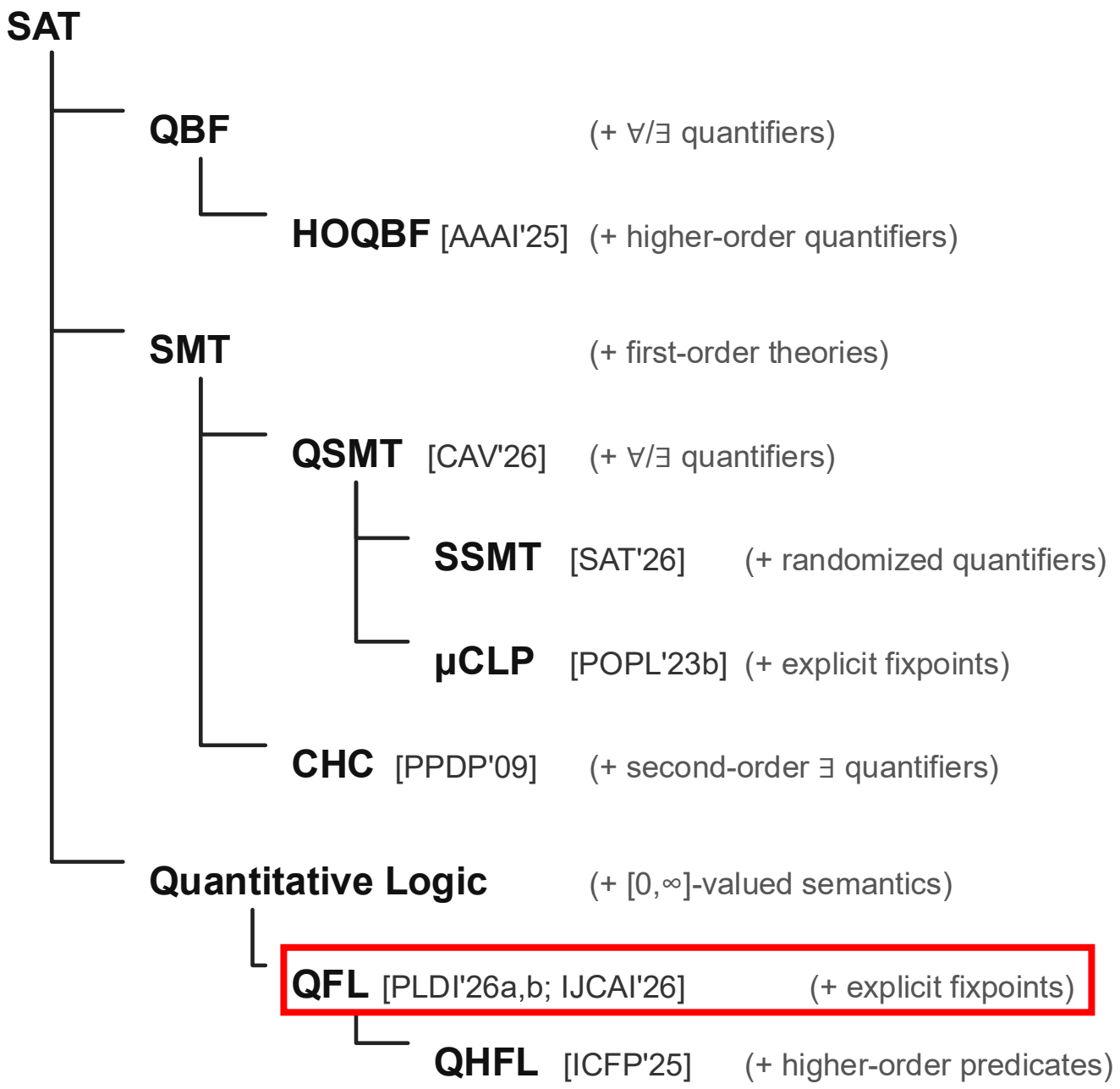
1. Reduction to ***predicate-constraint-solving problems***
   - **MuVal** for μCLP [AAAI20, CAV21tb, CAV21dt, POPL23mod, …]
2. Reduction to ***cyclic-proof-search problems***
   - **MuCyc** for μCLP [CAV17, POPL22, PLDI24, …]
3. Reduction to ***game-solving problems***
   - **MuStrat** for μCLP [POPL25, …]

[CAV17] Unno et al. Automating Induction for Solving Horn Clauses.
[AAAI20] Satake et al. Probabilistic Inference for Predicate Constraint Satisfaction.
[CAV21tb] Unno et al. Constraint-based Relational Verification.
[CAV21dt] Kura et al. Decision Tree Learning in CEGIS-Based Termination Analysis.
[POPL22] Tsukada, Unno. Software Model-Checking as Cyclic-Proof Search.
[POPL23mod] Unno et al. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification.
[PLDI24] Tsukada, Unno. Inductive Approach to Spacer.
[ICFP24] Kura, Unno. Automated Verification of Higher-Order Probabilistic Programs via a Dependent Refinement Type System.
[POPL25] Tsukada et al. A Primal-Dual Perspective on Program Verification Algorithms.

MuVal: A Fixpoint Logic Validity Checker based on Predicate Constraint Solving

- POPL 2023 Distinguished Paper Award (within Top 10% of accepted papers)
- Winner of the (non-)termination verification competition termCOMP 2024 (Integer Transition Systems Track) and termCOMP 2025 (C Track)
- Highly expressive
  - Support modal μ-calculus model checking of infinite-state programs (written in C)
  - Widely used as a backend solver for various verification and synthesis problems:
    - [POPL24] Heim, Dimitrova. Solving Infinite-State Games via Acceleration.
    - [CAV24] Schmuck et al. Localized Attractor Computations for Infinite-State Games.
    - [POPL25] Heim, Dimitrova. Translation of Temporal Logic for Efficient Infinite-State Reactive Synthesis
    - [CAV25heim] Heim, Dimitrova. Issy: A Comprehensive Tool for Specification and Synthesis of Infinite-State Reactive Systems.
    - [CAV25li] Li et al. SPROUT: A Verifier for Symbolic Multiparty Protocols.
    - [PLDI25] Murphy et al. Verifying Solutions to Semantics-Guided Synthesis Problems.
    - [OOPSLA25] Li et al. Characterizing Implementability of Global Protocols with Infinite States and Data.
    - [TACAS26] Heim, Dimitrova. Modular Attractor Acceleration in Infinite-State Games.
    - [PLDI26] Li et al. Implementability of Global Distributed Protocols Modulo Network Architectures.

Quantitative Semantics


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25]
(+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26]
(+ ∀/∃ quantifiers)
SSMT [SAT'26]
(+ randomized quantifiers)
μCLP [POPL'23b]
(+ explicit fixpoints)
CHC [PPDP'09]
(+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26]
(+ explicit fixpoints)
QHFL [ICFP'25]
(+ higher-order predicates)

Probabilistic Verification

- Target Programs
  - Probabilistic programs = while-language + randomness
- Quantitative Properties:
  - Termination probability (in $[0,1]$)
  - Expected runtime (in $[0,\infty]$)
  - Weakest pre-expectation (in $[0,\infty]$)

The expected value of a $[0,\infty]$-valued function f after running a program, expressed as a function of the initial state

Problem

How do we verify lower and upper bounds of these properties?

Example: Weakest pre-expectation

Program: Random walk on $\mathbb{Z}$

```
c = 0;
while (x > 0) {
        if flip(2/3) {
                x = x − 1;
        } else {
                x = x + 1;
        }
        c = c + 1;
}
```

Biased towards decreasing x

Weakest pre-expectation of c

(The expected value of c after running the program)

initial value of x

$$\begin{cases} 3x_0 & \text{if } x_0 > 0 \\ 0 & \text{if } x_0 \leq 0 \end{cases}$$

Expected Value as a Least Fixed Point in a Quantitative Fixed-point Logic (QFL)

Fact

The expected value after program execution can be characterized as a least fixed point in QFL

Let S be the set of the states of the program

The expected value can be expressed using $K : (S \to [0, \infty]) \to (S \to [0, \infty])$:

- The least fixed point μK
- The least solution of a fixed-point equation $X =_{\mu} K(X)$

μ means the least fixed point

Example: Expected Value as a Least Fixed Point

What is the expected value of c after running P?

Program P

X
```
c = 0; x = 10;
while (x > 0) {
    while flip(1/4) {        } Y
        x = x + 1;
    }
    x = x - 1; c = c + 1;
}
```

The Expected Value of c in QFL

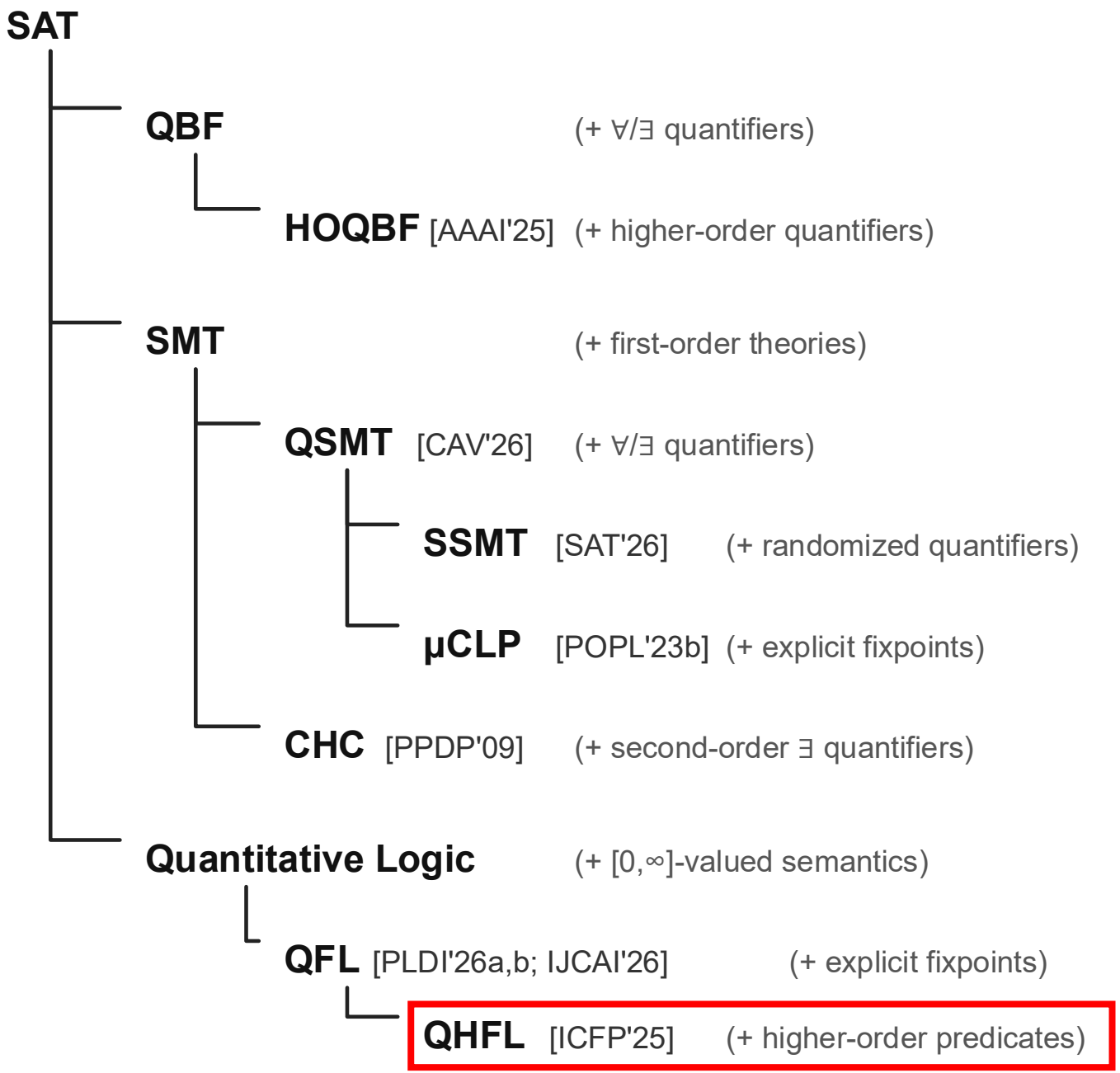
where

$$X(0,10)$$

the initial state of the program is $c = 0$ and $x = 10$

$$X(c,x) =_\mu \textbf{if } x > 0 \textbf{ then } Y(c,x) \textbf{ else } c$$

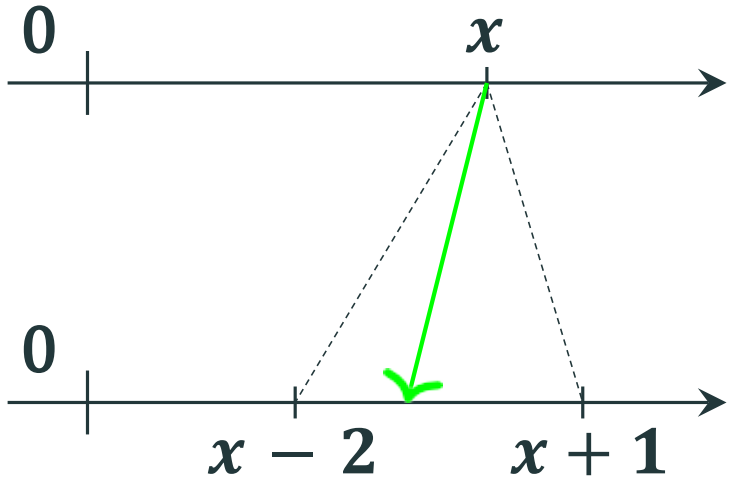
$$Y(c,x) =_\mu \frac{1}{4}Y(c,x+1) + \frac{3}{4}X(c+1,x-1)$$


SAT
QBF
(+ ∀/∃ quantifiers)
HOQBF [AAAI'25] (+ higher-order quantifiers)
SMT
(+ first-order theories)
QSMT [CAV'26] (+ ∀/∃ quantifiers)
SSMT [SAT'26] (+ randomized quantifiers)
μCLP [POPL'23b] (+ explicit fixpoints)
CHC [PPDP'09] (+ second-order ∃ quantifiers)
Quantitative Logic
(+ [0,∞]-valued semantics)
QFL [PLDI'26a,b; IJCAI'26] (+ explicit fixpoints)
QHFL [ICFP'25] (+ higher-order predicates)

Running Example: Expected Cost Analysis of (Higher-Order) Functional Probabilistic Programs

Program: random walk

$$\textbf{let rec } \mathrm{rw}\ x = \textbf{if } x \geq 0$$
$$\textbf{then } y \leftarrow \textbf{uniform}[0,1];$$
$$\checkmark\ (\mathrm{rw}\ (x + 3 \cdot y - 2))$$
$$\textbf{else } ()$$



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

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

Expected Cost Analysis via CPS Transformation

Functional Probabilistic Program

CPS

QHFL Formula

$$\mathrm{rw} : \mathrm{real} \to \mathrm{unit}$$
$$\textbf{let rec } \mathrm{rw}\ x = \textbf{if } x \geq 0$$
$$\quad \textbf{then } y \leftarrow \textbf{uniform}[0,1]; \left(\mathrm{rw}\ (x + 3 \cdot y - 2)\right)^{\checkmark}$$
$$\quad \textbf{else } ()$$

$$\mathrm{rw}' : \mathrm{real} \to (\mathrm{unit} \to [0,\infty]) \to [0,\infty]$$
$$\mathrm{rw}'\, x\ k =_{\mu} \textbf{if } x \geq 0$$
$$\quad \textbf{then unif } (\lambda y.\, 1 + \mathrm{rw}'(x + 3 \cdot y - 2)\ k)$$
$$\quad \textbf{else } k\ ()$$

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

CPS = Continuation-Passing Style, QHFL = Quantitative Higher-order Fixed-point Logic

Summary

- SAT established a highly successful symbolic Boolean reasoning paradigm
- Modern verification increasingly requires richer logical and semantic structures:
  - ***Quantifiers*** for reasoning about strategic interaction to enable branching-time verification, reactive synthesis, and game solving
  - ***Fixpoints*** to enable (unbounded) safety and liveness verification
  - ***Quantitative semantics*** for reasoning about probabilities, costs, expectations, and other quantitative properties
- General-purpose SAT/SMT-style symbolic reasoning has been extended to these richer settings

Key Challenge and Ongoing/Future Work

- **Key Challenge:** How can we scale symbolic reasoning to increasingly rich logical structures?
- **Ongoing/Future Work**: Establishing a mathematically concise theoretical foundation and solvers for rich logical structures, aiming to achieve **correctness**, **extensibility**, and **efficiency** simultaneously
  - Key theoretical tools include:
    - Fixpoint logic theories
    - Cyclic proof theory
    - Optimization theory (e.g., Lagrangian-based duality)
  - Slides and papers on these results, including materials from EPIT Spring School 2025, are available at: https://www.riec.tohoku.ac.jp/~unno/