First-Order Fixpoint Constraints

Horn Clauses and Beyond for Relational and Temporal Program Verification

Hiroshi Unno (University of Tsukuba, Japan)

part of this is joint work with Tachio Terauchi, Eric Koskinen, Sho Torii, Hiroki Sakamoto, and Yoji Nanjo

My Research: Automated Verification of Higher-Order Functional Programs

- **MoCHi**: Software Model Checker for OCaml
  - Assertion safety verification [PLDI'11, PEPM'13, ESOP'15, TACAS'15]
  - Termination verification [ESOP'14]
  - Non-termination verification [CAV'15]
  - Fair-termination verification [POPL'16]

Based on: higher-order model checking, binary reachability analysis, predicate abstraction, CEGAR based on **recursion-free Constrained Horn Clause (CHC) solving**

- **RCaml**: Refinement Type Checking and Inference System for OCaml
  - Assertion safety verification [FLOPS'08, PPDP'09, POPL'13, POPL'18]
  - (Maximally-weak) precondition inference [SAS'15]
  - Termination and non-termination verification [SAS'15, POPL'18]
  - Relational verification [CAV'17]
  - Temporal safety and liveness verification [LICS'18, CAV'18]

Based on: dependent refinement types and **CHC / fixpoint constraint solving**

This Talk

- **MoCHi**: Software Model Checker for OCaml
  - Assertion safety verification [PLDI'11, PEPM'13, ESOP'15, TACAS'15]
  - Termination verification [ESOP'14]
  - Non-termination verification [CAV'15]
  - Fair-termination verification [POPL'16]

Based on: higher-order model checking, binary reachability analysis, predicate abstraction, CEGAR based on **recursion-free Constrained Horn Clause (CHC) solving**

- **RCaml**: Refinement Type Checking and Inference System for OCaml
  - Assertion safety verification [FLOPS'08, PPDP'09, POPL'13, POPL'18]
  - (Maximally-weak) precondition inference [SAS'15]
  - Termination and non-termination verification [SAS'15, POPL'18]
  - Relational verification [CAV'17]
  - Temporal safety and liveness verification [LICS'18, CAV'18]

Based on: dependent refinement types and **CHC / fixpoint constraint solving**

Outline

1. CHC / Fixpoint Constraints for Program Verification
2. CHC Constraint Solving for Relational Verification
   - Based on [Unno, Torii and Sakamoto, CAV’17]
3. Fixpoint Constraint Solving for Temporal Verification
   - Based on [Nanjo, Unno, Koskinen and Terauchi, LICS’18]

Outline

1. **CHC / Fixpoint Constraints for Program Verification**
2. CHC Constraint Solving for Relational Verification
   - Based on [Unno, Torii and Sakamoto, CAV’17]
3. Fixpoint Constraint Solving for Temporal Verification
   - Based on [Nanjo, Unno, Koskinen and Terauchi, LICS’18]

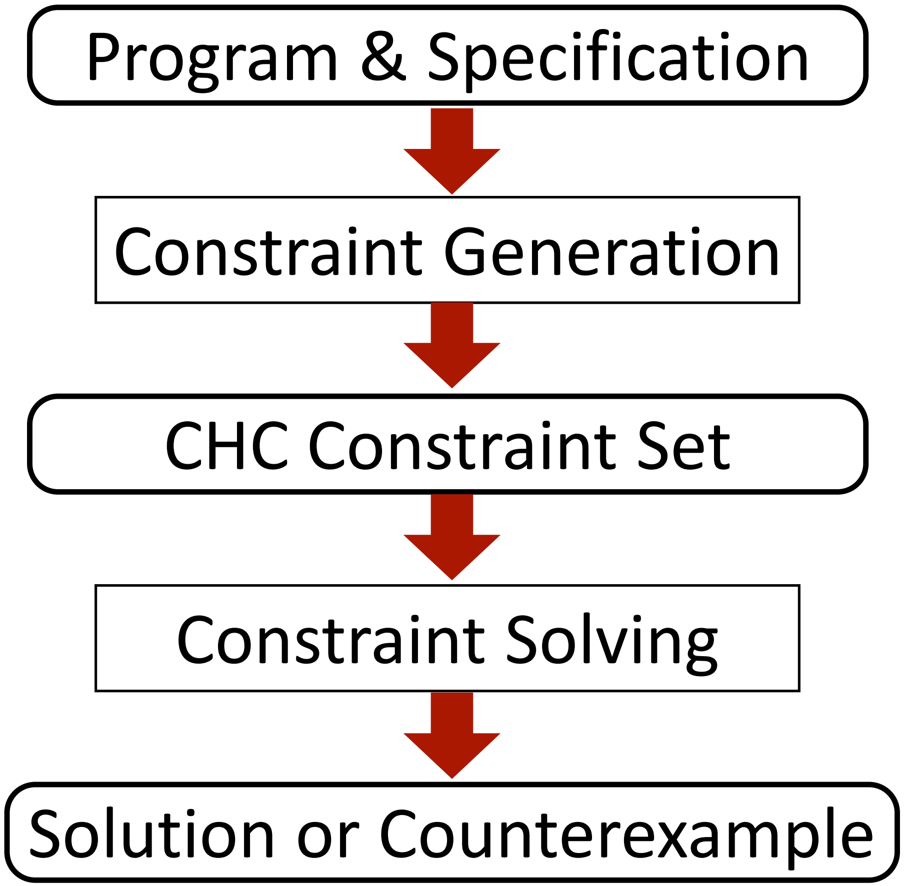
CHC based Program Verification

Verification Problems of Programs in
Various Paradigms (e.g., functional [U.+ ’08, ’09, Rondon+ ’08, …], procedural [Grebenshchikov+ ’12, Gurfinkel+ ’15], object-oriented [Kahsai+ ’16], multi-threaded [Gupta+ ’11]) with
Advanced Language Features (e.g., algebraic data structures, linked data structures, exceptions, higher-order functions) with
Side-Effects (e.g., non-termination, non-determinism, concurrency, assertions, destructive updates)

Reduce

CHC Constraint Solving Problems

Overall Flow of CHC Constraint based Program Verification



Overall Flow of CHC Constraint based Program Verification

Program & Specification

$$\forall x, y.\ x \geq 0 \land y \geq 0 \Rightarrow \text{mult}\ x\ y \geq 0$$

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

```
{- Haskell -}
mult :: Int -> Int -> Int
mult x 0 = 0
mult x y = x + mult x (y - 1)
```

```
/* C */
int mult(int x, int y) {
  int s = 0;
  while(y != 0){
    s += x;
    y--;
  }
  return s;
}
```

Overall Flow of CHC Constraint based Program Verification

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

CHC Constraint Set

Constraint Solving

Solution or Counterexample

Overall Flow of CHC Constraint based Program Verification

Program & Specification

↓

Constraint Generation

↓

CHC Constraint Set

↓

$$P(x, y, r) \equiv r = x \times y$$

ng

↓

Solution or Counterexample

CHC Constraint Set

predicate variables

(CHC constraint sets) $H ::= \{C_1, \ldots, C_n\}$
(CHCs) $C ::= \forall \tilde{x}. (h \Leftarrow P_1(\tilde{t_1}) \wedge \cdots \wedge P_n(\tilde{t_n}) \wedge \phi)$
(heads) $h ::= \bot \mid P(\tilde{t})$
(formulas) $\phi ::= \top \mid \bot \mid A(\tilde{t}) \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$
(terms) $t ::= x \mid f(\tilde{t})$

function symbols of the background theory

predicate symbols of the background theory

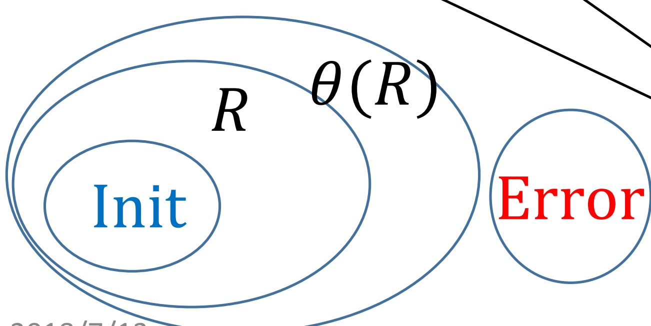
Predicate substitution θ : PVars $\rightarrow$ Preds
is called a *solution* of H if $\vDash \theta(\bigwedge H)$

Overall Flow of CHC Constraint based Program Verification

Initial states: $x = 10$
Program:
 while (x > 0) { x--; }
Error states: $x < 0$

$$R(x) \Leftarrow x = 10$$
$$R(x-1) \Leftarrow R(x) \land x > 0$$
$$\bot \Leftarrow R(x) \land x < 0$$

$$\theta = \{R \mapsto \lambda x. x \geq 0\}$$



Imperative Program & Safety Specification

Constraint Generation

CHC Constraint Set

Constraint Solving

Solvable or Not

First-Order Fixpoint Logic $\mathcal{L}$

- First-order logic extended with least fixpoints (LFPs) and greatest fixpoints (GFPs)

predicate variables

predicate symbols of the background theory

sorts (e.g. $\mathbb{Z}$) of the background theory

$$
\begin{array}{rcl}
(\text{formulas})\ \phi & ::= & \top \mid \bot \mid A(\widetilde{t}) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall x \in \mathcal{S}.\phi \mid \exists x \in \mathcal{S}.\phi \\
& \mid & X(\widetilde{t}) \mid (\mu X(\widetilde{x}\colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \mid (\nu X(\widetilde{x}\colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \\
(\text{terms})\ t & ::= & x \mid f(\widetilde{t})
\end{array}
$$

LFPs
(X occurs only positively in ϕ)

GFPs
(X occurs only positively in ϕ)

function symbols of the background theory

Fixpoint Constraint Solving

- Fixpoint constraint ϕ represented by an $\mathcal{L}$-formula is called ***solvable*** if $\models \phi$

- Generalizes CHC Constraint Solving

$$\begin{array}{l} R(x) \Leftarrow x = 10 \\ R(x-1) \Leftarrow R(x) \land x > 0 \\ \bot \Leftarrow R(x) \land x < 0 \end{array}$$

has a solution

$\Leftrightarrow$

$$\bot \Leftarrow \left(\mu R(x). \begin{array}{c} x = 10 \lor \\ R(x+1) \land x+1 > 0 \end{array}\right)(x) \land x < 0$$

is solvable

Applications to Verification of Liveness and Existential Properties

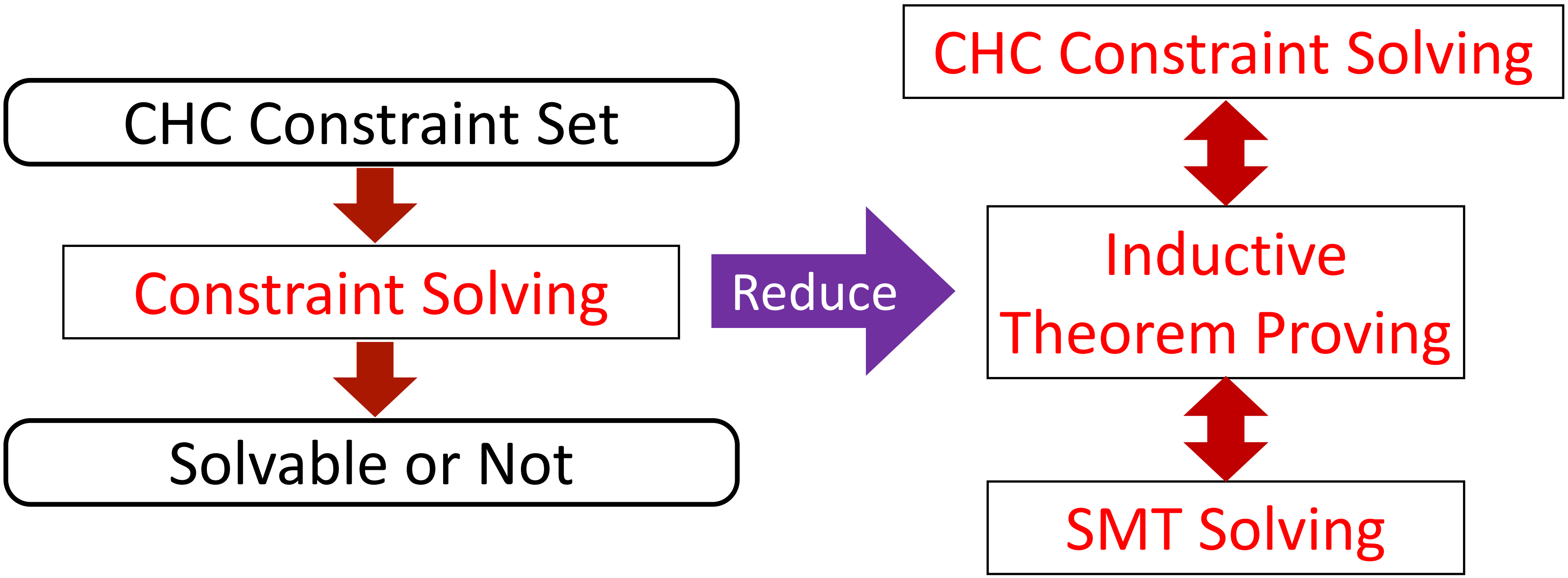
- Safety verification
  - $(\mu \mathrm{Reachable}(\tilde{x}).\,\phi)(\tilde{x}) \Rightarrow \neg \mathrm{Error}(\tilde{x})$
- Termination verification
  - $(\nu \mathrm{Diverging}(\tilde{x}).\,\phi)(\tilde{x}) \Rightarrow \neg \mathrm{Init}(\tilde{x})$
- Non-safety verification
  - $\exists \tilde{x}.\,\mathrm{Error}(\tilde{x}) \wedge (\mu \mathrm{Reachable}(\tilde{x}).\,\phi)(\tilde{x})$
- Non-termination verification
  - $\exists \tilde{x}.\,\mathrm{Init}(\tilde{x}) \wedge (\mu \mathrm{Diverging}(\tilde{x}).\,\phi)(\tilde{x})$

Outline

✓ CHC / Fixpoint Constraints for Program Verification

2. **CHC Constraint Solving for Relational Verification**
   - **Based on [Unno, Torii and Sakamoto, CAV'17]**
3. Fixpoint Constraint Solving for Temporal Verification
   - Based on [Nanjo, Unno, Koskinen and Terauchi, LICS'18]

This Work



- Enable verification of ***relational specifications*** across programs in various paradigms
- Support constraints over any background theories (if the backend SMT solver does)

Relational Specifications

- Specifications that relate
  the inputs and outputs of multiple function calls
  - Equivalence
  - Invertibility
  - Non-interference
  - Associativity
  - Commutativity
  - Distributivity
  - Monotonicity
  - Idempotency
  - …

Overall Flow of CHC Constraint based Program Verification

(Functional) Program & Relational Specification

↓

Constraint Generation

↓

CHC Constraint Set

↓

Constraint Solving

↓

Solvable or Not

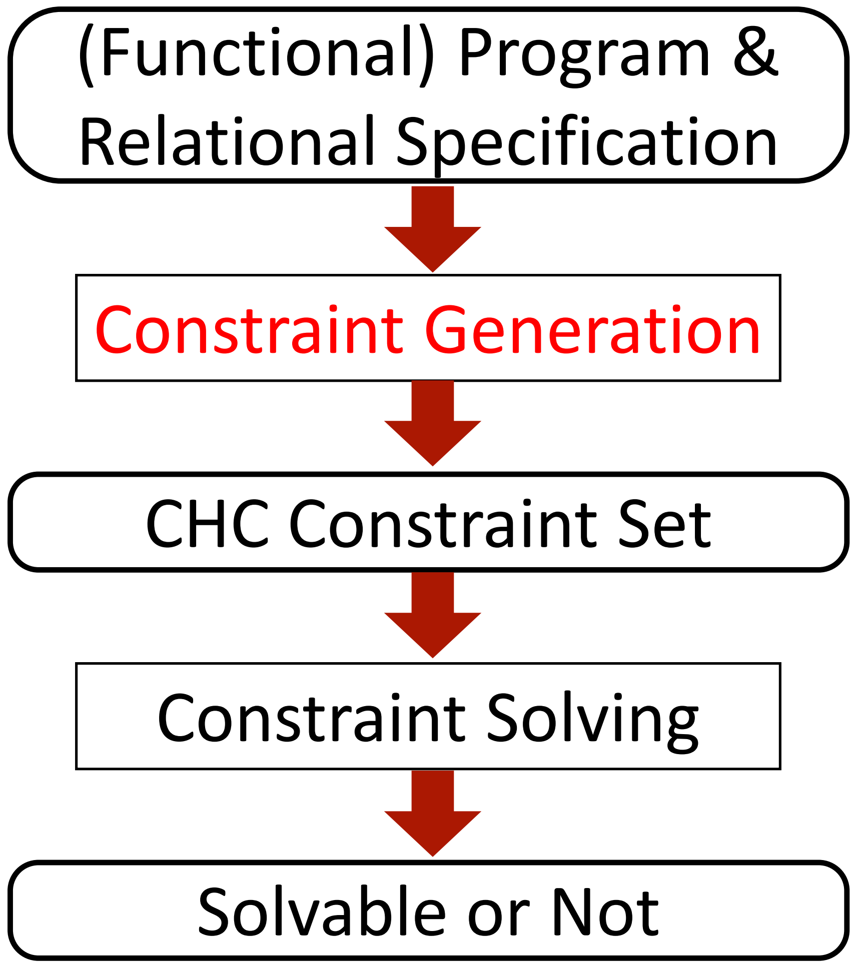
Example: (Functional) Program and Relational Specification

```
(* 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)
```

Overall Flow of CHC Constraint based Program Verification



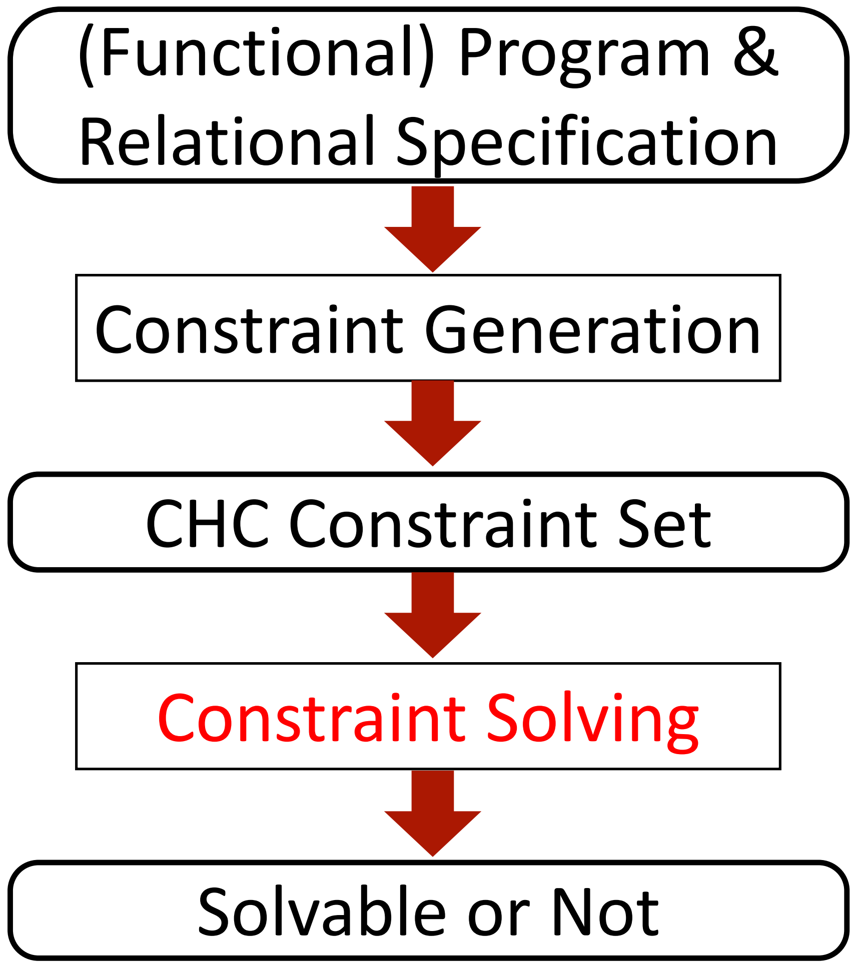
CHC Constraint Generation [U.+ '09]

```
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)
```

$$P(x,0,0)$$
$$P(x,y,x+r) \Leftarrow P(x,y-1,r) \land y \neq 0$$
$$Q(x,0,a,a)$$
$$Q(x,y,a,r) \Leftarrow Q(x,y-1,a+x,r) \land y \neq 0$$
$$s_1 + a = s_2 \Leftarrow P(x,y,s_1) \land Q(x,y,a,s_2)$$

Overall Flow of CHC Constraint based Program Verification



CHC Constraint Solving

- Check the existence of a solution for predicate variables satisfying all the CHC constraints
  - If a solution exists, the original program is guaranteed to satisfy the specification

Example (Non-relational) specification:
let main x y = if x >= 0 && y >= 0 then assert (mult x y >= 0)

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

Solution 1: $P(x,y,r) \equiv x \geq 0 \land y \geq 0 \Rightarrow r \geq 0$

QF-LIA

Solution 2: $P(x,y,r) \equiv r = \underline{x \times y}$

Nonlinear

QF-NIA

Previous Methods for CHC Solving

[U.+ '08,'09, Gupta+ '11, Hoder+ '11,'12, McMillan+ '13, Rümmer+ '13, …] (w/o Predicate Pairing [De Angelis+ '16])

<u>Find a solution expressible in QF-LIA (or QF-LRA)</u>

$$
\boxed{
\begin{array}{l}
P(x, 0, 0) \\
P(x, y, x + r) \Leftarrow P(x, y - 1, r) \land y \neq 0 \\
r \geq 0 \Leftarrow P(x, y, r) \land x \geq 0 \land y \geq 0
\end{array}
}
$$

Solution 1: $P(x, y, r) \equiv x \geq 0 \land y \geq 0 \Rightarrow r \geq 0$

QF-LIA

~~Solution 2:~~ $\cancel{P(x, y, r) \equiv r = x \times y}$

QF-NIA

Example Constraints that Can Not be Solved by Previous Methods

Constraint Solving Fails!

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

$$P(x, y, x + r) \Leftarrow P(x, y$$

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

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

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

QF-NIA

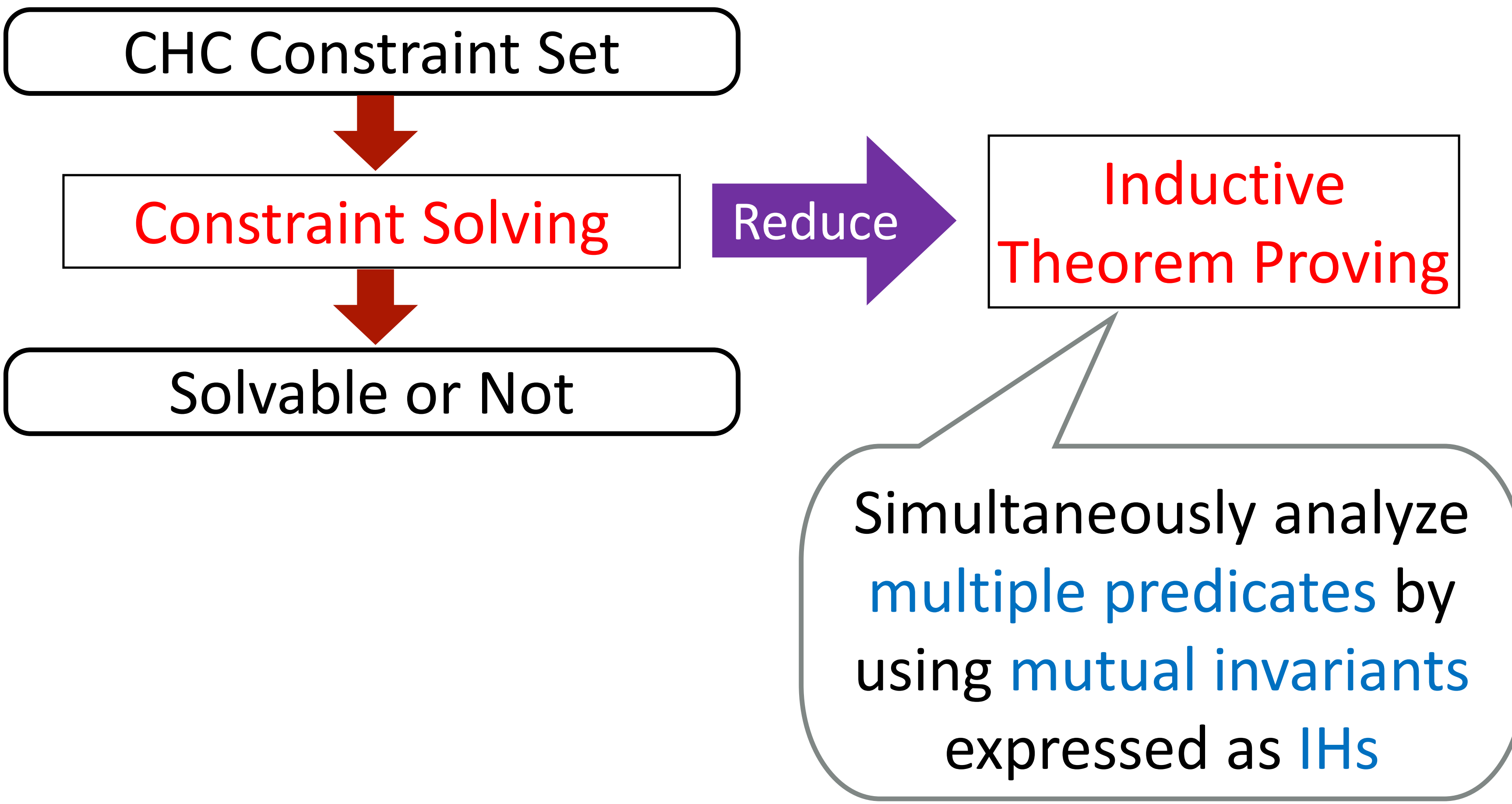
Analyzed separately from Q

Analyzed separately from P

$P(x, y, s_1) \equiv s_1 = x \times y$

$Q(x, y, a, s_2) \equiv s_2 = x \times y + a$

Our Constraint Solving Method



Reduction from Constraint Solving to Inductive Theorem Proving

$$P(x,0,0) \quad P(x,y,x+r) \Leftarrow P(x,y-1,r) \wedge y \neq 0$$
$$Q(x,0,a,a) \quad Q(x,y,a,r) \Leftarrow Q(x,y-1,a+x,r) \wedge y \neq 0$$
$$s_1 + a = s_2 \Leftarrow P(x,y,s_1) \wedge Q(x,y,a,s_2)$$

$$\frac{\models y=0 \wedge r=0}{P(x,y,r)} \quad \frac{P(x,y-1,r-x) \quad \models y \neq 0}{P(x,y,r)}$$
$$\frac{\models y=0 \wedge a=r}{Q(x,y,a,r)} \quad \frac{Q(x,y-1,a+x,r) \quad \models y \neq 0}{Q(x,y,a,r)}$$

Prove this by induction on derivation of $P(x,y,s_1)$

$$\forall x,y,s_1,a,s_2. P(x,y,s_1) \wedge Q(x,y,a,s_2) \Rightarrow s_1 + a = s_2$$

Principle of Induction on Derivation

$$\forall D.\ \psi(D) \quad \text{if and only if}$$
$$\forall D.\ \big(\forall D'. D' \prec D \Rightarrow \psi(D')\big) \Rightarrow \psi(D)$$

where $D' \prec D$ represents that
D' is a strict sub-derivation of D

$$D = \cfrac{\cfrac{\cfrac{D_1}{J_3} \quad D_2}{J_2} \quad D_3 \quad \cfrac{D_4}{J_4}}{J_1}$$

Assume
$\psi(D_1), \psi(D_2),$
$\psi(D_3), \psi(D_4)$
and prove $\psi(D)$

Horn Constraint Solving:

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

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

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

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

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

Induction hypotheses and lemmas

Judgment

$$\emptyset; P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2$$

Premises

$$\frac{\models y = 0 \wedge}{P(x,y,r)} \qquad \frac{, y-1, r-x) \quad \models y \neq 0}{P(x,y,r)}$$

$$\frac{\models y = 0 \wedge a = r}{Q(x,y,a,r)} \qquad \frac{Q(x,y-1,a+x,r) \quad \models y \neq 0}{Q(x,y,a,r)}$$

$$\frac{\models y = 0 \land r = 0}{P(x,y,r)} \qquad \frac{P(x, y-1, r-x) \quad \models y \neq 0}{P(x,y,r)}$$

$$\frac{\models y = 0 \land a = r}{Q(x,y,a,r)} \qquad \frac{Q(x, y-1, a+x, r) \quad \models y \neq 0}{Q(x,y,a,r)}$$

Add an induction hypothesis

Guard to avoid unsound application

$$\gamma = \begin{array}{l} \forall x', y', s_1', a', s_2'.\, D\big(P(x', y', s_1')\big) \prec D\big(P(x, y, s_1)\big) \land \\ \quad P(x', y', s_1{}') \land Q(x', y', a', s_2') \Rightarrow s_1{}' + a' = s_2{}' \end{array}$$

Induct **Unfold**

Case analysis on the last rule used

$$\frac{\gamma; \cdots, y = 0 \land s_1 = 0 \vdash \cdots \qquad \gamma; \cdots, P(x, y-1, s_1 - x), y \neq 0 \vdash \cdots}{\emptyset; \underline{P(x, y, s_1)}, Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \land r = 0}{P(x,y,r)} \qquad \frac{P(x,y-1,r-x) \quad \models y \neq 0}{P(x,y,r)}$$

$$\frac{\models y = 0 \land a = r}{Q(x,y,a,r)} \qquad \frac{Q(x,y-1,a+x,r) \quad \models y \neq 0}{Q(x,y,a,r)}$$

Case analysis on the last rule used

Unfold

$$\dfrac{\dfrac{\gamma; \cdots, \cdots \land y = 0 \land a = s_2 \vdash \cdots \qquad \gamma; \cdots, Q(x,y-1,a+x,s_2), \cdots \land y \neq 0 \vdash \cdots}{\gamma; P(x,y,s_1), \underline{Q(x,y,a,s_2)}, y = 0 \land s_1 = 0 \vdash s_1 + a = s_2}}{\emptyset; P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$\frac{\boxed{\models y = 0 \land r = 0}}{P(x, y, r)} \qquad \frac{P(x, y-1, r-x) \quad \models y \neq 0}{P(x, y, r)}$$

$$\frac{\boxed{\models y = 0 \land a = r}}{Q(x, y, a, r)} \qquad \frac{Q(x, y-1, a+x, r) \quad \models y \neq 0}{Q(x, y, a, r)}$$

$$\frac{\dfrac{\boxed{\gamma; \cdots, \cdots \land y = 0 \land a = s_2 \vdash \cdots}}{\gamma; P(x, y, s_1), Q(x, y, a, s_2), y = 0 \land s_1 = 0 \vdash s_1 + a = s_2}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\dfrac{\models y = 0 \land r = 0}{P(x,y,r)} \qquad \dfrac{P(x, y-1, r-x) \qquad \models y \neq 0}{P(x,y,r)}$$

$$\dfrac{\models y = 0 \land a = r}{Q(x,y,a,r)} \qquad \dfrac{Q(x, y-1, a+x, r) \qquad \models y \neq 0}{Q(x,y,a,r)}$$

Validity checking

Valid

$$\dfrac{\dfrac{\dfrac{\models y = 0 \land s_1 = 0 \land a = s_2 \Rightarrow s_1 + a = s_2}{\gamma; \cdots, y = 0 \land s_1 = 0 \land a = s_2 \vdash s_1 + a = s_2}}{\gamma; P(x,y,s_1), Q(x,y,a,s_2), y = 0 \land s_1 = 0 \vdash s_1 + a = s_2}}{\emptyset; P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$\boxed{\frac{\models y = 0 \wedge r = 0}{P(x, y, r)}} \qquad \frac{P(x, y-1, r-x) \qquad \models y \neq 0}{P(x, y, r)}$$

$$\frac{\models y = 0 \wedge a = r}{Q(x, y, a, r)} \qquad \boxed{\frac{Q(x, y-1, a+x, r) \qquad \models y \neq 0}{Q(x, y, a, r)}}$$

$$\cfrac{\cfrac{\boxed{\gamma; \cdots, Q(x, y-1, a+x, s_2), \cdots \wedge y \neq 0 \vdash \ \cdots}}{\gamma; P(x, y, s_1), Q(x, y, a, s_2), y = 0 \wedge s_1 = 0 \vdash s_1 + a = s_2}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\boxed{\dfrac{\models y = 0 \land r = 0}{P(x, y, r)}} \qquad \dfrac{P(x, y-1, r-x) \qquad \models y \neq 0}{P(x, y, r)}$$

$$\dfrac{\models y = 0 \land a = r}{Q(x, y, a, r)} \qquad \boxed{\dfrac{Q(x, y-1, a+x, r) \qquad \models y \neq 0}{Q(x, y, a, r)}}$$

Valid

$$\dfrac{\dfrac{\dfrac{\models y = 0 \land s_1 = 0 \land y \neq 0 \Rightarrow s_1 + a = s_2}{\gamma; \cdots, Q(x, y-1, a+x, s_2), y = 0 \land s_1 = 0 \land y \neq 0 \vdash s_1 + a = s_2}}{\gamma; P(x, y, s_1), Q(x, y, a, s_2), y = 0 \land s_1 = 0 \vdash s_1 + a = s_2}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\boxed{\dfrac{\models y = 0 \wedge r = 0}{P(x, y, r)}} \qquad \dfrac{P(x, y-1, r-x) \qquad \models y \neq 0}{P(x, y, r)}$$

$$\dfrac{\models y = 0 \wedge a = r}{Q(x, y, a, r)} \qquad \dfrac{Q(x, y-1, a+x, r) \qquad \models y \neq 0}{Q(x, y, a, r)}$$

$$\dfrac{\boxed{\gamma; \cdots, y = 0 \wedge s_1 = 0 \vdash \cdots}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \land r = 0}{P(x, y, r)} \qquad \boxed{\frac{P(x, y-1, r-x) \qquad \models y \neq 0}{P(x, y, r)}}$$

$$\frac{\models y = 0 \land a = r}{Q(x, y, a, r)} \qquad \frac{Q(x, y-1, a+x, r) \qquad \models y \neq 0}{Q(x, y, a, r)}$$

$$\frac{\boxed{\gamma; \cdots, P(x, y-1, s_1 - x), y \neq 0 \vdash \cdots}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \wedge r = 0}{P(x,y,r)} \qquad \frac{P(x, y-1, r-x) \quad \models y \neq 0}{P(x,y,r)}$$

$$\frac{\models y = 0 \wedge a = r}{Q(x,y,a,r)} \qquad \frac{Q(x, y-1, a+x, r) \quad \models y \neq 0}{Q(x,y,a,r)}$$

Unfold

Case analysis on the last rule used

$$\dfrac{\dfrac{\gamma; \cdots, \cdots \wedge y = 0 \wedge a = s_2 \vdash \ \cdots \qquad \gamma; \cdots, Q(x, y-1, a+x, s_2), \cdots \wedge y \neq 0 \vdash \ \cdots}{\gamma; P(x,y,s_1), \underline{Q(x,y,a,s_2)}, P(x, y-1, s_1 - x), y \neq 0 \vdash s_1 + a = s_2}}{\emptyset; P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \wedge r = 0}{P(x,y,r)} \qquad \boxed{\frac{P(x,y-1,r-x) \quad \models y \neq 0}{P(x,y,r)}}$$

$$\boxed{\frac{\models y = 0 \wedge a = r}{Q(x,y,a,r)}} \qquad \frac{Q(x,y-1,a+x,r) \quad \models y \neq 0}{Q(x,y,a,r)}$$

$$\cfrac{\cfrac{\boxed{\gamma; \cdots, \cdots \wedge y = 0 \wedge a = s_2 \vdash \cdots}}{\gamma; P(x,y,s_1), Q(x,y,a,s_2), \color{red}{P(x,y-1,s_1-x)}, y \neq 0 \vdash s_1 + a = s_2}}{\emptyset; P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \land r = 0}{P(x, y, r)} \qquad \boxed{\frac{P(x, y-1, r-x) \quad \models y \neq 0}{P(x, y, r)}}$$

$$\boxed{\frac{\models y = 0 \land a = r}{Q(x, y, a, r)}} \qquad \frac{Q(x, y-1, a+x, r) \quad \models y \neq 0}{Q(x, y, a, r)}$$

Valid

$$\dfrac{\dfrac{\dfrac{\models y \neq 0 \land y = 0 \land a = s_2 \Rightarrow s_1 + a = s_2}{\gamma; \cdots, y \neq 0 \land y = 0 \land a = s_2 \vdash s_1 + a = s_2}}{\gamma; P(x, y, s_1), Q(x, y, a, s_2), {\color{red}P(x, y-1, s_1 - x)}, y \neq 0 \vdash s_1 + a = s_2}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \wedge r = 0}{P(x, y, r)} \qquad \boxed{\frac{P(x, y-1, r-x) \quad \models y \neq 0}{P(x, y, r)}}$$

$$\frac{\models y = 0 \wedge a = r}{Q(x, y, a, r)} \qquad \boxed{\frac{Q(x, y-1, a+x, r) \quad \models y \neq 0}{Q(x, y, a, r)}}$$

$$\cfrac{\cfrac{\boxed{\gamma; \cdots, Q(x, y-1, a+x, s_2), \cdots \wedge y \neq 0 \vdash \cdots}}{\gamma; P(x, y, s_1), Q(x, y, a, s_2), P(x, y-1, s_1 - x), y \neq 0 \vdash s_1 + a = s_2}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \land r = 0}{P(x,y,r)} \qquad \boxed{\frac{P(x, y-1, r-x) \qquad \models y \neq 0}{P(x,y,r)}}$$

$$\frac{\models y = 0 \land a = r}{Q(x,y,a,r)} \qquad \boxed{\frac{Q(x, y-1, a+x, r) \qquad \models y \neq 0}{Q(x,y,a,r)}}$$

$$\boxed{\sigma(\gamma) = \begin{array}{c} D\big(P(x, y-1, s_1 - x)\big) \prec D\big(P(x,y,s_1)\big) \land P(x, y-1, s_1 - x) \land \\ Q(x, y-1, a+x, s_2) \Rightarrow (s_1 - x) + (a + x) = s_2 \end{array}}$$

IndHyp Apply induction hypothesis

$$\frac{\frac{\frac{\gamma; \cdots, y \neq 0 \land (s_1 - x) + (a + x) = s_2 \vdash s_1 + a = s_2}{\gamma; \cdots, \underline{P(x, y-1, s_1 - x), Q(x, y-1, a+x, s_2)}, y \neq 0 \vdash s_1 + a = s_2}}{\gamma; P(x,y,s_1), Q(x,y,a,s_2), P(x, y-1, s_1 - x), y \neq 0 \vdash s_1 + a = s_2}}{\emptyset; P(x,y,s_1), Q(x,y,a,s_2) \vdash s_1 + a = s_2}$$

$$\frac{\models y = 0 \land r = 0}{P(x, y, r)} \qquad \boxed{\frac{P(x, y-1, r-x) \qquad \models y \neq 0}{P(x, y, r)}}$$

$$\frac{\models y = 0 \land a = r}{Q(x, y, a, r)} \qquad \boxed{\frac{Q(x, y-1, a+x, r) \qquad \models y \neq 0}{Q(x, y, a, r)}}$$

Valid

$$\cfrac{\cfrac{\cfrac{\cfrac{\models y \neq 0 \land (s_1 - x) + (a + x) = s_2 \Rightarrow s_1 + a = s_2}{\gamma; \cdots, y \neq 0 \land (s_1 - x) + (a + x) = s_2 \vdash s_1 + a = s_2}}{\gamma; \cdots, P(x, y-1, s_1 - x), Q(x, y-1, a+x, s_2), y \neq 0 \vdash s_1 + a = s_2}}{\gamma; P(x, y, s_1), Q(x, y, a, s_2), P(x, y-1, s_1 - x), y \neq 0 \vdash s_1 + a = s_2}}{\emptyset; P(x, y, s_1), Q(x, y, a, s_2) \vdash s_1 + a = s_2}$$

QED

Properties of Inductive Proof System for CHC Constraint Solving

- Soundness: If the goal is proved, the original CHC constraints have a solution (which may not be expressible in the background theory)
- *Relative* Completeness: If the original constraints have a solution *expressible in the background theory*, the goal is provable

Automating Induction

- Use the following rule application strategy:
  - Repeatedly apply INDHYP until no new premises are added
  - Apply VALID whenever a new premise is added
  - Select some $P(\tilde{t})$ and apply INDUCT and UNFOLD
- Close a proof branch by using:
  - SMT solvers: provide efficient and powerful reasoning about **data structures** (e.g., integers, reals, algebraic data structures) but predicates are abstracted as uninterpreted functions
  - CHC constraint solvers: provide bit costly but powerful reasoning about **inductive predicates**

Prototype Constraint Solver

- Use **Z3** and ***μ*Z PDR** engine respectively as the backend SMT and CHC constraint solvers
- Integrated with a refinement type based verification tool **RCaml** for the OCaml functional language
- Can exploit lemmas which are:
  - User-supplied,
  - Heuristically obtained 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	$0 \leq x_1 \leq x_2 \wedge 0 \leq y_1 \leq y_2 \Rightarrow \texttt{mult}\ x_1\ y_1 \leq \texttt{mult}\ x_2\ y_2$	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 **RCaml**

- 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	$\texttt{try find\_opt}\ p\ l = \texttt{Some (find}\ p\ l\texttt{) with}$ $\texttt{Not\_Found} \rightarrow \texttt{find\_opt}\ p\ l = \texttt{None}$	equiv	H, E	✓	1.065
26	$\texttt{try mem (find ((=)}\ x\texttt{)}\ l\texttt{)}\ l\ \texttt{with Not\_Found} \rightarrow \neg(\texttt{mem}\ x\ l)$	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 of [Unno+ CAV'17]

- Proposed an automated verification method combining CHC constraint solving and inductive theorem proving
  - Enable relational verification across programs in various paradigms with advanced language features and side-effects
  - Support constraints over any background theories (if the backend SMT solver does)
- Ongoing work:
  - Automatic lemma discovery and goal generalization using invariant synthesis techniques (e.g. Craig interpolation)
  - Relational program synthesis

Outline

- ✓ CHC / Fixpoint Constraints for Program Verification
- ✓ CHC Constraint Solving for Relational Verification
  - ✓ Based on [Unno, Torii and Sakamoto, CAV'17]

3. Fixpoint Constraint Solving for Temporal Verification

- **Based on [Nanjo, Unno, Koskinen and Terauchi, LICS'18]**

Temporal Property Verification

Program

Temporal property

$$P \overset{?}{\models} \Phi$$

Check whether P satisfies Φ

This Work

Higher-order functional program

Value-dependent temporal property

$$P \overset{?}{\models} \Phi$$

Check whether P satisfies Φ by using
(1) a dependent refinement type & effect system and
(2) a deductive system for a first-order fixpoint logic

Main Contribution

- Foundation for **compositional & algorithmic** verification of **value-dependent temporal** properties of higher-order programs
  - cf. previous proposals are:
    - fully automated but whole program analysis [Kobayashi+ PLDI’11], [U.+ POPL’13], [Kuwahara+ ESOP’14], [Kuwahara+ CAV’15], [Murase+ POPL’16]
    - compositional but no support of the class of properties [Koskinen+ CSL-LICS’14], [U.+ POPL’18]

This Work

Higher-order functional program

Value-dependent temporal property

$$P \models^{?} \Phi$$

Check whether P satisfies Φ by using
(1) a dependent refinement type & effect system and
(2) a deductive system for a first-order fixpoint logic

Example: Functional Program

```
let rec send_msgs n =
  if n = 0 then ()
  else (event[Send]; send_msgs (n-1))
```

emit Send event

Generated event sequences:

$n < 0 : \mathrm{Send}^{\omega}$ (infinite repetition of Send)

$n = 0 : \epsilon$ (empty sequence)

$n = 1 : \mathrm{Send}$

$n = 2 : \mathrm{Send}, \mathrm{Send}$

$\vdots$

This Work

Higher-order functional program

Value-dependent temporal property

$$P \overset{?}{\models} \Phi$$

Check whether P satisfies Φ by using
(1) a dependent refinement type & effect system and
(2) a deductive system for a first-order fixpoint logic

This Work

predicate for ***finite*** event sequences

predicate for ***infinite*** event sequences

$$P \overset{?}{\models} (\Phi_\mu, \Phi_\nu)$$

Check whether ***finite*** event sequences generated by P satisfy Φ_μ and ***infinite*** event sequences generated by P satisfy Φ_ν

Example: Value-Dependent Temporal Property

```
let rec send_msgs n =
  if n = 0 then
    ()
  else
    (event[Send];
     send_msgs (n-1))
```

$n < 0 : \mathrm{Send}^{\omega}$
$n = 0 : \epsilon$
$n = 1 : \mathrm{Send}$
$n = 2 : \mathrm{Send}, \mathrm{Send}$
$\vdots$

$$\models \quad \begin{array}{l} \Phi^{\mu} \equiv \lambda x \in \Sigma^{*}.\, x = \mathrm{Send}^{n} \\ \Phi^{\nu} \equiv \lambda x \in \Sigma^{\omega}.\, x = \mathrm{Send}^{\omega} \end{array}$$

For terminating executions

n-times repetition of Send

For diverging executions

infinite repetition of Send

Further Examples

- See our LICS'18 paper for further examples that demonstrate the range of applications

Amortized Complexity

```
let rev l =
  let rec aux l acc = match l with
    | [] -> acc | h::t ->
        event[Tick]; aux t (h::acc)
  in aux l []
let is_empty (l1,l2) = l1 = [] && l2 = []
let enqueue e (l1,l2) = event[Enq];(l1,e::l2)
let rec dequeue (l1,l2) = match l1 with
  | [] -> dequeue (rev l2, [])
  | e::l1' -> event[Deq]; (e, (l1', l2))
let rec main (l1,l2) =
  if * then main (enqueue 42 (l1,l2))
  else if is_empty (l1,l2) then ()
  else main (snd (dequeue (l1,l2)))
```

$\texttt{main} : ((l1, l2) : \texttt{int list} \times \texttt{int list}) \to (\texttt{unit} \,\&\, \Phi)$
$\Phi^{\mu} = \lambda x.\#_{\underline{\texttt{Enq}}}(x) + |l2| = \#_{\underline{\texttt{Tick}}}(x) = \#_{\underline{\texttt{Deq}}}(x) - |l1|$
$\Phi^{\nu} = \lambda x.\top$

Higher-Order

```
let rec zoom () =
  event[Zoom]; zoom ()

let rec shrink t f d =
  if f () <= 0 then
    zoom ()
  else
    (event[Shrink];
     let t' = f() - d in
     shrink t' (fun x -> t') d)

let shrinker t d =
  shrink t (fun x -> t) d
```

$\texttt{shrinker} : (t : \{t \mid t \geq 0\}) \to (d : \{d \mid d > 0 \land t \bmod d = 0\}) \to (\texttt{unit} \,\&\, \Phi)$
$\Phi^{\mu} = \lambda x.\bot$
$\Phi^{\nu} = \lambda x.x \in \underline{\texttt{Shrink}}^{t/d} \cdot \underline{\texttt{Zoom}}^{\omega}$

Web Server Fairness

```
let rec listener npool pend =
 if * && pend < npool then
   (event[Accept];
    listener npool (pend + 1))
 else if pend > 0 then
   (event[Handle];
    listener npool (pend - 1))
 else
   (event[Wait];
    listener npool pend)

let server npool =
  listener npool 0
```

$\texttt{server} : (\texttt{npool} : \{\nu \mid \nu \geq 0\}) \to (\texttt{unit} \,\&\, (\lambda x.\bot, \lambda x.\phi))$
$\phi = \begin{pmatrix} x \in (\Sigma^* \cdot (\Sigma \setminus \underline{\texttt{Accept}})^{\texttt{npool}+1})^{\omega} \\ \Rightarrow x \in (\Sigma^* \cdot \underline{\texttt{Wait}})^{\omega} \end{pmatrix}$

This Work

Higher-order functional program

Value-dependent temporal property

$$P \overset{?}{\models} \Phi$$

Check whether P satisfies Φ by using
(1) a dependent refinement type & effect system and
(2) a deductive system for a first-order fixpoint logic

Contributions

1. A dependent refinement type & effect system for **compositional & algorithmic** temporal verification
   - **Compositional** analysis of **dependent temporal effects** represented by predicates of **first-order fixpoint logic $\mathcal{L}$**
   - **Algorithmic** type checking via validity checking for $\mathcal{L}$
2. A deductive system for the validity of $\mathcal{L}$
   - Use **invariants** and **well-founded relations** to **over- and under-approximate fixpoints**
     - Designed by transferring ideas from verification research
   - Can be used with **any background first-order theory**

Contributions

1. A dependent refinement type & effect system for **compositional & algorithmic** temporal verification
   - **Compositional** analysis of **dependent temporal effects** represented by predicates of **first-order fixpoint logic $\mathcal{L}$**
   - **Algorithmic** type checking via validity checking for $\mathcal{L}$
2. A deductive system for the validity of $\mathcal{L}$
   - Use **invariants** and **well-founded relations** to **over- and under-approximate fixpoints**
     - Designed by transferring ideas from verification research
   - Can be used with **any background first-order theory**

First-Order Fixpoint Logic $\mathcal{L}$ (revisited)

- First-order logic extended with least fixpoints (LFPs) and greatest fixpoints (GFPs)

predicate variables

predicate symbols of the background theory

$$\begin{array}{rcl}
(\text{formulas})\ \phi & ::= & \top \mid \bot \mid A(\widetilde{t}) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall x \in \mathcal{S}.\phi \mid \exists x \in \mathcal{S}.\phi \\
& \mid & X(\widetilde{t}) \mid (\mu X(\widetilde{x} : \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \mid (\nu X(\widetilde{x} : \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \\
(\text{terms})\ t & ::= & x \mid f(\widetilde{t}) \\
(\text{sorts})\ \mathcal{S} & ::\in & \{\mathbb{Z}, \Sigma^*, \Sigma^\omega\}
\end{array}$$

LFPs (X occurs only positively in ϕ)

GFPs (X occurs only positively in ϕ)

function symbols of the background theory

the set of finite event sequences

the set of infinite event sequences

We here fix the theory as the one above for ***temporal effect analysis*** , though we could choose any background first-order theory

Temporal Effect Analysis

functional program $e \Rightarrow (\Phi_e^{\mu}, \Phi_e^{\nu})$

***dependent* temporal effect** that describe the temporal behavior of e

Example:

```
let rec send_msgs n =
  if n = 0 then ()
  else (event[Send]; send_msgs (n-1))
```

The use of first-order fixpoint logic allows precise representation (cf. previous work only allowed (ω-)regular expressions [Skalka+'08, Hofmann+'14] or did not specify the effect language [Koskinen+'14])

predicate variable that relates n and the **finite** event sequence x

$$\Phi_e^{\mu} \equiv \lambda x \in \Sigma^*. \left(\mu X_\mu(n, x). \left(\begin{array}{c} n = 0 \ \wedge \ x = \epsilon \ \vee \\ n \neq 0 \ \wedge \ \left(\exists y.\ x = \textbf{Send} \cdot y \ \wedge \ X_\mu(n-1, y)\right) \end{array}\right)\right)(n, x)$$

$$\Phi_e^{\nu} \equiv \lambda x \in \Sigma^{\omega}. \left(\nu X_\nu(n, x). n \neq 0 \wedge \left(\exists y.\ x = \textbf{Send} \cdot y \wedge X_\nu(n-1, y)\right)\right)(n, x)$$

predicate variable that relates n and the **infinite** event sequence x

Dependent Refinement Type & Effect System

$$\boxed{\Gamma} \vdash \boxed{e} : (\boxed{\tau}\ \&\ (\boxed{\Phi^{\mu}, \Phi^{\nu}}))$$

Type Environment

Program

Dependent Refinement Type

Dependent Temporal Effect

Extends existing refinement type systems [Koskinen+'14, Rondon+'08, U.+'09, Terauchi'10, ...]

- Types & effects facilitate **compositional** analysis of dependent temporal effects
- Fixpoint logic deduction ⊩ enables **algorithmic** type checking

Key typing rules:

$$\frac{x \notin fv(\tau_2) \cup fv(\Phi_2) \quad \Gamma \vdash e_1 : (\tau_1\ \&\ \Phi_1) \quad \Gamma, x : \tau_1 \vdash e_2 : (\tau_2\ \&\ \Phi_2)}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : (\tau_2\ \&\ \boxed{\Phi_1 \cdot \Phi_2})}$$

Sequential composition of effects

$$\Phi_1 \cdot \Phi_2 = (\lambda x \in \Sigma^*.\exists x_1, x_2 \in \Sigma^*.x = x_1 \cdot x_2 \wedge \Phi_1^{\mu}(x_1) \wedge \Phi_2^{\mu}(x_2),$$
$$\lambda x \in \Sigma^{\omega}.\Phi_1^{\nu}(x) \vee (\exists y \in \Sigma^*, z \in \Sigma^{\omega}.x = y \cdot z \wedge \Phi_1^{\mu}(y) \wedge \Phi_2^{\nu}(z)))$$

$$\frac{\begin{array}{c} \tau_f' = (\widetilde{x} : \widetilde{\tau}) \to (\tau\ \&\ (\lambda x \in \Sigma^*.X_{\mu}(\widetilde{x}, x), \lambda x \in \Sigma^{\omega}.X_{\nu}(\widetilde{x}, x))) \\ \Gamma, f : \tau_f', \widetilde{x} : \widetilde{\tau} \vdash e : (\tau\ \&\ \Phi) \\ \boxed{q_{\mu} = \mu X_{\mu}(\widetilde{x}, x).\Phi^{\mu}(x) \quad q_{\nu} = \nu X_{\nu}(\widetilde{x}, x).[q_{\mu}/X_{\mu}]\Phi^{\nu}(x)} \\ \tau_f = (\widetilde{x} : \widetilde{\tau}) \to (\tau\ \&\ (\lambda x \in \Sigma^*.q_{\mu}(\widetilde{x}, x), \lambda x \in \Sigma^{\omega}.q_{\nu}(\widetilde{x}, x))) \end{array}}{\Gamma \vdash \mathtt{rec}(f, \widetilde{x}, e) : (\tau_f\ \&\ \Phi_{val})}$$

Fixpoints describing a dependent temporal effect of a recursive function

$$\frac{\Gamma \vdash e : \sigma_1 \quad \boxed{\Gamma \vdash \sigma_1 <: \sigma_2}}{\Gamma \vdash e : \sigma_2}$$

Subtyping

$$\frac{\Gamma \vdash \tau_1 <: \tau_2 \quad \boxed{\begin{array}{l} \Vdash \lfloor \Gamma \vdash \forall x \in \Sigma^*.\Phi_1^{\mu}(x) \Rightarrow \Phi_2^{\mu}(x) \rfloor \\ \Vdash \lfloor \Gamma \vdash \forall x \in \Sigma^{\omega}.\Phi_1^{\nu}(x) \Rightarrow \Phi_2^{\nu}(x) \rfloor \end{array}}}{\Gamma \vdash (\tau_1\ \&\ \Phi_1) <: (\tau_2\ \&\ \Phi_2)}$$

Check sub-effect relation via fixpoint logic deduction

Theorem 1 (Soundness): $\Gamma \vdash e : \big(\tau\ \&\ (\Phi^{\mu}, \Phi^{\nu})\big)$ implies $e \in [\![\Gamma \vdash \tau\ \&\ (\Phi^{\mu}, \Phi^{\nu})]\!]$
(e behaves as specified by $\big(\tau \& (\Phi^{\mu}, \Phi^{\nu})\big)$ under a valuation conforming to Γ)

Contributions

1. A dependent refinement type & effect system for **compositional & algorithmic** temporal verification
   - **Compositional** analysis of **dependent temporal effects** represented by predicates of **first-order fixpoint logic** $\mathcal{L}$
   - **Algorithmic** type checking via validity checking for $\mathcal{L}$
2. A deductive system for the validity of $\mathcal{L}$
   - Use **invariants** and **well-founded relations** to **over- and under-approximate fixpoints**
     - Designed by transferring ideas from verification research
   - Can be used with **any background first-order theory**

First-Order Fixpoint Logic $\mathcal{L}$ (revisited)

- First-order logic extended with least fixpoints (LFPs) and greatest fixpoints (GFPs)

predicate variables

predicate symbols of the background theory

sorts (e.g. $\mathbb{Z}$) of the background theory

$$
\begin{array}{rcl}
\text{(formulas) } \phi & ::= & \top \mid \bot \mid A(\widetilde{t}) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall x \in \mathcal{S}.\phi \mid \exists x \in \mathcal{S}.\phi \\
& \mid & X(\widetilde{t}) \mid (\mu X(\widetilde{x}\colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \mid (\nu X(\widetilde{x}\colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \\
\text{(terms) } t & ::= & x \mid f(\widetilde{t})
\end{array}
$$

function symbols of the background theory

LFPs
(X occurs only positively in ϕ)

GFPs
(X occurs only positively in ϕ)

Deductive System $\Vdash \phi$ for the Validity of $\mathcal{L}$

1. Over- and under-approximate fixpoint subformulas of ϕ by non-fixpoint formulas
   - For soundness, subformulas that occur positively and negatively are respectively under- and over-approximated
2. Resulting non-fixpoint formulas are discharged by a solver for the background first-order theory

- Techniques for obtaining approximations:

	Over-Approximation	Under-Approximation
LFP	Invariant (induction)	Well-founded relation
GFP	Well-founded relation	Invariant (co-induction)

Analogous to techniques in safety and liveness property verification

Example: Fixpoint Deduction via Over-Approx. of LFP

Check that p is a pre-fixpoint of F (or, equivalently, perform induction by unfolding LFP and applying I.H. to the recursive occurrences of X)

Deduction in background first-order theory

$$\dfrac{\boxed{\models F(p)(x) \Rightarrow p(x)} \qquad \dfrac{\boxed{\models p(x) \Rightarrow x = \mathrm{Send}^n}}{\Vdash \boxed{p(x)} \Rightarrow x = \mathrm{Send}^n}}{\Vdash \Phi_e^{\mu}(x) \Rightarrow x = \mathrm{Send}^n}$$

Over-approx. of LFP by pre-fixpoint

$$\lambda x \in \Sigma^*.\, x = \mathrm{Send}^n$$

$$\lambda x \in \Sigma^*.\left(\mu X_\mu(n,x).\, F\left(X_\mu\right)(n,x)\right)(n,x)$$

where $F(X)(n,x) = \left(\begin{array}{c} n = 0 \wedge x = \epsilon \vee \\ n \neq 0 \wedge \left(\exists y.\ x = \mathrm{Send} \cdot y \wedge X(n-1,y)\right) \end{array}\right)$

Example: Fixpoint Deduction via Over-Approx. of GFP

Check that the given well-founded relation p_2 witnesses that the given predicate $\boldsymbol{p_1}$ **and** $\boldsymbol{\Phi_e^\nu}$ **have no intersection** (see the paper for details)

Deduction in background first-order theory

$$\dfrac{\dfrac{\dfrac{\vDash \begin{array}{l} p_1(n,x) \land n \neq 0 \land x = \mathrm{Send} \cdot x' \Rightarrow \\ (p_1(n-1,x') \land p_2(n,x,n-1,x')) \end{array}}{\vdots}}{X_\nu(n,x); p_1; p_2; \top \ \overline{\uparrow}\ n \neq 0 \land \exists y.\, x = \mathrm{Send} \cdot y \land X_\nu(n-1,y)} \qquad \dfrac{\vDash \neg p_1(x) \Rightarrow x = \mathrm{Send}^\omega}{\Vdash \neg p_1(x) \Rightarrow x = \mathrm{Send}^\omega}}{\Vdash \Phi_e^\nu(x) \Rightarrow x = \mathrm{Send}^\omega}$$

Over-approx. of GFP by negation of p_1

$$\lambda x \in \Sigma^\omega.\, n \geq 0 \lor x \neq \mathrm{Send}^\omega$$

$$\lambda(n_1, x_1, n_2, x_2).\, n_1 > n_2 \geq 0$$

$$\lambda x \in \Sigma^\omega. \left(\nu X_\nu(n,x).\, n \neq 0 \land \begin{pmatrix} \exists y.\ x = \mathrm{Send} \cdot y \land \\ X_\nu(n-1,y) \end{pmatrix}\right)(n,x)$$

Deductive System $\Vdash \phi$ for $\mathcal{L}$

Background first-order theory solver

$$\frac{\models \psi}{\Vdash \psi}\ \textsc{Fp-Valid}$$

$$\frac{\models [(\lambda\widetilde{x}.\psi')/X]\psi \Rightarrow \psi' \quad \Vdash C^-[[\widetilde{t}/\widetilde{x}]\psi']}{\Vdash C^-[(\mu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Lfp}^-$$

Over-approximation of LFP (induction)

$$\frac{\models \psi' \Rightarrow [(\lambda\widetilde{x}.\psi')/X]\psi \quad \Vdash C^+[[\widetilde{t}/\widetilde{x}]\psi']}{\Vdash C^+[(\nu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Gfp}^+$$

Under-approximation of GFP (co-induction)

$$\frac{X(\widetilde{x}); p_1; p_2; \top \downarrow nnf(\psi) \quad \Vdash C^+[p_1(\widetilde{t})] \quad \models WF(p_2)}{\Vdash C^+[(\mu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Lfp}^+$$

$$\frac{X(\widetilde{x}); p_1; p_2; \top \,\overline{\uparrow}\, nnf(\psi) \quad \Vdash C^-[\neg p_1(\widetilde{t})] \quad \models WF(p_2)}{\Vdash C^-[(\nu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Gfp}^-$$

Approximation of fixpoints using well-founded relation (the next slide)

ψ represents a fixpoint-free formula.

$nnf(\psi)$ is negation normal form of ψ.

C^+ (resp. C^-) is positive (resp. negative) context.

Theorem 2 (Soundness of $\Vdash$): $\Vdash \phi$ implies $\models \phi$

Fixpoint Approximation Rules based on Well-Founded Relation

$$\dfrac{\models p_1(\widetilde{x}) \wedge \psi' \Rightarrow \psi}{X(\widetilde{x}); p_1; p_2; \psi' \downarrow \psi}\ \textsc{Apx}^{\mu}\text{-}\textsc{Base}$$

$$\dfrac{\models p_1(\widetilde{x}) \wedge \psi \Rightarrow p_1(\widetilde{t}) \wedge p_2(\widetilde{x}, \widetilde{t})}{X(\widetilde{x}); p_1; p_2; \psi \downarrow X(\widetilde{t})}\ \textsc{Apx}^{\mu}\text{-}\textsc{Rec}$$

$$\dfrac{X(\widetilde{x}); p_1; p_2; \psi \downarrow \psi_1 \quad X(\widetilde{x}); p_1; p_2; \psi \downarrow \psi_2}{X(\widetilde{x}); p_1; p_2; \psi \downarrow \psi_1 \wedge \psi_2}\ \textsc{Apx}^{\mu}\text{-}\wedge$$

$$\dfrac{\begin{array}{c}\models (p_1(\widetilde{x}) \wedge \psi) \Rightarrow (\psi_1' \vee \psi_2') \quad \mathit{fv}(\psi_i') \subseteq \{\widetilde{x}\} \quad X \notin \mathit{fpv}(\psi_i') \\ X(\widetilde{x}); p_1; p_2; \psi \wedge \psi_i' \downarrow \psi_i \quad (i = 1, 2)\end{array}}{X(\widetilde{x}); p_1; p_2; \psi \downarrow \psi_1 \vee \psi_2}\ \textsc{Apx}^{\mu}\text{-}\vee$$

$$\dfrac{\begin{array}{c}X(\widetilde{x}); p_1; p_2; \psi' \downarrow [x'/x]\psi \\ x' \notin \mathit{fv}(\psi') \cup \mathit{fv}(\psi) \cup \{\widetilde{x}\} \cup \mathit{fv}(p_1) \cup \mathit{fv}(p_2)\end{array}}{X(\widetilde{x}); p_1; p_2; \psi' \downarrow \forall x.\psi}\ \textsc{Apx}^{\mu}\text{-}\forall$$

$$\dfrac{\begin{array}{c}\models (p_1(\widetilde{x}) \wedge \psi') \Rightarrow \exists x'.\psi'' \quad \mathit{fv}(\psi'') \subseteq \{\widetilde{x}, x'\} \quad X \notin \mathit{fpv}(\psi'') \\ X(\widetilde{x}); p_1; p_2; \psi' \wedge \psi'' \downarrow [x'/x]\psi \\ x' \notin \mathit{fv}(\psi') \cup \mathit{fv}(\psi) \cup \{\widetilde{x}\} \cup \mathit{fv}(p_1) \cup \mathit{fv}(p_2)\end{array}}{X(\widetilde{x}); p_1; p_2; \psi' \downarrow \exists x.\psi}\ \textsc{Apx}^{\mu}\text{-}\exists$$

Under-approximation of LFP

Dual

$$\dfrac{\models p_1(\widetilde{x}) \wedge \psi' \Rightarrow \neg\psi}{X(\widetilde{x}); p_1; p_2; \psi' \overline{\uparrow} \psi}\ \textsc{Apx}^{\nu}\text{-}\textsc{Base}$$

$$\dfrac{\models p_1(\widetilde{x}) \wedge \psi \Rightarrow p_1(\widetilde{t}) \wedge p_2(\widetilde{x}, \widetilde{t})}{X(\widetilde{x}); p_1; p_2; \psi \overline{\uparrow} X(\widetilde{t})}\ \textsc{Apx}^{\nu}\text{-}\textsc{Rec}$$

$$\dfrac{\begin{array}{c}\models (p_1(\widetilde{x}) \wedge \psi) \Rightarrow (\psi_1' \vee \psi_2') \quad \mathit{fv}(\psi_i') \subseteq \{\widetilde{x}\} \quad X \notin \mathit{fpv}(\psi_i') \\ X(\widetilde{x}); p_1; p_2; \psi \wedge \psi_i' \overline{\uparrow} \psi_i \quad (i = 1, 2)\end{array}}{X(\widetilde{x}); p_1; p_2; \psi \overline{\uparrow} \psi_1 \wedge \psi_2}\ \textsc{Apx}^{\nu}\text{-}\wedge$$

$$\dfrac{X(\widetilde{x}); p_1; p_2; \psi \overline{\uparrow} \psi_1 \quad X(\widetilde{x}); p_1; p_2; \psi \overline{\uparrow} \psi_2}{X(\widetilde{x}); p_1; p_2; \psi \overline{\uparrow} \psi_1 \vee \psi_2}\ \textsc{Apx}^{\nu}\text{-}\vee$$

$$\dfrac{\begin{array}{c}\models (p_1(\widetilde{x}) \wedge \psi') \Rightarrow \exists x'.\psi'' \quad \mathit{fv}(\psi'') \subseteq \{\widetilde{x}, x'\} \quad X \notin \mathit{fpv}(\psi'') \\ X(\widetilde{x}); p_1; p_2; \psi' \wedge \psi'' \overline{\uparrow} [x'/x]\psi \\ x' \notin \mathit{fv}(\psi') \cup \mathit{fv}(\psi) \cup \{\widetilde{x}\} \cup \mathit{fv}(p_1) \cup \mathit{fv}(p_2)\end{array}}{X(\widetilde{x}); p_1; p_2; \psi' \overline{\uparrow} \forall x.\psi}\ \textsc{Apx}^{\nu}\text{-}\forall$$

$$\dfrac{\begin{array}{c}X(\widetilde{x}); p_1; p_2; \psi' \overline{\uparrow} [x'/x]\psi \\ x' \notin \mathit{fv}(\psi') \cup \mathit{fv}(\psi) \cup \{\widetilde{x}\} \cup \mathit{fv}(p_1) \cup \mathit{fv}(p_2)\end{array}}{X(\widetilde{x}); p_1; p_2; \psi' \overline{\uparrow} \exists x.\psi}\ \textsc{Apx}^{\nu}\text{-}\exists$$

Over-approximation of GFP

Lemma: Suppose that ψ is in negation normal form, X is not free in ψ' and p_2 is a well-founded relation. We have:

- $X(\tilde{x}); p_1; p_2; \psi' \downarrow \psi$ implies $p_1(\tilde{x}) \Rightarrow (\mu X(\tilde{x}). \neg\psi' \vee \psi)(\tilde{x})$
- $X(\tilde{x}); p_1; p_2; \psi' \overline{\uparrow} \psi$ implies $(\nu X(\tilde{x}).\ \psi' \wedge \psi)(\tilde{x}) \Rightarrow \neg p_1(\tilde{x})$

Summary of [Nanjo+ LICS'18]

- Foundation for **compositional & algorithmic** verification of **value-dependent temporal** properties of higher-order programs

1. Dependent refinement type & effect system
   - **Compositional** analysis of **dependent temporal effects** represented by predicates of **first-order fixpoint logic $\mathcal{L}$**
   - **Algorithmic** type checking via validity checking for $\mathcal{L}$
2. Deductive system for the validity of $\mathcal{L}$

	Over-Approximation	Under-Approximation
LFP	Invariant (induction)	Well-founded relation
GFP	Well-founded relation	Invariant (co-induction)

   - Can be used with **any background first-order theory**

- Ongoing Work
  - Automation and implementation
  - Extensions to branching- and relational-properties verification

Outline

- ✓ CHC / Fixpoint Constraints for Program Verification
- ✓ CHC Constraint Solving for Relational Verification
  - ✓ Based on [Unno, Torii and Sakamoto, CAV'17]
- ✓ Fixpoint Constraint Solving for Temporal Verification
  - ✓ Based on [Nanjo, Unno, Koskinen and Terauchi, LICS'18]

Conclusion

- Fixpoint constraint solving generalizes CHC solving
  - Significantly widen the range of applications to verification of liveness and existential properties
- Inductive theorem proving techniques facilitate (relational) CHC solving, and vice versa
- Safety and liveness verification techniques (invariants and well-founded relations) facilitate fixpoint constraint solving
- Future Work
  - Fixpoint constraint solving based on inductive and co-inductive theorem proving for verification of temporal relational properties (e.g. trace equivalence) and hyperproperties [Clarkson+ ’09]