An Overview of the HFL Model Checking Project (at UTokyo)

Naoki Kobayashi

The University of Tokyo

In collaboration with: Kazuyuki Asada, Florian Bruze, Adrien Champion, Grigory Fedyukobich, Aarti Guputa, Atushi Igarashi, Etienne Lozes, Takeshi Nishikawa, Ryosuke Sato, Takeshi Tsukada, Hiroshi Unno, and (ex-)students at UTokyo

This Talk

- **An Overview of Our Project on Automated Program Verification Based on Higher-Order Fixpoint Logic (HFL)**
  - **HFL** [Viswanathan&Viswanathan 04] **as a higher-order extension of the modal μ-calculus**
  - **HFL(Z) (HFL with integers) as an extension of Constrained Horn Clauses (CHC, a.k.a. CLP) with higher-order predicates and fixpoint alternation**
  - **Natural reduction from higher-order program verification to HFL(Z) model checking** [K+ ESOP18][Watanabe+ PEPM19]
    - More uniform approach than our previous approach based on HORS model checking [K, POPL09][K+ POPL10][K+ PLDI11][K, JACM13]...
  - **Automated techniques for HFL(Z) model checking based on CHC solving** [K+ SAS19][Hosoi+ APLAS19][K+ TACAS19][Katsura+ APLAS20] ...
  - **Machine learning techniques for CHC solving** [Champion+ TACAS18] ...

Outline

- ♦ **Introduction to HFL and HFL(Z)**
  - – **What is higher-order fixpoint logic?**
  - – **HFL model checking as a higher-order extension of finite state model checking**
  - – **HFL(Z) as an extension of Constrained Horn Clauses (CHC)**
- ♦ **Reductions from program verification to HFL(Z) model checking**
- ♦ **Solving HFL(Z) model checking using types, CHC solving, and higher-order model checking**
- ♦ **Machine learning techniques for CHC solving**

Outline

- ♦ Introduction to HFL and HFL(Z)
  - What is higher-order fixpoint logic?
  - HFL model checking as a higher-order extension of finite state model checking
  - HFL(Z) as an extension of Constrained Horn Clauses (CHC)
- ♦ Reductions from program verification to HFL(Z) model checking
- ♦ Solving HFL(Z) model checking using types, CHC solving, and higher-order model checking
- ♦ Machine learning techniques for CHC solving

Higher-Order Modal Fixpoint Logic (HFL)

[Viswanathan&Viswanathan 04]

♦ Higher-order extension of the modal μ-calculus

φ ::=	true	
	$\varphi_1 \wedge \varphi_2$	
	$\varphi_1 \vee \varphi_2$	
	$[a]\varphi$	φ *must* hold after a
	$\langle a\rangle\varphi$	φ *may* hold after a
	X	variable
	$\mu X.\varphi$	*least* fixpoint (the least X such that $X=\varphi$)
	$\nu X.\varphi$	*greatest* fixpoint (the greatest X such that $X=\varphi$)

e.g. $\mu X. \langle b\rangle \mathrm{true} \vee \langle a\rangle X$

“b” may occur after a finite number of “a” transitions

(i.e., there exists a transition sequence in which “b” occurs after a finite number of “a” transitions)

Higher-Order Modal Fixpoint Logic (HFL)

[Viswanathan&Viswanathan 04]

- **Higher-order extension of the modal μ-calculus**

φ ::= true	
$\varphi_1 \wedge \varphi_2$	
$\varphi_1 \vee \varphi_2$	
$[a]\varphi$	φ must hold after a
$\langle a\rangle\varphi$	φ may hold after a
X	predicate variable
$\mu X^{\kappa}.\varphi$	*least* fixpoint (the least X such that X=φ)
$\nu X^{\kappa}.\varphi$	*greatest* fixpoint (the greatest X such that X=φ)
$\lambda X^{\kappa}.\ \varphi$	(higher-order) predicate
$\varphi_1\ \varphi_2$	application
κ ::= $\bullet$	the type of propositions
$\kappa_1 \rightarrow \kappa_2$	

Selected Typing Rules for HFL

$$\Gamma \vdash \mathbf{true} : \bullet$$

$$\frac{\Gamma \vdash \varphi : \bullet}{\Gamma \vdash [a]\varphi : \bullet}$$

$$\frac{\Gamma \vdash \varphi : \bullet \qquad \Gamma \vdash \psi : \bullet}{\Gamma \vdash \varphi \wedge \psi : \bullet}$$

$$\frac{\Gamma, X : \kappa_1 \vdash \varphi : \kappa_2}{\Gamma \vdash \lambda X.\varphi : \kappa_1 \to \kappa_2}$$

$$\Gamma, X : \kappa \vdash X : \kappa$$

$$\frac{\Gamma \vdash \varphi : \kappa_1 \to \kappa_2 \qquad \Gamma \vdash \psi : \kappa_1}{\Gamma \vdash \varphi\ \psi : \kappa_2}$$

$$\frac{\Gamma, X : \kappa \vdash \varphi : \kappa}{\Gamma \vdash \mu X.\varphi : \kappa}$$

Example

$$\left(\nu F^{\bullet\to\bullet}.\lambda X.\ X \wedge [a](F\ ([b]X)\right)\ \langle c\rangle$$

$$= \left(\lambda X.\ X \wedge [a](\nu F\ldots)\ ([b]X)\right)\ \langle c\rangle$$

$$= \langle c\rangle \wedge\ [a]\big((\nu F^{\bullet\to\bullet}.\lambda X.\ X \wedge [a](F\ ([b]X))\ ([b]\langle c\rangle)\big)$$

$$= \langle c\rangle \wedge\ [a]\big((\lambda X.\ X \wedge [a](\nu F\ldots)\ ([b]X))\ ([b]\langle c\rangle)\big)$$

$$= \langle c\rangle \wedge\ [a]\big(\ [b]\langle c\rangle \wedge [a](\nu F\ldots)\ ([b][b]\langle c\rangle)\big)$$

$$= \langle c\rangle \wedge [a][b]\langle c\rangle \wedge [a]^2[b]^2\ \langle c\rangle \wedge \ldots$$

After any transitions of the form $a^n b^n$, $\langle c\rangle$ holds

HFL Model Checking

[Viswanathan&Viswanathan 2004]

Given

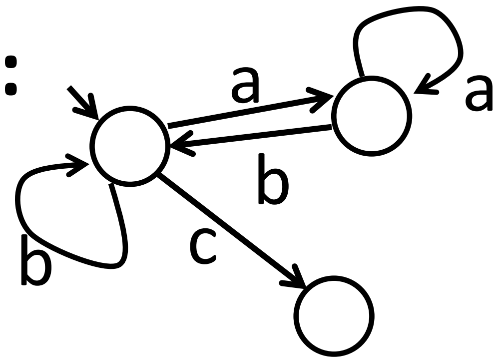
$\mathcal{L}$: (finite-state) labeled transition system

φ: HFL formula,

does $\mathcal{L}$ satisfy φ?

e.g. $\mathcal{L} \models \varphi$ for:

$\mathcal{L}$:



φ: $(\nu F^{\bullet\to\bullet}.\lambda X.\ X \wedge [a](F\ ([b]X))\ \langle c\rangle$

An alternative notation:

F <c> where

$F\ X =_\nu X \wedge [a](F\ ([b]X))$

HFL Model Checking

[Viswanathan&Viswanathan 2004]

Given
L: (finite-state) labeled transition system
φ: HFL formula,
does L satisfy φ?

- k-EXPTIME complete for order-k HFL [Axelsson+ 07]
  but a practical algorithm exists [Hosoi+ 19]
  (order($\bullet$) = 0, order($\kappa_1 \rightarrow \ldots \rightarrow \kappa_n \rightarrow \bullet$) = 1+max(order($\kappa_1$), ..., order($\kappa_n$)))
- Polynomial time translation exists
  between HFL model checking and HORS model checking [K+ POPL17]

The other kind of higher-order model checking [Ong 06]

HFL(Z): An extension of HFL with integers

$$\begin{array}{l}\varphi ::= \text{true} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [a]\varphi \mid \langle a\rangle\varphi \\ \qquad \mid X \mid \mu X^{\kappa}.\varphi \mid \nu X^{\kappa}.\varphi \mid \lambda X^{\tau}.\,\varphi \mid \varphi_1\,\varphi_2 \end{array} \Bigg\}\ \text{pure HFL}$$

$$\qquad \mid \varphi\, e \mid e_1 = e_2$$

$$e ::= n \mid X \mid e_1 + e_2$$

$$\kappa ::= o \mid \tau \to \kappa$$

$$\tau ::= \kappa \mid \text{int}$$

Example:

$(\mu E.\lambda x.\ x=0 \vee E(x-2))n$

$\equiv\ (\lambda x.\ x=0 \vee (\mu E.\lambda x.\ \ldots)(x-2))n$

$\equiv\ n=0 \vee (\mu E.\lambda x.\ x=0 \vee E(x-2))(n-2)$

$\equiv\ n=0 \vee n-2=0 \vee \ldots$

$\equiv$ "n is an even non-negative integer"

$(\nu X.\lambda x.\ P(x) \wedge X(x+1))0$

$\equiv\ (\lambda x.\ P(x) \wedge (\nu X.\lambda x.\ \ldots)(x+1))0$

$\equiv\ P(0) \wedge (\nu X.\lambda x.\ P(x) \wedge X(x+1))\ 1$

$\equiv\ P(0) \wedge P(1) \wedge \ldots$

$\equiv\ \forall x \geq 0.\ P(x)$

HFL(Z) Model/Validity Checking

HFL(Z) Model Checking:
Given
$\mathcal{L}$: (finite-state) labeled transition system
φ: a closed HFL(Z) formula,
does $\mathcal{L}$ satisfy φ?

HFL(Z) Validity Checking:
Given
φ: a closed HFL(Z) formula without modalities ([a], <a>),
is φ valid?
(or, does the trivial model satisfy φ ?)

Outline

- ♦ Introduction to HFL and HFL(Z)
  - What is higher-order fixpoint logic?
  - HFL model checking as a higher-order extension of finite state model checking
  - HFL(Z) as an extension of Constrained Horn Clauses (CHC)
- ♦ Reductions from program verification to HFL(Z) model checking
- ♦ Solving HFL(Z) model checking using types, CHC solving, and higher-order model checking
- ♦ Machine learning techniques for CHC solving

CHC satisfiability as (ν-only) HFL(Z) validity checking

Is
$P(x) \Leftarrow \varphi(x)$
$\text{false} \Leftarrow P(e)$
satisfiable?

Is
$\text{false} \Leftarrow (\mu P.\lambda x.\varphi(x))(e)$
valid?

the least P such that $P(x) \Leftarrow \varphi(x)$

Is
$(\nu\overline{P}.\lambda x.\overline{\varphi}(x))(e)$
valid?
($\overline{\varphi}(x)$: de Morgan dual of $\varphi(x)$)

Example

$\mathrm{Fact}(n, r) \Leftarrow n=0, r=1$
$\mathrm{Fact}(n, r) \Leftarrow n\neq 0, \mathrm{Fact}(n-1, s), r=s\times n$
$r\geq n \Leftarrow \mathrm{Fact}(n, r)$

$$r\geq n \Leftarrow$$
$$(\,\mu\mathrm{Fact}.\lambda(n,r).\ (n=0\wedge r=1)$$
$$\vee\ \exists s.\ (n\neq 0 \wedge r=s\times n \wedge \mathrm{Fact}(n-1,s)))(n, r)$$

$$r\geq n\vee(\,\nu\overline{\mathrm{Fact}}.\lambda(n,r).\ (n\neq 0\vee r\neq 1)$$
$$\wedge\ \forall s.\ (n=0 \vee r \neq s\times n \vee \overline{\mathrm{Fact}}(n-1,s)))(n, r)$$

cf.

```
let rec fact(n) =
  if n=0 then r=1 else fact(n-1)×n
let main n = assert(fact(n)≥n)
```

CHC satisfiability as (ν-only) HFL(Z) validity checking

Is
$P(x) \Leftarrow \varphi(x)$
$\text{false} \Leftarrow P(e)$
satisfiable?

Is
$\text{false} \Leftarrow (\mu P.\lambda x.\varphi(x))(e)$
valid?

the least P such that $P(x) \Leftarrow \varphi(x)$?

Is
$(\nu\overline{P}.\lambda x.\overline{\varphi}(x))(e)$
valid?
($\overline{\varphi}(x)$: de Morgan dual of $\varphi(x)$)

Example

$\text{Fact}(n, r) \Leftarrow n=0, r=1$
$\text{Fact}(n, r) \Leftarrow n\neq 0, \text{Fact}(n-1, s), r=s\times n$
$r\geq n \Leftarrow \text{Fact}(n, r)$

$$r\geq n \Leftarrow (\mu\text{Fact}.\lambda(n,r).\ (n=0\wedge r=1) \vee \exists s.\ (n\neq 0 \wedge r=s\times n \wedge \text{Fact}(n-1,s)))(n, r)$$

$$r\geq n\vee(\nu\overline{\text{Fact}}.\lambda(n,r).\ (n\neq 0\vee r\neq 1) \wedge \forall s.\ (n=0 \vee r \neq s\times n \vee \overline{\text{Fact}}(n-1,s)))(n, r)$$

HFL(Z) = CHC
+ higher-order predicates } HoCHC [Burn+18]
+ fixpoint alternation

Outline

- ♦ Introduction to HFL and HFL(Z)
  - What is higher-order fixpoint logic?
  - HFL model checking as a higher-order extension of finite state model checking
  - HFL(Z) as an extension of Constrained Horn Clauses (CHC)
- ♦ Reductions from program verification to HFL(Z) model checking
- ♦ Solving HFL(Z) model checking using types, CHC solving, and higher-order model checking
- ♦ Machine learning techniques for CHC solving

Higher-Order Program Verification vs HFL Model Checking

	Models	Spec
HO program verification	HO programs	safety, termination, ...
Finite state model checking	finite state systems	modal μ-calculus formula
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	HFL formula

??

??

e.g.

F <c> where

$F\ X =_{\nu} X \wedge [a](F\ ([b]X))$

"c is enabled after any transitions of the form $a^n b^n$"

Higher-Order Program Verification vs HFL Model Checking

	Models	Spec
HO program verification	HO programs	safety, termination, ...
Finite state model checking	finite state systems	modal μ-calculus formula
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	HFL formula

"The program's behavior is correct"

From Program Verification to HFL Model Checking: Example

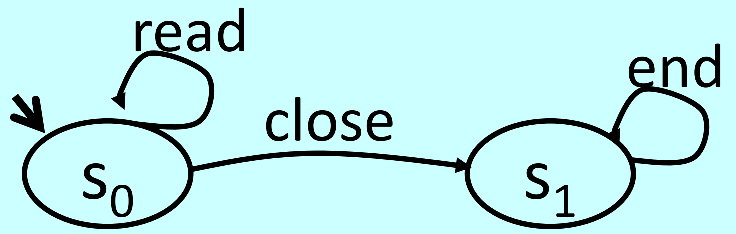
```
let y = open "foo"
in
  read(y); close(y)
```

HFL formula that says "the behavior of the program is correct"

<read><close><end>true

Is the file "foo" accessed according to read* close?

LTS:



From Program Verification to HFL Model Checking: Example

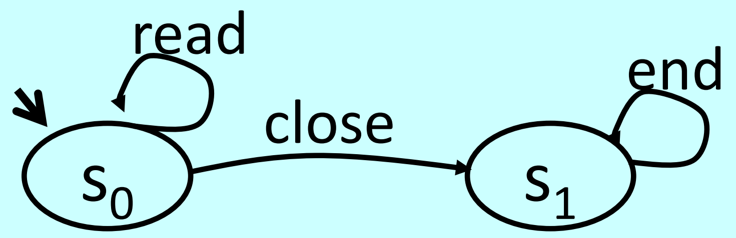
```
let y = open "foo"
in
  read(y); close(y)
```

HFL formula that says "the behavior of the program is correct"

<read><close><end>true

Is the file "foo" accessed according to read* close?

Does LTS:



satisfy the formula S?

From Program Verification to HFL Model Checking: Example

```
let y = open "foo"
in
  if * then
    (read(y); close(y))
  else close(y)
```

HFL formula that says "the behavior of the program is correct"

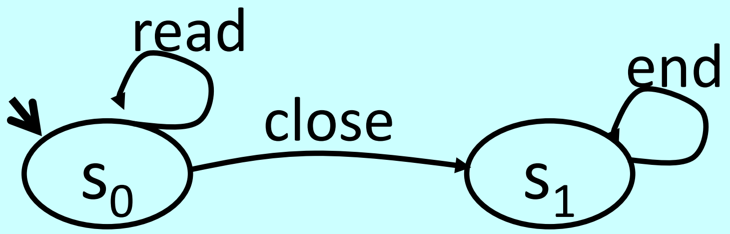
<read><close><end>true

∧

<close><end>true

Is the file "foo" accessed according to read* close?

Does LTS:



satisfy the formula S?

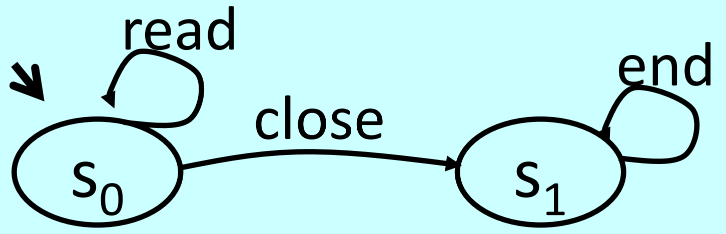
From Program Verification to HFL Model Checking: Example

```
let f x =
 if * then close(x)
 else (read(x); f x)
in
let y = open “foo”
in
    f (y)
```

HFL formula that says “the behavior of the program is correct”

Is the file “foo” accessed according to read* close?

Does LTS:



satisfy the formula S?

From Program Verification to HFL Model Checking: Example

```
let f x k =
 if * then close x k
 else read x (f x k)
in
let y = open "foo"
in
    f y ()
```

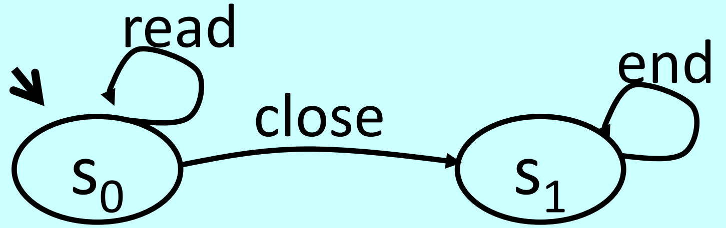
HFL formula that says "the behavior of the program is correct"

$$F\ x\ k =_{\nu} \langle close \rangle k \wedge (\langle read \rangle (F\ x\ k))$$

$$S =_{\nu} F\ true\ (\langle end \rangle true)$$

Is the file "foo" accessed according to read* close?

Does LTS:



satisfy the formula S?

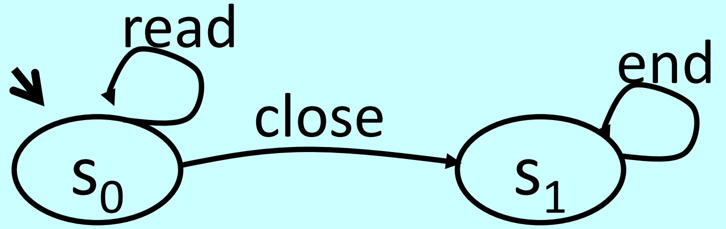
From Program Verification to HFL(Z) Model Checking

```
let f n x k =
 if n≤0 then close x k
 else
  read x (f (n-1) x k)
in
let y = open "foo"
in  f m y ()
```

$$F\ n\ x\ k =_{\mu}$$
$$(n\le 0 \Rightarrow \langle close\rangle k)$$
$$\wedge\ (\neg n\le 0 \Rightarrow$$
$$\langle read\rangle(F\ (n-1)\ x\ k))$$
$$S =_{\mu} F\ m\ true\ (\langle end\rangle true)$$

Is the file "foo" accessed according to read* close?

Does LTS:



satisfy the formula S?

From Program Verification to HFL(Z) Model Checking

let f n x k =

if

e

ir

le

ir

F n x k =$_\nu$

This approach provides a sound and complete logical characterization of:

- reachability problem
- termination problem
- linear/branching-time temporal properties

for higher-order functional programs
[K+ ESOP 2018] [Watanabe+, PEPM 2019]

accessed according to read* close?

satisfy the formula S?

From Termination Verification to HFL(Z) Model Checking

```
let f x y =
 if x≤ y then ()
 else
  if * then f (x-1) y
  else f x (y+1)
in f m n
```

(Must-)termination:

$\forall m, n.\ F\ m\ n$ where:

$F\ x\ y =_{\mu}$
$(x \le y \Rightarrow \text{true}) \land$
$(x > y \Rightarrow F\ (x-1)\ y \land F\ x\ (y+1))$

May-termination:

$\exists m, n.\ F\ m\ n$ where:

$F\ x\ y =_{\mu}$
$(x \le y \land \text{true}) \lor$
$(x > y \land (F\ (x-1)\ y \lor F\ x\ (y+1)))$

May-not-termination:

$\exists m, n.\ F\ m\ n$ where:

$F\ x\ y =_{\nu}$
$(x \le y \land \text{false}) \lor$
$(x > y \land (F\ (x-1)\ y \lor F\ x\ (y+1)))$

Outline

- ♦ Introduction to HFL and HFL(Z)
- ♦ Reductions from program verification to HFL(Z) model checking
- ♦ HFL(Z) validity checking using types, CHC solving, and higher-order model checking
  - Overview
  - From HFL(Z) to νHFL(Z)
  - Two approaches to νHFL(Z) validity checking
  - Fold/unfold transformation
- ♦ Machine learning techniques for CHC solving

HFL(Z) Validity Checking

A closed HFL(Z) formula (without modalities)

Remove μ, ∃ ([K+, SAS19] for first-order fragment)
(cf. Reduction from termination to safety verification)

A closed νHFL(Z) formula

Predicate abstraction [Iwayama+, 20]

Refinement types [Burn+ 18] [Katsura+, 20]

A pure νHFL formula (w/o integers)

Constrained Horn Clauses (CHC)

Higher-order model checking
[K09]
[Hosoi+, 19]
...

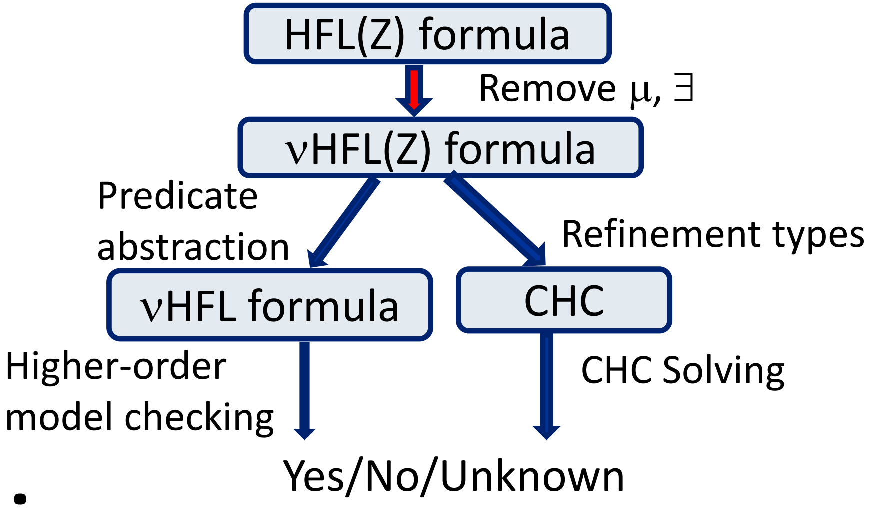
CHC Solving

Yes/No/Unknown
precise but slow

Yes/No/Unknown
imprecise but fast

Outline

- ♦ Introduction to HFL and HFL(Z)
- ♦ Reductions from program verification to HFL(Z) model checking
- ♦ HFL(Z) validity checking using types, CHC solving, and higher-order model checking
  - – Overview
  - – From HFL(Z) to νHFL(Z)
  - – Two approaches to νHFL(Z) validity checking
  - – Fold/unfold transformation



- ♦ Machine learning techniques for CHC solving

From HFL(Z) to ν-only HFL(Z)

(higher-order case: ongoing, first-order case: [K+, SAS19], inspired by termination verification [Fedyukovich+, CAV18])

- **Approximate μ by finite unfolding**

$\mu X.F(X) \geq F^n(\perp)$ (approximate)

$= (\nu X'.\lambda z.\ z>0 \wedge F(X'\ (z-1))\ )\ n$ (representation by ν)

$= \forall u \geq n.\ (\nu X'.\lambda z.\ z>0 \wedge F(X'\ (z-1))\ )\ u$ (trick to help solvers)

Example:

$\forall i.\ (\mu X.\lambda y.\ y \leq 0 \vee X(y-1))\ i$

From HFL(Z) to ν-only HFL(Z)

(higher-order case: ongoing,   first-order case: [K+, SAS19], inspired by termination verification [Fedyukovich+, CAV18])

- **Approximate μ by finite unfolding**

$\mu X.F(X) \;\geq\; F^n(\perp)$	(approximate)
$\quad = (\nu X'.\lambda z.\ z>0 \wedge F(X'\ (z-1))\ )\ n$	(representation by ν)
$\quad = \forall u \geq n.\ (\nu X'.\lambda z.\ z>0 \wedge F(X'\ (z-1))\ )\ u$	(trick to help solvers)

Example:

$\forall i.\ \underline{(\mu X.\lambda y.\ y\leq 0 \vee X(y-1))\ i}$

$\quad = i\leq 0 \vee (\mu X.\lambda y.\ ...)(i-1)$

$\quad = i\leq 0 \vee i-1\leq 0 \vee (\mu X.\lambda y.\ ...)(i-2)$

$\quad = i\leq 0 \vee i\leq 1 \vee i\leq 2 \vee ...$

From HFL(Z) to ν-only HFL(Z)

(higher-order case: ongoing, first-order case: [K+, SAS19], inspired by termination verification [Fedyukovich+, CAV18])

- Approximate μ by finite unfolding

$\mu X.F(X) \;\geq\; F^n(\bot)$ (approximate)

$= \; (\nu X'.\lambda z.\, z>0 \wedge F(X'\,(z-1))\,)\; n$ (representation by ν)

$= \forall u \geq n.\, (\nu X'.\lambda z.\, z>0 \wedge F(X'\,(z-1))\,)\; u$ (trick to help solvers)

Example:

$\forall i.\, (\mu X.\lambda y.\, y\leq 0 \vee X(y-1))\; i$

$\geq \forall i.\, \forall u \geq \max(i+1,1).\, (\nu X'.\lambda(z, y).\, z>0 \wedge (y\leq 0 \vee X'(z-1, y-1)))\, (u, i)$

$= \forall i.\, \forall u \geq \max(i+1,1).\, ((\mu X'.\lambda(z, y).\, z\leq 0 \vee (y>0 \wedge X'(z-1, y-1)))\, (u, i) \Rightarrow \text{false})$

Valid by the satisfiability of the CHC (let $X'(z,y) \equiv z \leq 0 \vee z \leq y$) :

$\{X'(z,y) \Leftarrow z\leq 0,\;\; X'(z,y) \Leftarrow y>0 \wedge X'(z-1, y-1),\;\; \text{false} \Leftarrow u\geq i+1 \wedge u\geq 1 \wedge X'(u,i)\,\}$

Mu2CHC [K+ SAS19]

- **Reduce**
  - μ-calculus properties (which subsume CTL/LTL/CTL*) of while-programs
  - LTL properties of first-order recursive programs

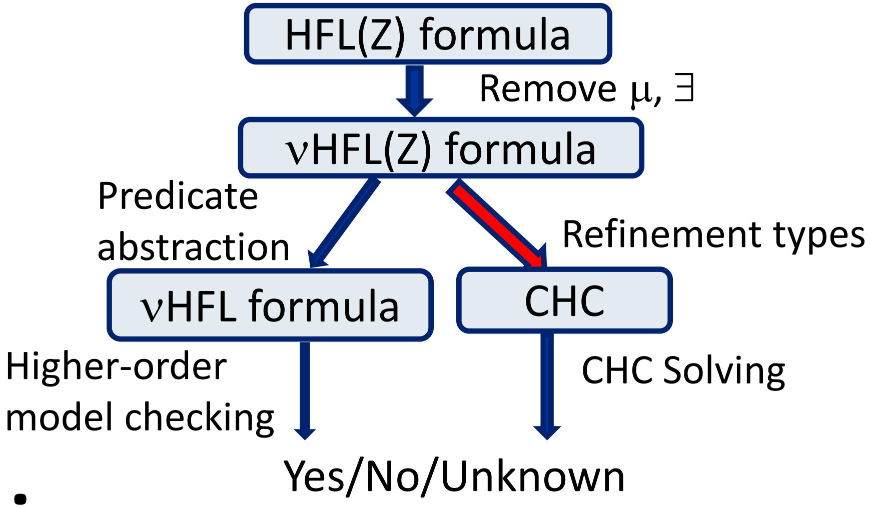
  to CHC solving via first-order HFL(Z) formulas

Experimental results on CTL verification benchmark (Cook&Koskinen [13])

Property	$\models \varphi$		$\not\models \varphi$	
	[CK13]	Mu2CHC	[CK13]	Mu2CHC
1. AG(p⇒AFq)	4.6	0.40	12.5	0.41
2. AG(p⇒AFq)	9.1	0.10	3.5	0.32
3. AG(p⇒EFq)	9.5	0.23	18.1	1.57
4. AG(p⇒EFq)	1.5	0.65	105.7	0.82
5. AG(p⇒AFq)	2.1	0.49	6.5	3.91
6. AG(p⇒AFq)	1.8	0.15	1.2	2.91
7. AG(p⇒EFq)	3.7	4.91	8.7	6.33
8. AG(p⇒EFq)	1.5	5.55	5.6	4.25
9. AG(p⇒AFq)	38.9	0.65	1930.9	3.27
10. AG(p⇒AFq)	148.0	28.20	1680.7	29.53
⋮	⋮		⋮	

Outline

- ♦ Introduction to HFL and HFL(Z)
- ♦ Reductions from program verification to HFL(Z) model checking
- ♦ HFL(Z) validity checking using types, CHC solving, and higher-order model checking
  - – Overview
  - – From HFL(Z) to νHFL(Z)
  - – From νHFL(Z) to (extended) CHC
  - – Fold/unfold transformation
- ♦ Machine learning techniques for CHC solving



Refinement Types for νHFL(Z)

[Katsura+ 20] (cf. refinement types for HoCHC [Burn+18])

τ ::= $\bullet[\psi]$ (types for propositions that hold whenever ψ holds)

$\quad|$ $x{:}\mathrm{int} \to \tau$ (dependent types for integer predicates)

$\quad|$ $\tau_1 \to \tau_2$ (non-dependent types for higher-order predicates)

ψ ::= a formula of linear integer arithmetic

Examples:

$\lambda x.x\geq 0$: $(x{:}\mathrm{int} \to \bullet[x>0])$

the type of integer predicates that are true (at least) for positive integers

$\lambda p^{\mathrm{int}\to\bullet}.p\ 1$: $(x{:}\mathrm{int} \to \bullet[x>0]) \to \bullet[\mathrm{true}]$

the type of (higher-order) predicates on integer predicates
that are true (at least) for integer predicates p such that p x holds for any x>0

Refinement Type System for νHFL(Z)

Soundness (but not completeness)

If $\vdash \varphi : \bullet[\mathbf{true}]$ then $\models \varphi$

$$\Gamma \vdash e_1 \geq e_2 : \bullet[e_1 \geq e_2]$$

$$\frac{\Gamma \vdash \varphi_1 : \bullet[\psi_1] \qquad \Gamma \vdash \varphi_2 : \bullet[\psi_2]}{\Gamma \vdash \varphi_1 \wedge \varphi_2 : \bullet[\psi_1 \wedge \psi_2]}$$

$$\frac{\Gamma \vdash \varphi : x{:}\mathbf{int} \to \tau}{\Gamma \vdash \varphi\ e : [e/x]\tau}$$

$$\frac{\Gamma \vdash \varphi : \tau' \qquad \Gamma \vdash \tau' <: \tau}{\Gamma \vdash \varphi : \tau}$$

$$\Gamma, x{:}\tau \vdash x{:}\tau$$

$$\frac{\Gamma, x{:}\tau_1 \vdash \varphi : \tau_2}{\Gamma \vdash \lambda x.\varphi : x{:}\tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash \varphi_1 : \tau_2 \to \tau \qquad \Gamma \vdash \varphi_2 : \tau_2}{\Gamma \vdash \varphi_1\ \varphi_2 : \tau}$$

$$\frac{\Gamma, X{:}\tau \vdash \varphi : \tau}{\Gamma \vdash \nu X.\varphi : \tau}$$

Refinement Type Inference
(see [Katsura+ 20] for details)

- **A standard template-based type inference algorithm yields:**
  - **CHC, for νHFL(Z) formulas obtained from (un)reachability verification problems**
    => Standard CHC solvers such as Z3 Spacer and HoIce can be used
  - ***Extended* CHC (with disjunctions in heads):**

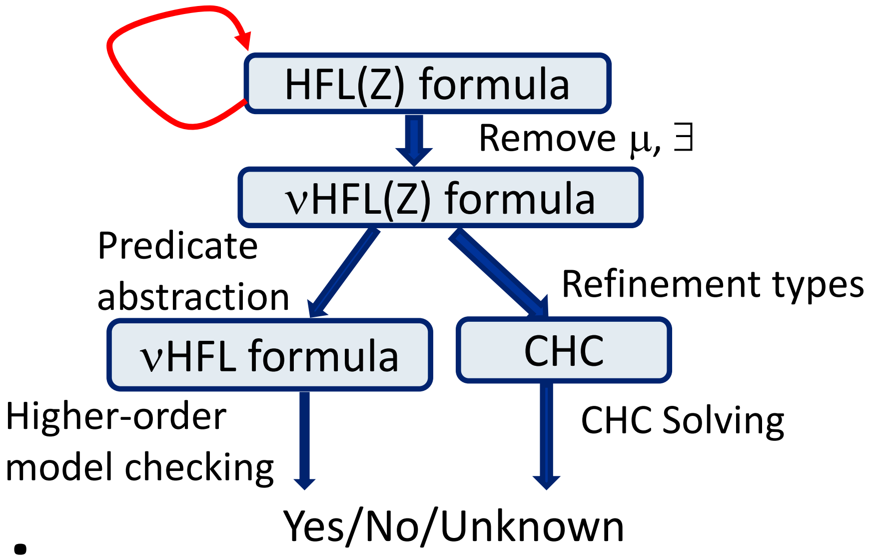
    $$H_1 \vee \ldots \vee H_k \Leftarrow B_1 \wedge \ldots \wedge B_n$$

    for general νHFL(Z) formulas

    => Extended CHC solvers such as PCSat [Unno+ 20] are required

Outline

- ♦ **Introduction to HFL and HFL(Z)**
- ♦ **Reductions from program verification to HFL(Z) model checking**
- ♦ **HFL(Z) validity checking using types, CHC solving, and higher-order model checking**
  - – **Overview**
  - – **From HFL(Z) to νHFL(Z)**
  - – **From νHFL(Z) to (extended) CHC**
  - – **Fold/unfold transformation**



- ♦ **Machine learning techniques for CHC solving**

Fold/Unfold Transformations for CHC [De Angelis+ 18]

$\mathrm{Even}(n) \Leftarrow n=0 \lor \mathrm{Even}(n\text{-}2).$

$\mathrm{false} \Leftarrow \mathrm{Even}(n) \land \mathrm{Even}(n+1).$

is SAT, but the witness requires a mod constraint: $\mathrm{Even}(n) \equiv n \bmod 2=0.$

Can we prove SAT without using the mod constraint?

Prepare a new predicate $\mathrm{E2}(n) := \mathrm{Even}(n) \land \mathrm{Even}(n+1).$

$$
\begin{aligned}
\mathrm{E2}(n) &\Leftarrow \mathrm{Even}(n) \land \mathrm{Even}(n+1) \\
&\Leftarrow \mathrm{Even}(n) \land (n+1=0 \lor \mathrm{Even}(n\text{-}1)) && \text{(unfold)} \\
&\Leftarrow (n+1=0 \land \mathrm{Even}(n)) \lor (\mathrm{Even}(n\text{-}1) \land \mathrm{Even}(n)) \\
&\Leftarrow (n+1=0 \land \mathrm{Even}(n)) \lor \mathrm{E2}(n\text{-}1) && \text{(fold)}
\end{aligned}
$$

$\mathrm{Even}(n) \Leftarrow n=0 \lor \mathrm{Even}(n\text{-}2).$

$\mathrm{E2}(n) \Leftarrow (n+1=0 \land \mathrm{Even}(n)) \lor \mathrm{E2}(n\text{-}1)$

$\mathrm{false} \Leftarrow \mathrm{E2}(n)$

has a trivial model: $\mathrm{Even}(n) \equiv n \geq 0,\ \mathrm{E2}(n) \equiv \mathrm{false}$

Fold/Unfold Transformations for HFL(Z)?

CHC

$\mathrm{Even}(n) \Leftarrow n{=}0 \vee \mathrm{Even}(n\text{-}2).$

$\mathrm{false} \Leftarrow \mathrm{Even}(n) \wedge \mathrm{Even}(n{+}1).$

Corresponding HFL(Z) formula

$\mathrm{Even}(n) \vee \mathrm{Even}(n{+}1)$ where

$\mathrm{Even}(n) =_\nu n{\neq}0 \wedge \mathrm{Even}(n\text{-}2).$

$\mathrm{Even}(n) \vee \mathrm{Even}(n{+}1)$

$= \mathrm{Even}(n) \vee (n{+}1{\neq}0 \wedge \mathrm{Even}(n\text{-}1))$ (unfold)

$= (\mathrm{Even}(n) \vee n{+}1 \neq 0) \wedge (\mathrm{Even}(n\text{-}1) \vee \mathrm{Even}(n))$

$\mathrm{E2}(n)$, where

$\mathrm{E2}(n) =_\nu (\mathrm{Even}(n) \vee n{+}1 \neq 0) \wedge \mathrm{E2}(n\text{-}1)$

$\mathrm{Even}(n) =_\nu n{\neq}0 \wedge \mathrm{Even}(n\text{-}2).$

Q: Is fold/unfold transformation applicable to arbitrary alternations of μ and ν?

A: Yes, but with a certain sanity condition.

See [K+, TACAS 20] for first-order HFL(Z).

See also [Kori+, CSL 21] on cyclic proofs for HFL.

Outline

- ♦ **Introduction to HFL and HFL(Z)**
- ♦ **Reductions from program verification to HFL(Z) model checking**
- ♦ **HFL(Z) validity checking using types, CHC solving, and higher-order model checking**
- ♦ **Machine learning techniques for CHC solving**
  - – ICE learning for CHC
  - – Neural networks for qualifier discovery

HoIce: ICE-Learning-Based CHC Solver

[Champion+ TACAS18]

- **An extension of the ICE-learning framework [Garg+ CAV14]**

Even(n) $\Leftarrow$ n=0 $\vee$ Even(n-2).
false $\Leftarrow$ Even(n) $\wedge$ Even(n+1).

Teacher
(call SMT to check candidate models)

learning data
(e.g. {Even(0), Even(1) $\Rightarrow$ Even(3), ...})

Learner
(qualifier discovery + Boolean function synthesis)

candidate model
(e.g. Even(x) $\equiv$ x>0)

- **A reasonably good performance**
- **Tend to generate simple models**
(=> suitable for refinement type inference, with applications also in higher-order model checking [Sato+ 19])

Solver	Score	#SAT	#UNSAT	Avg time
Spacer	270	153	117	5.04
Eldarica	**234**	131	103	15.93
Ultimate Unihorn Automizer	**177**	96	81	36.94
Hoice	**176**	110	66	9.85
PCSat	**123**	81	42	24.69
Ultimate Tree Automizer	**73**	29	44	4.85

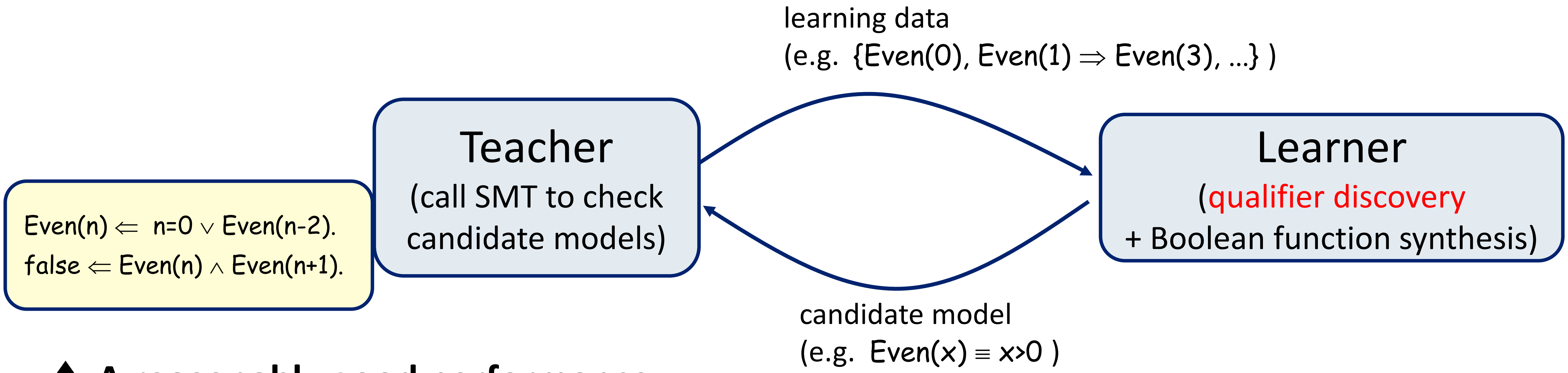
(Result of CHC-Comp19, LIA-Nonlin category)
From https://chc-comp.github.io/2019/chc-comp19.pdf)

* 283 instances total

HoIce: ICE-Learning-Based CHC Solver

[Champion+ TACAS18]

- **An extension of the ICE-learning framework** **[Garg+ CAV14]**



- **A reasonably good performance**
- **Tend to generate simple models**
  (=> suitable for refinement type inference, with applications also in higher-order model checking [Sato+ 19])
- **Applicable to extended CHCs**
  (with disjunctions in heads)

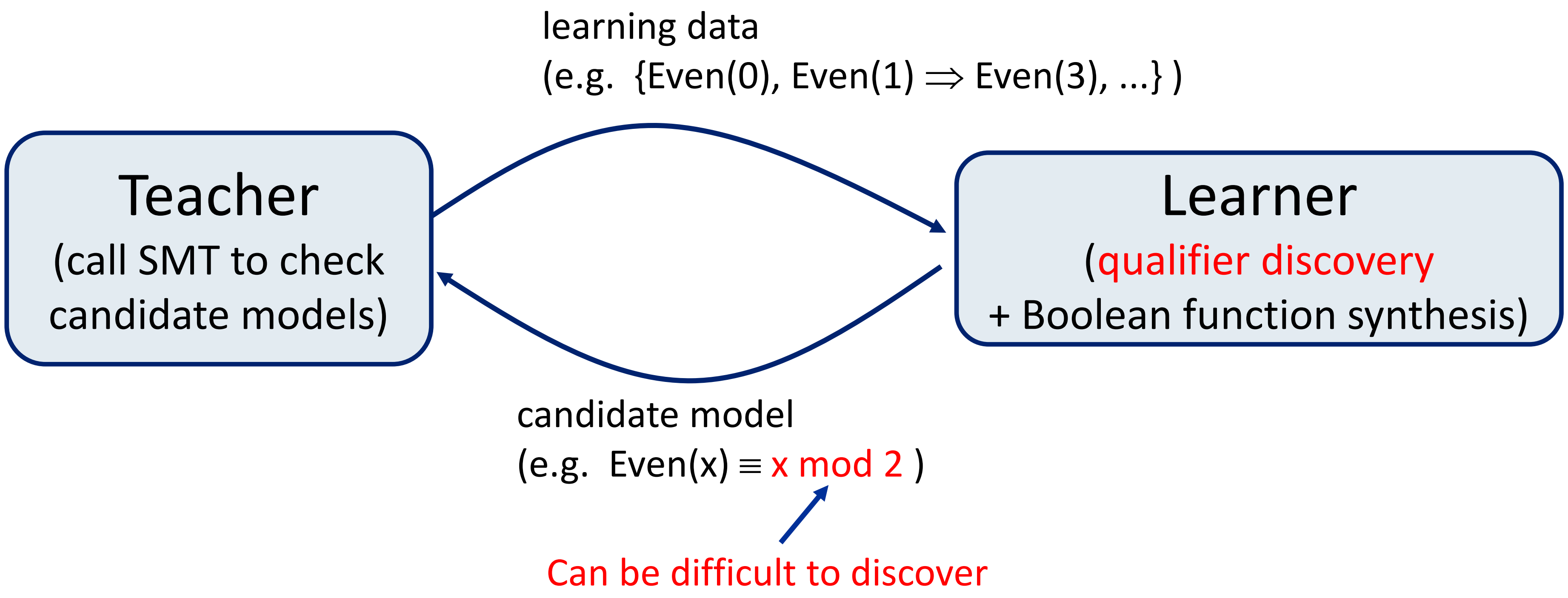
Solver	Score	#SAT	#UNSAT	Avg time
Spacer	270	153	117	5.04
Eldarica	**234**	131	103	15.93
Ultimate Unihorn Automizer	**177**	96	81	36.94
Hoice	**176**	110	66	9.85
PCSat	**123**	81	42	24.69
Ultimate Tree Automizer	**73**	29	44	4.85

(Result of CHC-Comp19, LIA-Nonlin category)
From https://chc-comp.github.io/2019/chc-comp19.pdf)

* 283 instances total

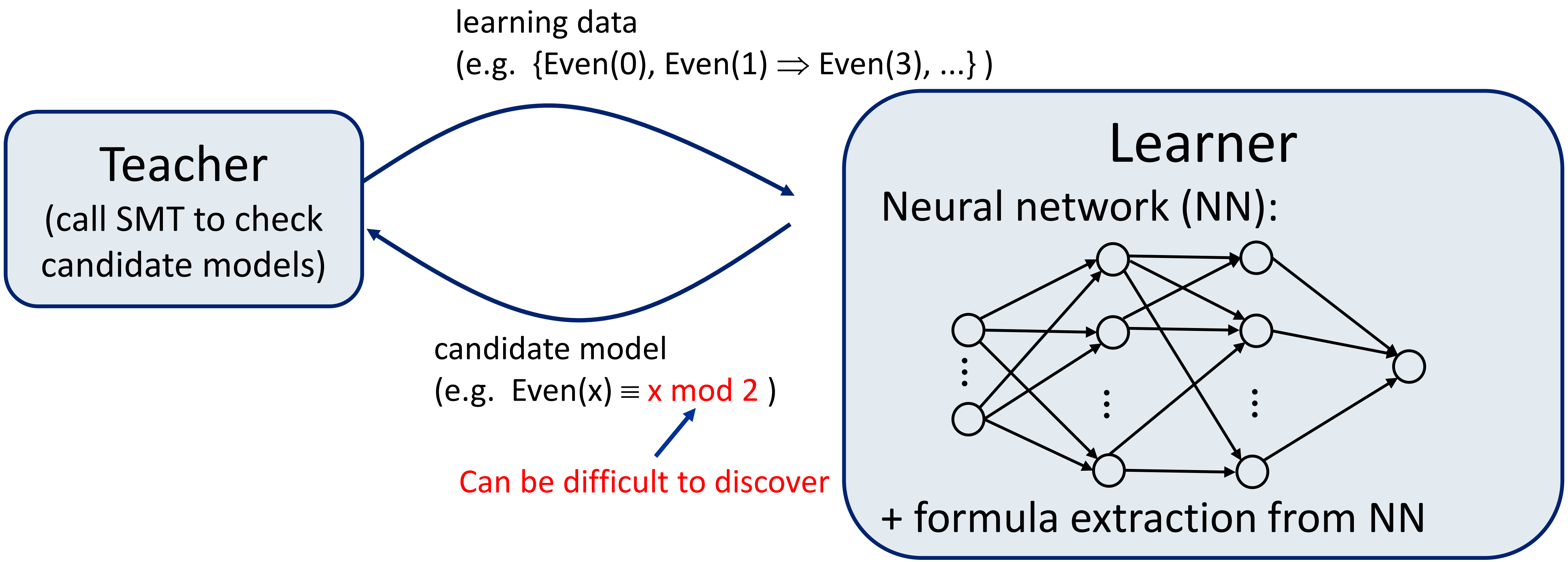
Neural Networks for Qualifier Discovery

(Ongoing work)



Neural Networks for Qualifier Discovery

(Ongoing work)



Conclusion

- ♦ **Automated program verification project based on HFL(Z)**
  - **HFL(Z) may be viewed as an extension of CHC with higher-order predicates and fixpoint alternation**
  - **Provide a uniform verification framework for temporal properties (safety, termination, liveness, ...) of higher-order functional programs**
  - **Many of the existing techniques for CHC solving and program verification can be lifted to those for HFL(Z) validity checking (e.g. fold/unfold transformation)**
  - **CHC solvers are important building blocks for HFL(Z) validity checking**
    - **Sometimes we need more than pure CHC solving (e.g. disjunctions in heads, for refinement-type-based approach to HFL(Z) validity checking)**
    - **Any improvement of CHC solvers would be appreciated!**

References

- Relationship between HORS/HFL model checking
  - K, Lozes, Bruze, “On the relationship between higher-order recursion schemes and higher-order fixpoint logic”, POPL 17
- (pure) HFL model checking algorithm
  - Hosoi, K, Tsukada, “A Type-Based HFL Model Checking Algorithm”, APLAS 19
- From program verification to HFL(Z) model/validity checking
  - K, Tsukada, Watanabe, “Higher-Order Program Verification via HFL Model Checking”, ESOP 18
  - Watanabe, Tsukada, K, “Reduction from branching-time property verification of higher-order programs to HFL validity checking”, PEPM 19
- From (first-order) HFL(Z) to ν-only HFL(Z)
  - K, Nishikawa, Igarashi, Unno, “Temporal Verification of Programs via First-Order Fixpoint Logic”, SAS 19

References

- Solving ν-only HFL(Z) validity checking
  - Iwayama, K, Suzuki, Tsukada, “Predicate Abstraction and CEGAR for νHFL_Z Validity Checking”, SAS 20
  - Katsura, Iwayama, K, Tsukada, “A New Refinement Type System for Automated νHFL_Z Validity Checking”, APLAS 20
- Fold/unfold transformation for (first-order) HFL(Z)
  - K, Fedyukovich, Gupta, “Fold/Unfold Transformations for Fixpoint Logic”, TACAS 20
  - (related) Kori, Tsukada, K, “A Cyclic Proof System for HFL_N”, CSL 21
- Machine learning techniques for CHC solving
  - Champion, Chiba, K, Sato, “ICE-Based Refinement Type Discovery for Higher-Order Functional Programs”, TACAS 18