Chronosymbolic Learning

Efficient CHC Solving with Symbolic Reasoning and Inductive Learning

Ziyan “Ray” Luo [1,2] and Xujie Si [1,3]

[1] Mila, [2] McGill University, [3] University of Toronto

Table of contents

What are CHCs? (Constrained Horn Clauses)

Why we need to solve CHCs?

How about the **current CHC solvers**?

How do these solvers **motivate** our method?

Key concepts and overall **architecture** of proposed framework

A simplistic **instantiation** of our framework

Experimental result

Future work, Q & A

What are CHCs?

SAT Problem

$$(x \land y) \lor (x \land \neg z)$$

Logical implication: $p \rightarrow q \triangleq \neg p \lor q$

SMT Problem

$$\forall a, b, c,\ \underline{a > 0} \land \underline{b \leq a} \land \underline{c = 0} \rightarrow a + b + c \geq 0$$

Constrained Horn Clause (CHC)

$$\forall a, b, c,\ a > 0 \land b \leq a \land c = 0 \rightarrow p(a, b, c)$$

CHC System

$$\mathcal{C}_0 : \forall a, b, c,\ a > 0 \land b \leq a \land c = 0 \rightarrow p(a, b, c)$$

$$\mathcal{C}_1 : \forall a, b, c, c_1,\ c_1 = 1 + c \land p(a, b, c) \rightarrow q(a, b, c_1)$$

$$\mathcal{C}_2 : \forall a, b, c,\ b < a \cdot c \land q(a, b, c) \rightarrow \bot$$

Problem: find (synthesize) such interpretations of p, q, or prove it UNSAT!

(Solution interpretations, Def. 1)

CHC System, Terminologies

Unknown predicate symbols (relation)

$$\forall \mathcal{V} \cdot (\varphi \wedge p_1(X_1) \wedge \cdots \wedge p_k(X_k) \rightarrow h(X)), \text{ for } k \geq 1$$

For all variables ("Constrained") Conjunction Implication p->q means ~p or q

$$\forall \mathcal{V} \cdot \{(\varphi \wedge p_1(X_1) \wedge \cdots \wedge p_k(X_k)\} \rightarrow h(X)), \text{ for } k \geq 1$$

Can be omitted Body Head

$\mathcal{C}_0 : \forall a, b, c, \ a > 0 \ \wedge \ b \leq a \ \wedge \ c = 0 \rightarrow p(a, b, c)$	Fact (also a Rule)
$\mathcal{C}_1 : \forall a, b, c, c_1, \ \boxed{c_1 = 1 + c} \wedge p(a, b, c) \rightarrow q(a, b, c_1)$	Rule
$\mathcal{C}_2 : \forall a, b, c, \ b < a \cdot c \ \wedge \ q(a, b, c) \rightarrow \perp$	Query

For simplicity, trivial rules that have pre-determined truth value are omitted

CHC solver can be a *backbone* for program verification

- CHC system is SAT ⇔ Program is safe
- Cast the program verification problem as constraint solving problem
- Naturally handles ***loop invariant** generation* problem (but not limited to)

> Known as the biggest challenge in program verification!

An abstract interpretation of a loop

$$x > 1 \land y = 0 \rightarrow p(x, y) \quad (1)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow p(x', y') \quad (2)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow x' >= y' \quad (3)$$

$$x > 1 \land y = 0 \rightarrow x >= y \quad (4)$$

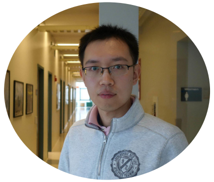
```
...
assume(x > 1);
int y = 0;
while (*){
    x = x + y;
    y = y + 1;
}
assert(x >= y);
```

Pre-condition: `assume(x > 1); int y = 0;`

Command: `while (*){ ... }` — Some nondeterministic expression of x, y

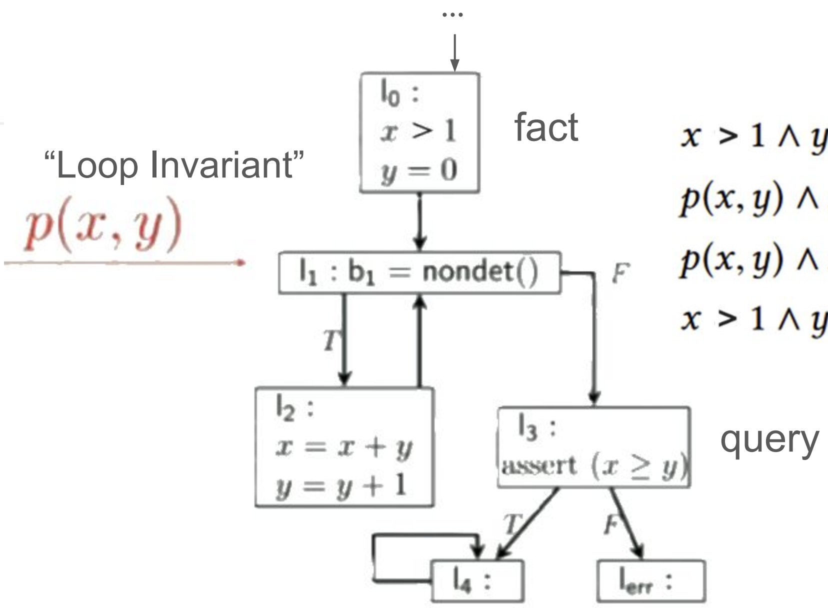
Post-condition: `assert(x >= y);`

CHC solver can be a *backbone* for program verification



Program | **CFG** | **CHCs**

```
...
assume(x > 1);
int y = 0;
while (*){
    x = x + y;
    y = y + 1;
}
assert(x >= y);
```



$$x > 1 \land y = 0 \rightarrow p(x,y) \quad (1)$$

$$p(x,y) \land x' = x + y \land y' = y + 1 \rightarrow p(x',y') \quad (2)$$

$$p(x,y) \land x' = x + y \land y' = y + 1 \rightarrow x' >= y' \quad (3)$$

$$x > 1 \land y = 0 \rightarrow x >= y \quad (4)$$

We will use programs to explain CHCs

Table 5. Mapping of program verification concepts to CHC solving concepts.

Program Verification Concept	CHC Solving Concept
Verification Conditions (VCs)	CHC system $\mathcal{H}$
Initial program state	Fact $\mathcal{C}_f$
Transition	Rule $\mathcal{C}_r$
Assertion/Unsafe condition	Query $\mathcal{C}_q$
Inductive invariant/Loop invariant	Solution Interpretation $\mathcal{I}^* [p]$ to predicate p
Counterexample trace	Refutation proof $\mathcal{R}$
Reachable program states	Positive samples s^+
Unsafe program states	Negative samples s^-

“Symbolic approaches”

White-box vs. black-box approaches

- **Bounded model checking (BMC):** encode the initial states, k loop transitions, and bad states into logical formula (Can prove UNSAFE if the formula is SAT)

$$Init(V) \wedge \underbrace{Tr(V, V') \wedge Tr(V', V'') \wedge \ldots \wedge Tr(V^{(k-1)}, V^{(k)})}_{k} \wedge Bad(V^{(k)})$$

```
main(){
 int x,y;
 x=1; y=0;
 while(*){
  x=x+y;
  y++;}
 assert(x>=y);}
```

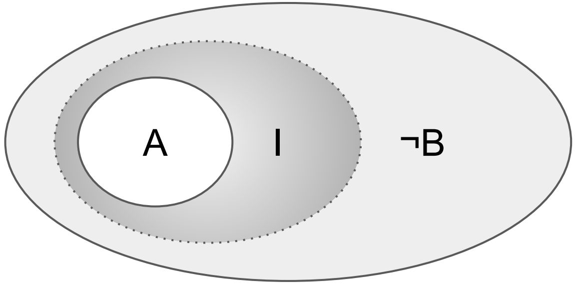
Some nondeterministic expression of x, y (arrow pointing to the `*` in `while(*)`)

SMT encoding for k=2:

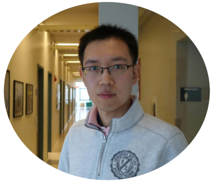
$$x = 1, y = 0 \wedge x_1 = x + y, y_1 = y + 1 \wedge x_2 = x_1 + y_1, y_2 = y_1 + 1 \wedge x_2 < y_2$$

- **Unbounded model checking:** find a fixed-point through inductive generalization heuristics (e.g., Craig interpolants (CI), *IC3*, PDR, Spacer, GSpacer)
  - ✓ Efficiently maximize the power of the solvers
  - X Often rely on heuristics to do inductive generalization
  - X Not flexible with data samples (which is cheap sometimes)

$A \Rightarrow I \qquad I \Rightarrow \neg B$

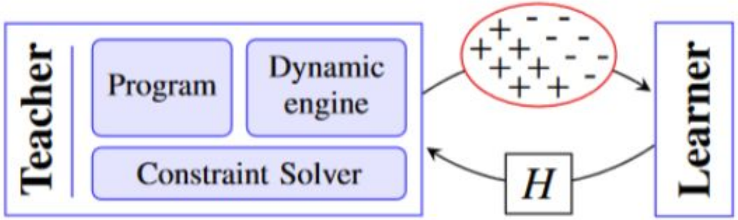


White-box vs. black-box approaches

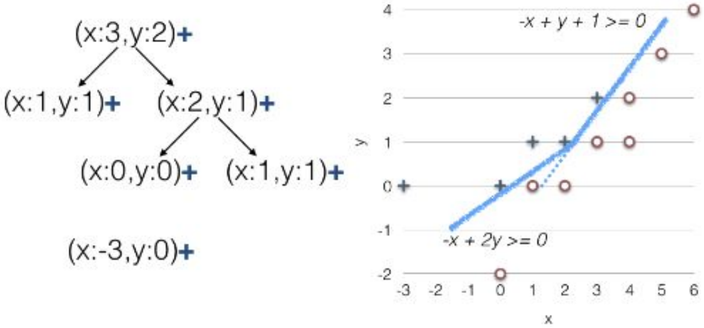


- *Teacher and Learner* paradigm, "Guess-and-check"
- Hypothesize interpretations by induction learning
- Iteratively refine the hypothesis

✓ Can take global information into account, better *generalization* (using ML rather than CI)
✓ For arbitrary guesses, it's relatively easy to get data samples as counterexamples
x Cheap prior knowledge in CHC systems is ignored
x State explosion issue exists



ICE: A Robust Framework for Learning Invariants, Garg et al., 2014



A Data-Driven CHC Solver, Zhu et al., 2018 (LinearArbitrary)

Motivation

- Black-box approaches is **sample inefficient**
- White-box methods is 1) difficult to deal with data samples
  2) hard to do inductive generalization

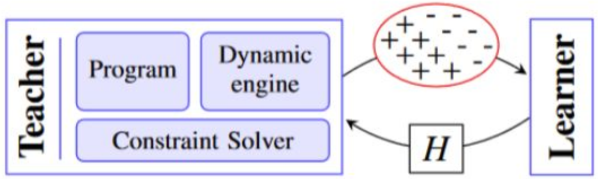
```
...
assume(x > 1);
int y = 0;
while (*){
    x = x + y;
    y = y + 1;
}
assert(x >= y);
```

$$x > 1 \land y = 0 \rightarrow p(x, y) \quad (1)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow p(x', y') \quad (2)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow x' >= y' \quad (3)$$

$$x > 1 \land y = 0 \rightarrow x >= y \quad (4)$$



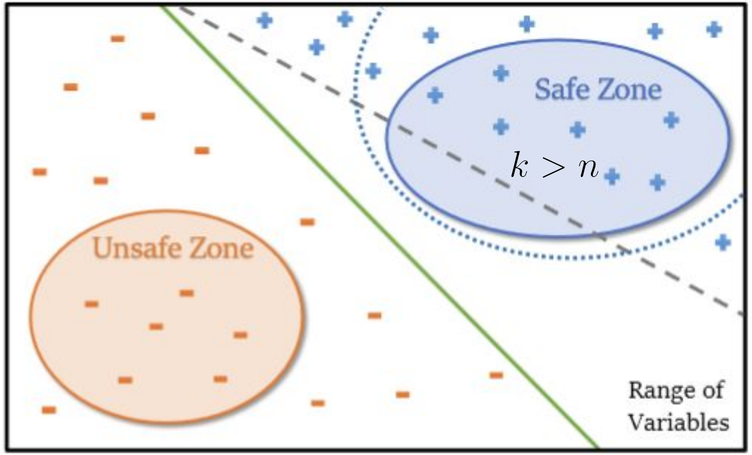
- Need many samples to "relearn" $x > 1 \land y = 0$
- Each learning iteration gives only 1 additional sample
- Might have infinite interpretations for a set of positive and negative samples

Motivation

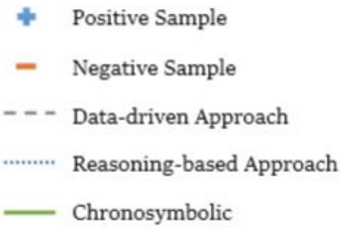
Can we find a way to finding global patterns of CHCs *without the resort to any hand-crafted heuristics?*

Can we make the data-driven methods more sample efficient?

Can we present symbolic and data-driven methods in a *unified framework*?



Desideratum: green solid line



Contributions

- Identify and formulate the key concepts in CHC solving: **samples, zones**, **counterexample**, and etc. Their connection is also discussed.
    - Enable *synergistic* working for symbolic methods and learning-based methods
    - Here we don't assume any specific algorithm for learner and reasoner
- Design a modular framework, Chro**no***symbolic Learning* to realize the desiderata
    - Naming: Synchronous, CHC, symbolic, no. (number)
- Propose a minimal instance, showing how components interact in our framework
- Provide artifacts for the minimal instance
- Give an evaluation of the instance and show the potential

(Inspired by and generalized from program verification)

Samples

Definition 4 (Positive Sample). *A data point s^+ is a positive sample of predicate p in $\mathcal{H}$ iff $p(s^+) = \top$ must hold to make all rules in $\mathcal{H}$ SAT.*

```
...
assume(x > 1);
int y = 0;
while (*){
    x = x + y;
    y = y + 1;
}
assert(x >= y);
```

rules

$$x > 1 \land y = 0 \rightarrow p(x, y)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow p(x', y')$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow x' >= y'$$

query

$$x > 1 \land y = 0 \rightarrow x >= y$$

For predicate p(x, y)

- Positive samples:

  (2, 0), (2, 1), (3, 2), (5, 3), …

- Negative samples:

  (2, 3), (0, 2), (-174732, 123), …

- Implication samples:

  ((2, 0), (2, 1)), ((-1, 0), (-1, 1)), …

(Forward) Reachable program configurations

Program configurations that is unsafe (backward reachable starting from the unsafe condition)

Not necessarily samples that appears in program
But should *depict the "semantics" of the loop*

Zones

Definition 7 (Safe and Unsafe Zones). *A safe (unsafe) zone of a predicate p, $\mathcal{S}_p$ ($\mathcal{U}_p$), is a set of positive (negative) samples of p.*

- Zone is a set of samples; samples are zones (with cardinality of 1)
- Sometimes it's easy to know some zones (Lemma 7,8)

```
...
assume(x > 1);
int y = 0;
while (*){
    x = x + y;
    y = y + 1;
}
assert(x >= y);
```

$$x > 1 \wedge y = 0 \rightarrow p(x, y)$$

$$p(x, y) \wedge x' = x + y \wedge y' = y + 1 \rightarrow p(x', y')$$

$$p(x, y) \wedge x' = x + y \wedge y' = y + 1 \rightarrow x' >= y'$$

$$x > 1 \wedge y = 0 \rightarrow x >= y$$

Zones can be symbolically represented:

For predicate p(x, y)

- Example of a safe zone: $\{x > 1 \wedge y = 0\}$
- Example of an unsafe zone: $\{x < y\}$

  (Or any zones that can be implied by current zones, conceptually)

Some immediate useful lemmas

- Connection of solutions and samples

Lemma 1. *If $\mathcal{H}$ is SAT, for each solution interpretation $\mathcal{I}^*$ of $\mathcal{H}$,*

(1) if s^+ is a positive sample of p in $\mathcal{H}$, we have $\mathcal{I}^*[p](s^+) = \top$.
(2) if s^- is a negative sample of p in $\mathcal{H}$, we have $\mathcal{I}^*[p](s^-) = \bot$.
(3) if $s^{\rightarrow} = (s_1^{\rightarrow}, \cdots, s_n^{\rightarrow}, s_h^{\rightarrow})$ is a implication sample of body predicates $(p_1, ..., p_n)$ and head predicate h in $\mathcal{H}$, $\mathcal{I}^*[p_1](s_1^{\rightarrow}) \wedge \cdots \wedge \mathcal{I}^*[p_n](s_n^{\rightarrow}) \rightarrow \mathcal{I}^*[h](s_h^{\rightarrow})$.

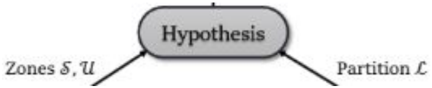
A valid solution interpretation should separate any positive and negative samples.

- Valid zones can directly extracted from the fact and query

Lemma 7 (Initial Safe Zones). *A fact $\mathcal{C}_f : \phi_f \rightarrow h_f(T)$ produces a safe zone for h_f: $\mathcal{S}^0_{h_f}(T) = \exists \mathcal{X}_\varphi, \phi_f$.*

Lemma 8 (Initial Unsafe Zones). *A linear query $\mathcal{C}_q : \phi_q \wedge p_q(T) \rightarrow \bot$ produces an unsafe zone for p_q: $\mathcal{U}^0_{p_q}(T) = \exists \mathcal{X}_\varphi, \phi_q$.*

Why we need Zones?



1. **Integrated into the learner's hypothesis to enhance it (Lemma 2)**

   Add S: $x > 1 \land y = 0$ to the hypothesis, and the new data samples will never be in $x > 1 \land y = 0$

2. **Provide the learner with additional samples (Sampling)**

   Sample from S: $x > 1 \land y = 0$ to get positive samples like (2, 0), (100, 0), …

3. **Simplify the UNSAT checking of the CHC system**

   We assume there is a solution interpretation and make hypotheses, until there is a **conflict**

   Lemma 5 (sample-sample conflict): samples cannot be both positive and negative

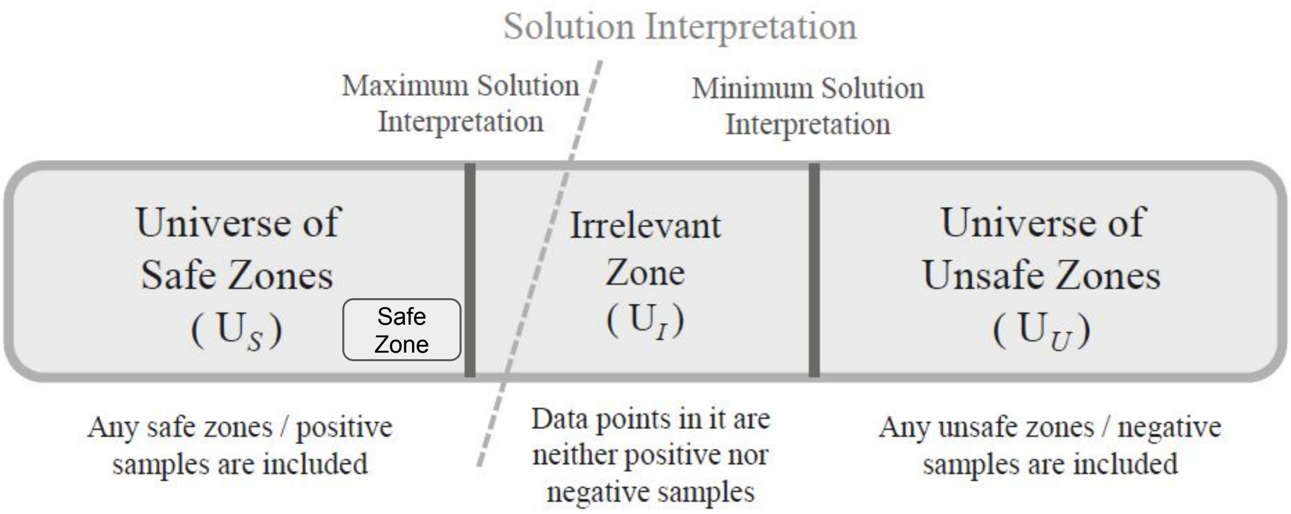
   Lemma 6 (sample-zone conflict): a positive sample cannot be in unsafe zones

   Lemma 11: safe and unsafe zones cannot overlap

Solution space of a CHC system

The universes of safe and unsafe zones are disjoint

Zones ensure more efficient “data coverage” of Us and Uu than samples



Counterexamples

Definition 8 (Counterexample). *A counterexample* $c = ((s_{p_1}, \cdots, s_{p_k}), s_h)$ *for a CHC* C *and an interpretation* $\mathcal{I}$ *is a set of data points*[15] *such that under* c, $\mathcal{I}[C'] = \bot$, *where* C' *is the same constraint as* C *but without the quantifier.*

```
...
assume(x > 1);
int y = 0;
while (*){
    x = x + y;
    y = y + 1;
}
assert(x >= y);
```

- A situation that current interpretation fails
- Formulate the information the teacher provides to the learner
- Can be **converted** into positive / negative samples (Lemma 3,4)
- The information can potentially extended to zonal representation

$$x > 1 \land y = 0 \rightarrow p(x, y) \quad (1)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow p(x', y') \quad (2)$$

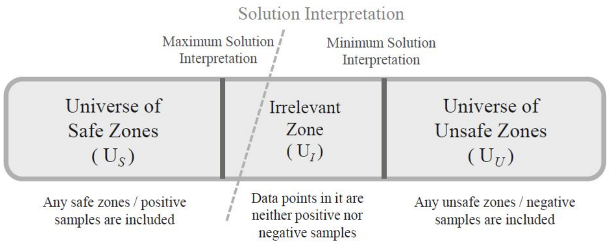
$$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow x' >= y' \quad (3)$$

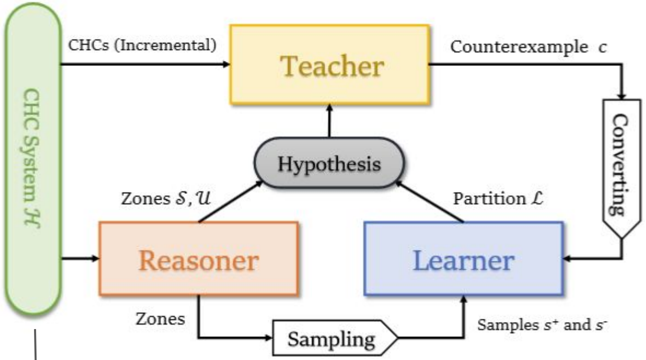
$$x > 1 \land y = 0 \rightarrow x >= y \quad (4)$$

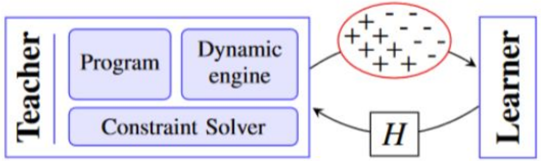
For CHC (2) and p(x, y) ≡ x = y + 2:

- Example of a counterexample: (((2,0)), (2,1))
- It makes CHC (2) $\top \rightarrow \bot$
- SMTModel(¬(CHC (2) ∧ p(x, y) ≡ x = y + 2))

Chronosymbolic Learning







$x > 1 \land y = 0 \rightarrow p(x, y)$

$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow p(x', y')$

$p(x, y) \land x' = x + y \land y' = y + 1 \rightarrow x' >= y'$

$x > 1 \land y = 0 \rightarrow x >= y$

In each iteration:

- The reasoner and learner propose new zones and partition based on current zones and samples
- Make a hypothesis based on them
- The teacher checks the satisfiability for one certain interpreted CHC
- If SAT, *switch* to another CHC
- If not SAT, return a *counterexample*, *convert* it into data samples
- Do *UNSAT checking for CHC system* (Lemma 5, 6, 11) using samples and zones
- Sample data from zones
- If all CHC is SAT, we find the solution interpretation

Instantiations of Chronosymbolic Learning

- Previous methods can be seen as special instances of our framework
- (6): Better state-space coverage and more efficient UNSAT checking (Lemma 2)
- For some instances, w/o S/U (4, 5) is better
  - Can be seen as different exploration strategies

Table 1. Several candidates of making hypotheses.

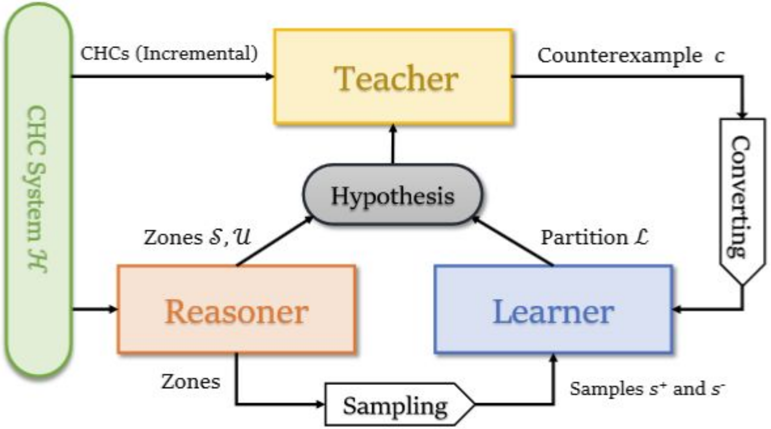
Methods	Candidate Hypothesis	
BMC-styled	$\tilde{\mathcal{I}}_s\,[p_i] = \mathcal{S}_{p_i}$	(2)
LinearArbitrary-styled	$\tilde{\mathcal{I}}_l\,[p_i] = \mathcal{L}_{p_i}$	(3)
Chronosymbolic w/o safe zones	$\tilde{\mathcal{I}}_{lu}\,[p_i] = \mathcal{L}_{p_i} \wedge \neg\, \mathcal{U}_{p_i}$	(4)
Chronosymbolic w/o unsafe zones	$\tilde{\mathcal{I}}_{sl}\,[p_i] = \mathcal{S}_{p_i} \vee \mathcal{L}_{p_i}$	(5)
Chronosymbolic	$\tilde{\mathcal{I}}_{slu}\,[p_i] = \mathcal{S}_{p_i} \vee (\mathcal{L}_{p_i} \wedge \neg\, \mathcal{U}_{p_i})$	(6)

Our instance of Chronosymbolic Learning

- A data-driven learner (L) + a BMC-styled reasoner (S, U)
- makeHypothesis() is done by:
  - Chronosymbolic-single: Always using (6)
  - Chronosymbolic-cover: alternates from (2, 3, 4, 5, 6) using some scheduling heuristics
  - Ablation study: Always using (2, 3, 4, 5)

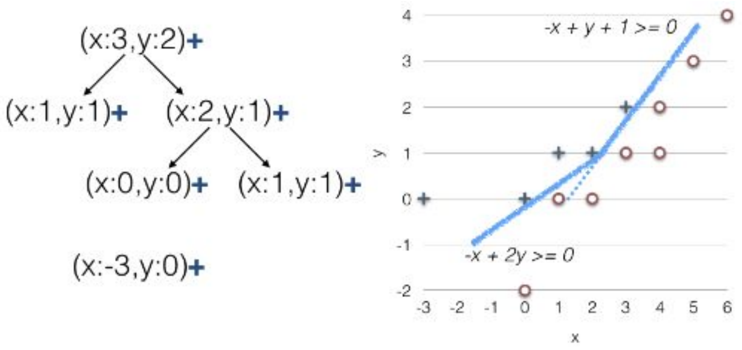
Table 1. Several candidates of making hypotheses.

Methods	Candidate Hypothesis	
BMC-styled	$\tilde{\mathcal{I}}_s\,[p_i] = \mathcal{S}_{p_i}$	(2)
LinearArbitrary-styled	$\tilde{\mathcal{I}}_l\,[p_i] = \mathcal{L}_{p_i}$	(3)
Chronosymbolic w/o safe zones	$\tilde{\mathcal{I}}_{lu}\,[p_i] = \mathcal{L}_{p_i} \wedge \neg\,\mathcal{U}_{p_i}$	(4)
Chronosymbolic w/o unsafe zones	$\tilde{\mathcal{I}}_{sl}\,[p_i] = \mathcal{S}_{p_i} \vee \mathcal{L}_{p_i}$	(5)
Chronosymbolic	$\tilde{\mathcal{I}}_{slu}\,[p_i] = \mathcal{S}_{p_i} \vee (\mathcal{L}_{p_i} \wedge \neg\,\mathcal{U}_{p_i})$	(6)



Learner

- Collect the counterexample and convert them to positive and negative samples
- The learner contains a *dataset* and a *machine learning toolchain (SVM+DT)*
  - SVM (learn arbitrary hyperplanes) + Decision Tree (tune and recombine those hyperplanes)
  - Refer to our paper for further details



A Data-Driven CHC Solver, Zhu et al., 2018
(LinearArbitrary)

Reasoner

- BMC-styled image/pre-image computation

$$Init(V) \wedge \underbrace{Tr(V, V') \wedge Tr(V', V'') \wedge \ldots \wedge Tr(V^{(k-1)}, V^{(k)})}_{k} \wedge Bad(V^{(k)})$$

Append a transition to the current zone to expand the zone

Lemma 9 (Forward Expansion). *From given safe zones $\mathcal{S}_{p_i}^{m}$, we can expand them in one forward transition by a non-fact rule $\mathcal{C}_r : \phi \wedge p_1(T_1) \wedge \cdots \wedge p_k(T_k) \rightarrow h(T)$ to get an expanded safe zone $\mathcal{S}_h^{m+1}$, where $\mathcal{S}_h^{m+1}(T) = \exists \mathcal{X}_\varphi, \phi \wedge \mathcal{S}_{p_1}^{m}(T_1) \wedge \cdots \wedge S_{p_k}^{m}(T_k)$.*

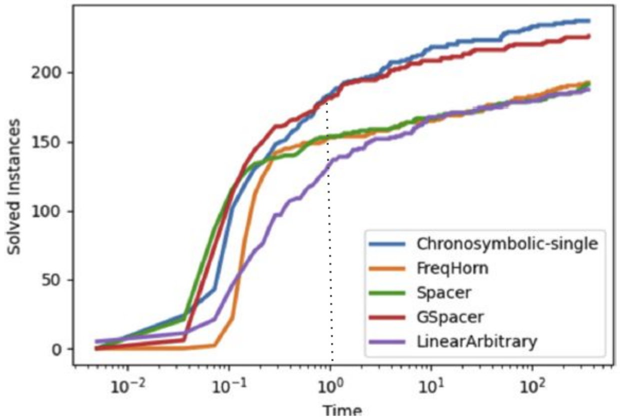
Lemma 10 (Backward Expansion). *From a given unsafe zone $\mathcal{U}_h^m$, we can expand it in one backward transition by a non-fact linear rule $\mathcal{C}_r : \phi \wedge p(T_0) \rightarrow h(T)$ to get an expanded unsafe zone $\mathcal{U}_p^{m+1}$, where $\mathcal{U}_p^{m+1}(T_0) = \exists \mathcal{X}_\varphi, \phi \wedge \mathcal{U}_h^m$.*

Major experiment settings

- 288 arithmetic instances collected by FreqHorn, including non-linear ones
- Timeout: 360s (but we also care about efficiency within this period)
- Chronosymbolic-single: one configuration (hyperparameter set and strategy) for all instances (Meta-Learning-liked)
- Chronosymbolic-cover: all solved instances in 13 configurations (like LinearArbitrary which sets specific hyperparameters for different instances)

Experiment

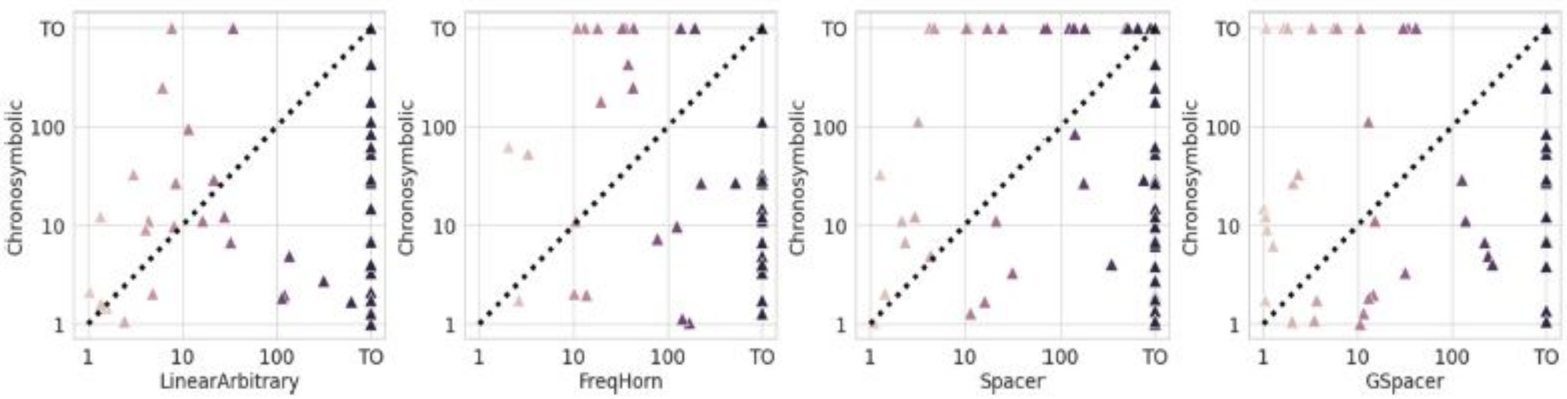
- Efficient in arithmetic benchmarks
- Meet challenges in benchmarks with many Bool vars
- On our main dataset, the average time for SVM, DT, the teacher and reasoner are 26.68s, 4.74s, 14.58s,1.29s respectively



Method	#total	percentage	#safe	#unsafe	avg-time (s)	avg-time-solved (s)
LinearArbitrary	187	64.93%	148	39	135.0	13.48
FreqHorn	191	66.32%	191	0	129.1	11.80
FreqHorn-expl	50	17.36%	0	50	299.5	13.57
Spacer	184	63.89%	132	52	132.8	15.30
GSpacer	220	76.39%	174	46	83.50	7.83
Chronosymbolic-single	**237**	**82.29%**	**189**	**48**	**68.33**	**7.51**
Chronosymbolic-cover	**252**	**87.50%**	**204**	**48**	-	-

Experiment

Below the diagonal: ours > baseline



(seconds)

Ablation

BMC-styled	$\tilde{\mathcal{I}}_s[p_i] = \mathcal{S}_{p_i}$	(2)
LinearArbitrary-styled	$\tilde{\mathcal{I}}_l[p_i] = \mathcal{L}_{p_i}$	(3)
Chronosymbolic w/o safe zones	$\tilde{\mathcal{I}}_{lu}[p_i] = \mathcal{L}_{p_i} \wedge \neg\,\mathcal{U}_{p_i}$	(4)
Chronosymbolic w/o unsafe zones	$\tilde{\mathcal{I}}_{sl}[p_i] = \mathcal{S}_{p_i} \vee \mathcal{L}_{p_i}$	(5)
Chronosymbolic	$\tilde{\mathcal{I}}_{slu}[p_i] = \mathcal{S}_{p_i} \vee (\mathcal{L}_{p_i} \wedge \neg\,\mathcal{U}_{p_i})$	(6)

Table 4. Different configurations of CHRONOSYMBOLIC LEARNING.

Configuration		#total	percentage	#safe	#unsafe	avg-time (s)	avg-time-solved (s)
without safe zones	(4)	228	79.17%	183	45	78.56	9.11
without unsafe zones	(5)	218	75.69%	173	45	94.75	12.87
without both zones	(3)	211	73.26%	166	45	93.84	10.09
without learner	(2)	131	45.49%	96	35	196.3	0.16
parallel		216	75.00%	180	36	–	–
Chronosymbolic-single		**237**	**82.29%**	**189**	**48**	**68.33**	**7.51**

Parallel: learner and reasoner running individually and simultaneously for 360s

Future work and discussion

- Better Learner:
  - Some better symbolic classifier that can deal with zones and samples simultaneously
  - Better handling a large number of Boolean variables
  - Embracing the LLMs as symbolic classifiers
- Better Reasoner:
  - Go beyond BMC; e.g., model-based projection
  - Add support for approximated zones (IC3/PDR),
  - Zones induced from samples (symbolic regression)
  - Better procedure to simplify complicated zones
  - Currently reasoner can benefit from the learner, but learner’s impact on reasoner is small
- Better Teacher:
  - Can produce zonal feedback / counterexample
- Better support for NL CHCs
- Beyond Integer Arithmetics

Thank you for your careful listening!

- Code is available here: https://github.com/Chronosymbolic/Chronosymbolic-Learning, with examples on how it works, and the detailed experimental results
- Slides can be downloaded from my personal website https://zyluo.netlify.app/
- Email me (ziyan.luo@mail.mcgill.ca) or Xujie (six@cs.toronto.edu) for further questions or comments

Paper:

Code: