Constraint Solving and Machine Learning for Program Verification and Synthesis

Hiroshi Unno

University of Tsukuba / RIKEN AIP

www.cs.tsukuba.ac.jp/~uhiro

Research Interests

- Formal specification, ***verification***, and ***synthesis***

  of (mainly but not limited to) higher-order functional programs

  by **AI techniques** such as ***constraint solving*** and ***machine learning***

- Ongoing projects
  - ***Synthesis*** of High-Level Programs from Temporal and Relational Specifications (PI: Hiroshi Unno)
  - Program ***Verification*** Techniques for the **AI** Era (PI: Naoki Kobayashi)
  - **AI** Security and Privacy (PI: Jun Sakuma)
  - Metamathematics for Systems Design Project (PI: Ichiro Hasuo)
  - …

This Talk

- Tutorial of program ***verification*** and ***synthesis*** based on ***constraint solving*** and ***machine learning***

Background

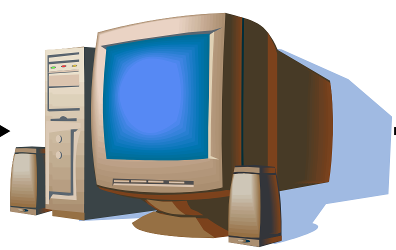
- Our society heavily relies on computer systems
- Failure or malfunction of safety-critical systems would lead to human, social, economic, and environmental damage
  - 1985-1987 – Therac-25 medical accelerator delivered lethal radiation doses to patients
  - June 4, 1996 – Ariane 5 Flight 501 exploded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

Program Verification

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

Program & Spec.

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



Program Verifier

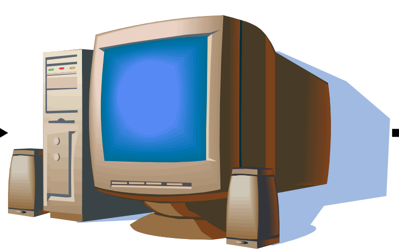
A proof of Q
or
A counterexample to Q
(e.g., a program input leading to a spec. violation)
or
Unknown

Program Synthesis

- Input an **incomplete program** and its **specification ϕ**, and output an **executable program P** that satisfies ϕ
  - ϕ specifies extensional (what P computes) and/or intentional (how P computes) behaviors of P
  - ϕ is represented as a logical formula, input/output examples (e.g., MS Excel FlashFill), a natural language sentence, …

Incomplete Program & Specification

```
let rec mc x = ?
let n = randi() in
if n ≤ 101 then
  assert (mc n = 91)
```



Program Synthesizer

A program satisfying the spec.

```
let rec mc x =
  if x > 100 then
    x - 10
  else
    mc (mc (x + 11))
```

or

A proof of unrealizability

or

Unknown

Enabling Technologies

- ***Program logics*** for mechanizing verification & synthesis
  - Hoare logic for proving Hoare triples $\{P\}c\{Q\}$ meaning that:

    For any initial state σ that satisfies the precondition P, if the execution of the program c under σ terminates, the postcondition Q is satisfied by the resulting state
  - Separation logic
  - Dependent refinement type system
  - Graded modal type system
- ***Constraint solvers*** for automating verification & synthesis
  - SAT solvers: satisfiability checker for propositional formulas
  - SMT solvers: satisfiability checker for predicate formulas over first-order theories on ***integers, reals, lists, arrays***, …

What about *functions* (that represent *inductive invariants, ranking functions, recurrent sets, Skolem functions, …*)?

This Talk

- Tutorial of program ***verification*** and ***synthesis*** based on ***constraint solving*** and ***machine learning*** over ***functions***
- **First part**: How to reduce program ***verification*** and ***synthesis*** to ***constraint solving***
- **Second part**: How to solve constraints via integrated ***deductive*** and ***inductive*** reasoning
  - ***Deductive*** reasoning by ***theorem proving*** (e.g., SAT, SMT)
  - ***Inductive*** reasoning by ***machine learning*** (e.g., decision tree learning, reinforcement learning)

This Talk

- Tutorial of program ***verification*** and ***synthesis*** based on ***constraint solving*** and ***machine learning*** over ***functions***
- **First part**: How to reduce program ***verification*** and ***synthesis*** to ***constraint solving***
- **Second part**: How to solve constraints via integrated ***deductive*** and ***inductive*** reasoning
  - ***Deductive*** reasoning by ***theorem proving*** (e.g., SAT, SMT)
  - ***Inductive*** reasoning by ***machine learning*** (e.g., decision tree learning, reinforcement learning)

Program Verification via Constraint Solving

Target Program $\boldsymbol{P}$ & Specification $\boldsymbol{\psi}$

Constraint Generation

Constraints $\boldsymbol{\mathcal{C}}$ on ***Function Variables***

Verification Intermediary Independent of Particular Target and Method ☺

Constraint Solving

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

Program Verification via **Constrained Horn Clauses (CHCs)**

Target Program $\boldsymbol{P}$ & Specification $\boldsymbol{\psi}$

Constraint Generation

- JayHorn for Java [Kahsai+ ’16]
- SeaHorn for C [Gurfinkel+ ’15]
- RCaml for OCaml [Unno+ ’09]

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

Constraint Solving

- SPACER [Komuravelli+ ’14]
- Hoice [Champion+ ’18]
- Eldarica [Hojjat+ ’18]

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

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

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

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

or $$\bot \Longleftarrow (X_1(\widetilde{t_1}) \wedge \cdots \wedge X_m(\widetilde{t_m}) \wedge \phi)$$

where $X_0, X_1 \ldots, X_m$ are predicate variables,
$\widetilde{t_0}, \ldots, \widetilde{t_m}$ are sequences of terms of a 1st-order theory T,
ϕ is a formula of T without predicate variables.

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

Prog. & Spec. → CHCs → SAT or UNSAT

Constraint Generation　Constraint Solving

Example Program and *Partial Correctness* Specification:

Pre-condition

$\{x = x_0\}$

$y = 0;$

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

$\quad y \leftarrow y + 1;$

$\quad x \leftarrow x - 1$

done

$\{y = x_0\}$

Post-condition

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

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

Prog. & Spec. → CHCs → SAT or UNSAT

Constraint Generation | Constraint Solving

Input:

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

Output $\mathcal{C}$:

represents a ***loop invariant***

① $I(x_0, x, y) \Longleftarrow x = x_0 \wedge y = 0,$

② $I(x_0, x - 1, y + 1)$
$\Longleftarrow I(x_0, x, y) \wedge x \neq 0,$

③ $y = x_0 \Longleftarrow I(x_0, x, y) \wedge x = 0$

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

Program Verification via **Constrained Horn Clauses (CHCs)**

Target Program $\boldsymbol{P}$ & Specification $\boldsymbol{\psi}$

Limited to *Linear-Time* Safety Verification ☹

Constraint Generation

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

Constraint Solving

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

Program Verification via Predicate Constraint Satisfaction [Satake+ '20]

Target Program P & Specification ψ

Applicable to *(Finitely-) Branching-Time* Safety Verification ☺

Constraint Generation

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

Constraint Solving

PCSat [Satake+ '20, Unno+ '20]

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

Linear-Time vs. Branching-Time Verification of Non-det. Programs

- The target program P may exhibit non-determinism caused by user input, network comm., scheduling, …
- ***Linear-time*** verification concerns properties of the ***execution traces*** of P
- ***Branching-time*** verification concerns properties of the ***computation tree*** of P
    - Subsumes **linear-time** verification
    - Example: ***Non-termination verification*** of deciding whether ***there is*** an infinite execution of P (cf. ***termination verification*** decides whether ***all*** execution of P is finite)

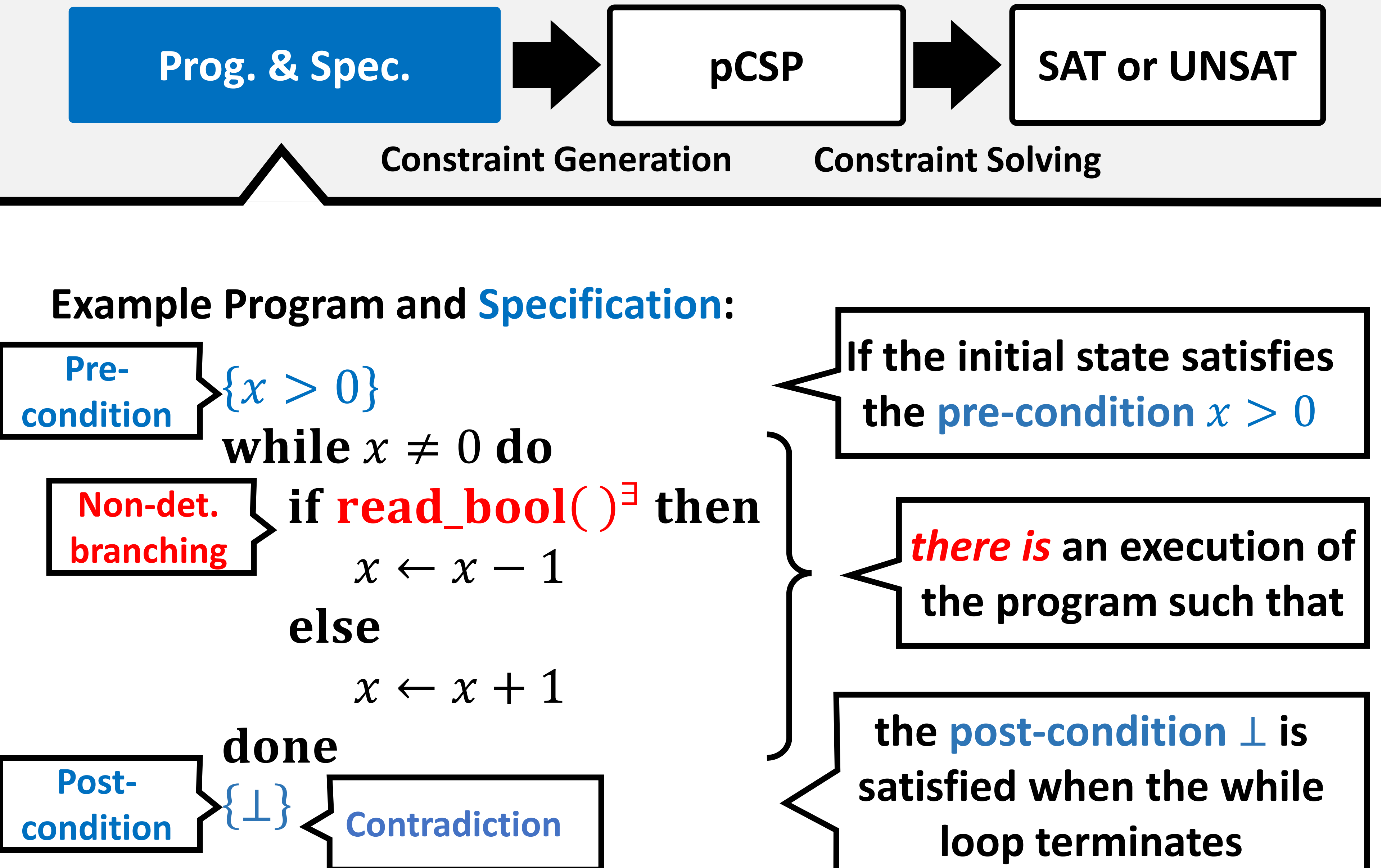
pCSP: Predicate Constraint Satisfaction Problem [Satake+ '20]

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

$$\left(X_1(\widetilde{t_1}) \lor \cdots \lor X_\ell(\widetilde{t_\ell})\right) \Longleftarrow \left(X_{\ell+1}(\widetilde{t_{\ell+1}}) \land \cdots \land X_m(\widetilde{t_m}) \land \phi\right)$$

  where $X_1, \ldots, X_m$ are predicate variables,
  $\widetilde{t_1}, \ldots, \widetilde{t_m}$ are sequences of terms of a 1st-order theory T,
  ϕ is a formula of T without predicate variables.

- $\mathcal{C}$ is ***satisfiable*** (modulo T) if there is an interpretation ρ of predicate variables such that $\rho \vDash \bigwedge \mathcal{C}$
- $\mathcal{C}$ is ***CHCs*** if $\ell \leq 1$ for all clause in $\mathcal{C}$



Prog. & Spec.
pCSP
SAT or UNSAT
Constraint Generation
Constraint Solving
Example Program and Specification:
Pre-condition
$\{x > 0\}$
while $x \neq 0$ do
Non-det. branching
if $\text{read\_bool}()^{\exists}$ then
$x \leftarrow x - 1$
else
$x \leftarrow x + 1$
done
Post-condition
$\{\bot\}$
Contradiction
If the initial state satisfies the pre-condition $x > 0$
there is an execution of the program such that
the post-condition $\bot$ is satisfied when the while loop terminates

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

Constraint Generation   Constraint Solving

Example Program and Specification:

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

while $x \neq 0$ do
  if read_bool( )$^{\exists}$ then (Non-det. branching)
    $x \leftarrow x - 1$
  else
    $x \leftarrow x + 1$
done

Post-condition: $\{\bot\}$ (Contradiction)

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

there is an execution of the program such that

the while loop *never* terminates

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

Constraint Generation

Constraint Solving

Input:

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

Output $\mathcal{C}$:

① $I(x) \Longleftarrow x > 0,$

② $I(x-1) \vee I(x+1) \Longleftarrow I(x) \wedge x \neq 0,$

③ $\bot \Longleftarrow I(x) \wedge x = 0$

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

$\mathcal{C}$ is beyond CHCs!

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

Program Verification via *Extended* Predicate Constraint Satisfaction [Unno+ '20]

Target Program ***P*** & Specification ***ψ***

↓ Constraint Generation

Applicable to (*Infinitely-*) *Branching-Time* Safety & *Liveness* Verification ☺

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

↓ Constraint Solving

PCSat [Satake+ '20, Unno+ '20]

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

Safety vs. Liveness Verification

- **Safety** is a class of properties of the form
  "something bad will never happen"
  - Examples (absence of): assertion failure, division-by-zero, array boundary violation, …
- **Liveness** is a class of properties of the form
  "something good will eventually happen"
  - Examples: termination, deadlock freedom, …

pfwCSP: Extension of pCSP with Functional and Well-founded Predicates [Unno+ '20]
(cf. ∀∃CHCs with dwf [Beyene+ '13])

- A finite set $\mathcal{C}$ of pCSP clauses equipped with a map $\mathcal{K}$ from predicate variable X in $\mathcal{C}$ to $\{\star, \lambda, \Downarrow\}$
  - X is ordinary predicate if $\mathcal{K}(X) = \star$
  - X is ***functional*** predicate if $\mathcal{K}(X) = \lambda$
  - X is ***well-founded*** predicate if $\mathcal{K}(X) = \Downarrow$
- $\mathcal{C}$ is ***satisfiable*** (modulo T) if there is an interpretation ρ of predicate variables such that
  - $\rho \vDash \bigwedge \mathcal{C}$
  - $\forall X. \mathcal{K}(X) = \lambda \implies \rho(X)$ characterizes a ***total function***
  - $\forall X. \mathcal{K}(X) = \Downarrow \implies \rho(X)$ represents a ***well-founded relation***

Prog. & Spec. → pfwCSP → SAT or UNSAT

Constraint Generation   Constraint Solving

Example Program and Specification:

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

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

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

$\textbf{done}$

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

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

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

there is an execution of the program such that

the while loop *never* terminates

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

Constraint Generation

Constraint Solving

Input:

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

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

Output $\mathcal{C}$:

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

① $I(x) \Leftarrow x > 0,$

② $(\exists r.\, I(r - x)) \Leftarrow I(x) \land x > 0,$

$\bot \Leftarrow I(x) \land x \le 0$

Prog. & Spec. → pfwCSP → SAT or UNSAT

Constraint Generation   Constraint Solving

Input:

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

Output $\mathcal{C}$:

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

① $I(x) \Leftarrow x > 0,$

② $I(r - x) \Leftarrow S_\lambda(x, r) \land I(x) \land x > 0,$

③ $\bot \Leftarrow I(x) \land x \leq 0$

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

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

Constraint Generation   Constraint Solving

Example Program and *Total Correctness* Specification:

Pre-condition: $[x \neq 0]$

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

Post-condition: $[\top]$ ← Tautology

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

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

Input:

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

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

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

represents a ***loop invariant***

① $I(x) \Leftarrow x = 0,$

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

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

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

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

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

Further Applications of pfwCSP

- Refinement type inference [Unno+ ’09,’13,’18, Nanjo’18, Katsura+ ’20]
- Validity checking of fixpoint logic formulas
- LTL, CTL, CTL*, modal-mu calculus model checking
- Infinite-state infinite-duration game solving
- Bisimulation and bisimilarity verification
- Hyperproperties verification
- Program synthesis
- … (see [Unno+ ’20] and upcoming papers)

Program Synthesis via Constraint Solving

Language $\mathcal{L}$ & Specification ψ

Constraint Generation

Constraints $\mathcal{C}$ on ***Function Variables***

Synthesis Intermediary Independent of Particular Target and Method ☺

Constraint Solving

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

Program Synthesis via **Syntax-Guided Synthesis (SyGuS)**

Language $\mathcal{L}$ & Specification ψ

↓ Constraint Generation

SyGuS Constraints $\mathcal{C}$ on Function Variables

↓ Constraint Solving

- CVC4 [Reynolds+ '15,'19]
- DryadSynth [Huang+ '20]
- PCSat [Satake+ '20, Unno+ '20]

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

SyGuS: Syntax-Guided Synthesis [Alur+ '15]

- Fix a first-order background theory T such as:
  - Linear integer arithmetic (LIA)
  - Strings (for FlashFill benchmarks)
  - Bit-vectors (for Hackers' Delight benchmarks)
- Given
  - Specification: T-formula ϕ over a function variable f
  - Language: context-free grammar G characterizing the set $\mathcal{L}(G)$ of allowed T-terms
- Find a term $t \in \mathcal{L}(G)$ such that $\vDash [t/f]\phi$

Example LIA SyGuS Constraints $\mathcal{C}$:

- Language: G that generates any term of LIA
- Specification: $\phi \equiv \begin{pmatrix} f(x,y) \geq x \land f(x,y) \geq y \land \\ (f(x,y) = x \lor f(x,y) = y) \end{pmatrix}$

$\mathcal{C}$ is satisfied by $f(x,y) \equiv \mathrm{if}\ x > y\ \mathrm{then}\ x\ \mathrm{else}\ y$

$\mathcal{C}$ can be reduced to the **pfwCSP**:

$$r \geq x \land r \geq y \land (r = x \lor r = y) \Leftarrow F_\lambda(x,y,r)$$

In general, SyGuS constraints $\mathcal{C}$ can be converted to a **pfwCSP** using a predicate that characterizes $\mathcal{L}(G)$

This Talk

- Tutorial of program ***verification*** and ***synthesis*** based on ***constraint solving*** and ***machine learning*** over ***functions***
- **First part**: How to reduce program ***verification*** and ***synthesis*** to ***constraint solving***
- **Second part**: How to solve constraints via integrated ***deductive*** and ***inductive*** reasoning
  - ***Deductive*** reasoning by ***theorem proving*** (e.g., SAT, SMT)
  - ***Inductive*** reasoning by ***machine learning*** (e.g., decision tree learning, reinforcement learning)

Program Verification and Synthesis via **Predicate Constraint Satisfaction**

Target Program $\boldsymbol{P}$ & Specification $\boldsymbol{\psi}$

Language $\boldsymbol{\mathcal{L}}$ & Specification $\boldsymbol{\psi}$

↓ Constraint Generation

pfwCSP Constraints $\boldsymbol{\mathcal{C}}$ on ***Predicate Variables***

↓ Constraint Solving

$\boldsymbol{\mathcal{C}}$ is **Sat**, $\boldsymbol{\mathcal{C}}$ is **Unsat**, or **Unknown**

Challenges in Constraint Solving

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

To address these challenges, researchers are ***integrating deductive & inductive reasoning*** techniques within the framework of ***CounterExample Guided Inductive Synthesis (CEGIS)*** *[Solar-Lezama+ '06]*

CounterExample Guided Inductive Synthesis (CEGIS)

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

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$\emptyset$

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

Teacher

Constraints $\mathcal{C}$:

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

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

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

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

Teacher

Constraints $\mathcal{C}$:

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

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

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$\neg I(0)$

Teacher

Constraints $\mathcal{C}$:

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

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

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$\neg I(0)$

$I(1) \Leftarrow 1 > 0$

Teacher

Constraints $\mathcal{C}$:

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

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

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

$\neg I(0)$

$I(1)$

Teacher

Constraints $\mathcal{C}$:

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

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

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

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

Teacher

Constraints $\mathcal{C}$:

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

No. $\{I(x) \mapsto x = 1\}$ is not.

The 2nd clause is violated when $x = 1$

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

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

Teacher

Constraints $\mathcal{C}$:

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

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

Example Run of CEGIS

Learner

Example Instances $\mathcal{E}$:

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

Teacher

Constraints $\mathcal{C}$:

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

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

Is CEGIS just (Online) Supervised Learning of Classification?

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

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

Machine Learning for CEGIS

- Adapt ML models and algorithms to implement **Learner**
  - Piecewise linear classifiers [Sharma+ ’13a, Garg+ ’14, Unno+ ’20]
  - Decision trees [Krishna+ ’15, Garg+ ’16, Champion+ ’18, Ezudheen+ ’18, Zhu+ ’18]
  - Neural networks [Chang+ ’19, Zhao+ ’20, Abate+ ’21]
  - Greedy set covering w/ logic minimization [Padhi+ ’16, Sharma+ ’13b]
  - Metropolis Hastings MCMC sampler [Sharma+ ’14]
  - Probabilistic inference, survey propagation [Satake+ ’20]
  - Ensemble learning [Padhi+ ’20]
- Learning to learn
  - Reinforcement learning of NNs to generate candidates [Si ’18]
  - Reinforcement learning of strategy to adjust classification models used by **Learner** (joint work w/ Tsukada, Sekiyama, Suenaga)

SMT-based Piecewise Linear Classification (aka Template-based Synthesis)

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

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

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

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

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

A Candidate Solution: $\rho \equiv \{I(x) \mapsto x \geq 0\}$

Decision Tree Learning

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

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

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

Predicates: $Q \equiv \left\{ \begin{matrix} x \geq 0, x \leq 0, x \geq 1, \\ x \geq -1, x \leq 1, x \leq -1 \end{matrix} \right\}$

Classifier: $\rho \equiv \{I(x) \mapsto x \geq 0\}$

Template-based Synthesis vs Decision Tree Learning

- Template-based Synthesis (TB)
  - ☹ Fixes the ***shape*** of solution (updated upon failure)
  - ☺ Flexibly find necessary ***predicates*** via SMT solving
  - ☺ Atoms in $\mathcal{E}$ are consistently ***labeled*** using $\mathcal{E}$ as an SMT formula
- Decision Tree Learning (DT)
  - ☹ Fixes the ***predicates*** of solution (updated upon failure)
  - ☺ Flexibly adjust the ***shape*** based on information gain
  - ☹ Atoms are consistently ***labeled*** using $\mathcal{E}$ as a SAT formula
- Evaluation on SyGuS-Comp'19 Inv track XC benchmarks
  - TB solved **228** instances (out of **276**) and DT solved **180** instances

Future Research Directions

- Efficient synthesis of ***complex and large functions*** from ***complex and large constraints***
  - (Co)Inductive functions
  - Functions over (linked, (co)algebraic, array) data structures
  - Improve labeling, sampling and filtering of examples, and generation and ranking of predicates
- Convergence theory of CEGIS
- More applications

Summary

- Various program ***verification*** and ***synthesis*** problems can be reduced to ***constraint solving*** problems
  - The separation of constraint generation and solving facilitate tool development
- CEGIS-based constraint solving integrates ***deductive*** and ***inductive*** reasoning to address challenges
  - ***Deductive*** reasoning by ***theorem proving*** (e.g., SAT, SMT)
  - ***Inductive*** reasoning by ***machine learning*** (e.g., decision tree learning, reinforcement learning)