

# Probabilistic Inference for Predicate Constraint Satisfaction

Hiroshi Unno (University of Tsukuba)

Joint Work with Yuki Satake and Hinata Yanagi

# Program Verification via Predicate Constraint Satisfaction

Target Program  $P$  & Specification  $\psi$

Constraint  
Generation

Constraints  $C$  on Predicate Variables

Constraint  
Solving

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

# Previous Work: Program Verification via Constrained Horn Clauses (CHCs) [Björner+ '15]

Target Program  $P$  & Specification  $\psi$

Limited to *Linear-Time*  
Safety Verification ☹️

Constraint  
Generation

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

CHCs Constraints  $C$  on Predicate Variables

Verification Intermediary  
Independent of Particular  
Target and Method 😊

Constraint  
Solving

SPACER [Komuravelli+ '14]  
Hoice [Champion+ '18]  
Eldarica [Hojjat+ '18]

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

# This Work: Program Verification via **Predicate Constraint Satisfaction Problem (pCSP)**

Target Program  $P$  & Specification  $\psi$

Support *Branching-Time*  
Safety Verification 😊

Constraint  
Generation

New method for Looping  
& Recursive Programs

**pCSP** Constraints  $C$  on Predicate Variables

Verification Intermediary  
Independent of Particular  
Target and Method 😊

Constraint  
Solving

New method based on  
Probabilistic Inference

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

# This Work: Program Verification via **Predicate Constraint Satisfaction Problem (pCSP)**

Target Program  $P$  & Specification  $\psi$

Constraint  
Generation

**pCSP** Constraints  $c$  on Predicate Variables

Verification Intermediary  
Independent of Particular  
Target and Method 😊

Constraint  
Solving

$c$  is **Sat** ( $P$  satisfies  $\psi$ ),  
 $c$  is **Unsat** ( $P$  violates  $\psi$ ),  
or **Unknown**

# Predicate Constraint Satisfaction Problem (pCSP)

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

$$X_1(\tilde{t}_1) \vee \cdots \vee X_\ell(\tilde{t}_\ell) \vee \phi \vee \\ \neg X_{\ell+1}(\tilde{t}_{\ell+1}) \vee \cdots \vee \neg X_m(\tilde{t}_m)$$

where  $X_1, \dots, X_m$  are predicate variables,  
 $\tilde{t}_1, \dots, \tilde{t}_m$  are sequences of terms,  
 $\phi$  is a first-order formula w/o predicate variables.

# Predicate Constraint Satisfaction Problem (pCSP)

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

$$X_1(\tilde{t}_1) \vee \cdots \vee X_\ell(\tilde{t}_\ell) \Leftarrow$$

$$X_{\ell+1}(\tilde{t}_{\ell+1}) \wedge \cdots \wedge X_m(\tilde{t}_m) \wedge \neg\phi$$

where  $X_1, \dots, X_m$  are predicate variables,

$\tilde{t}_1, \dots, \tilde{t}_m$  are sequences of terms,

$\phi$  is a first-order formula w/o predicate variables.

- $\mathcal{C}$  is *satisfiable* (modulo first-order theories) if there is an interpretation  $\rho$  of predicate variables such that  $\rho \models \bigwedge \mathcal{C}$
- $\mathcal{C}$  is *called CHCs* [Bjørner+ '15] if  $\ell \leq 1$  for each clause in  $\mathcal{C}$

# This Work: Program Verification via **Predicate Constraint Satisfaction Problem (pCSP)**

Target Program  $P$  & Specification  $\psi$

Support *Branching-Time*  
Safety Verification 😊

Constraint  
Generation

**pCSP** Constraints  $c$  on Predicate Variables

Constraint  
Solving

$c$  is **Sat** ( $P$  satisfies  $\psi$ ),  
 $c$  is **Unsat** ( $P$  violates  $\psi$ ),  
or **Unknown**



# Branching-Time Safety Verification of Finitely-Branching Programs

- **Safety** is a class of properties of the form *“something bad will never happen”*
- **Branching-time** verification concerns properties of the *computation tree* of the given program that may exhibit non-deterministic behavior (cf. **linear-time** verification concerns properties of *execution traces*)
- Subsumes *non-termination verification* of deciding whether *there is* a non-terminating execution

# This Work: Program Verification via **Predicate Constraint Satisfaction Problem (pCSP)**

Target Program  $P$  & Specification  $\psi$

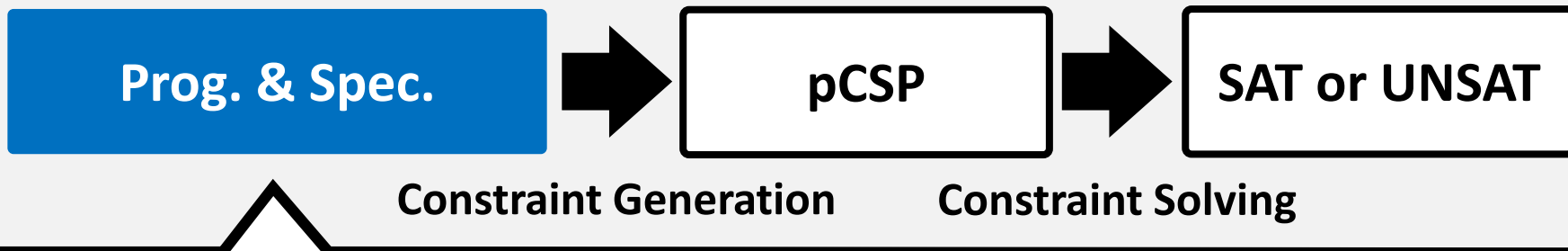
Constraint  
Generation

New method for Looping  
& Recursive Programs

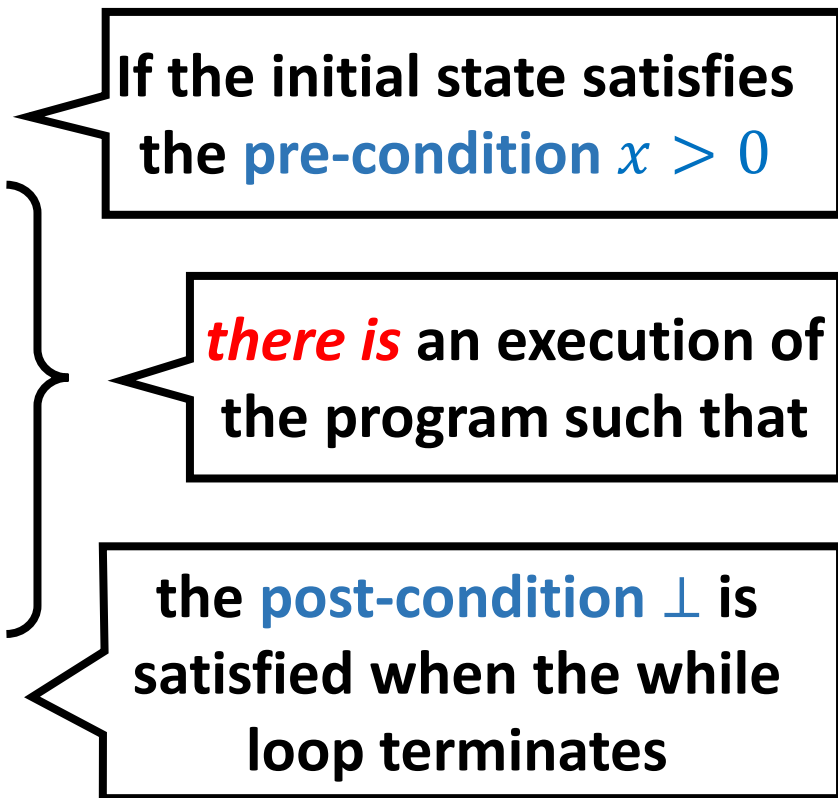
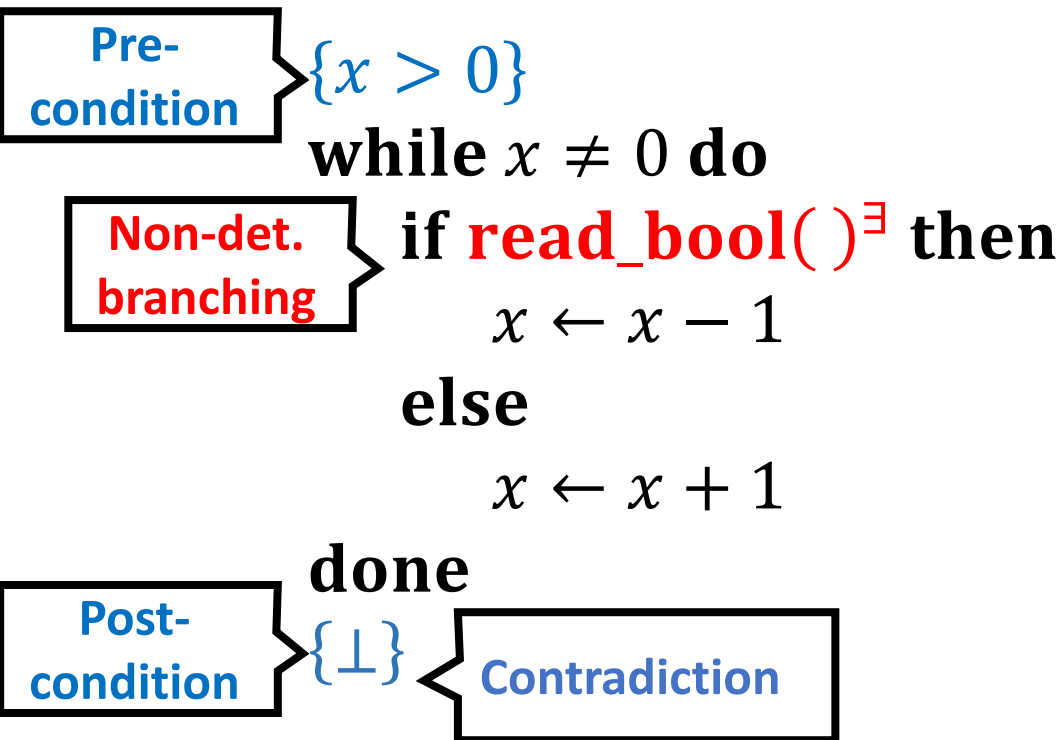
**pCSP** Constraints  $C$  on Predicate Variables

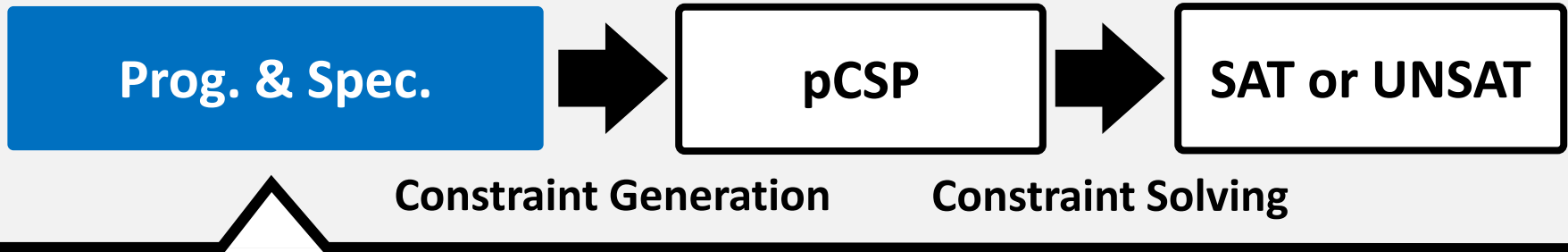
Constraint  
Solving

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



### Example Program and Specification:





Example Program and **Specification**:

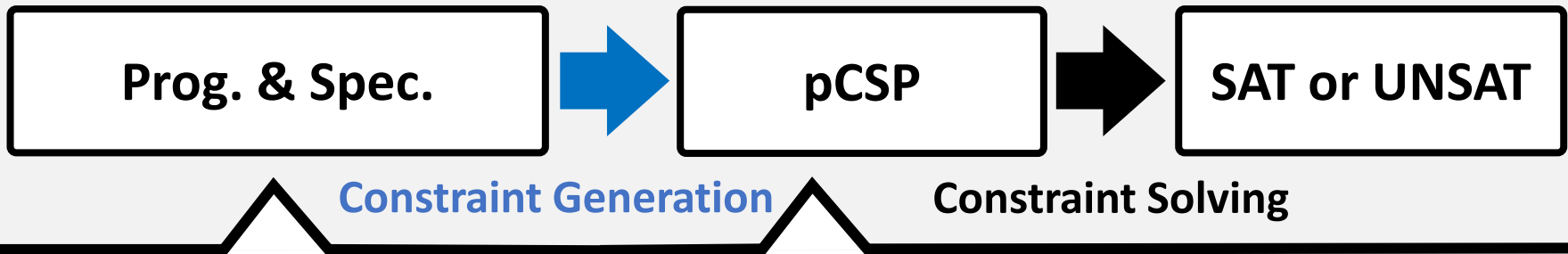
```

Pre-condition { $x > 0$ }
while  $x \neq 0$  do
  Non-det. branching if read_bool()∃ then
     $x \leftarrow x - 1$ 
  else
     $x \leftarrow x + 1$ 
done
Post-condition { $\perp$ } 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**



**Input:**

```

{x > 0}
while x ≠ 0 do
  if read_bool()³ then
    x ← x - 1
  else
    x ← x + 1
done
{⊥}
  
```

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

represents a *loop invariant* preserved by *some* execution

- ①  $I(x) \Leftarrow x > 0,$
- ②  $I(x - 1) \vee I(x + 1)$
- ③  $\perp \Leftarrow I(x) \wedge x = 0$

$\mathcal{C}$  is beyond CHCs!

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

# This Work: Program Verification via **Predicate Constraint Satisfaction Problem (pCSP)**

Target Program  $P$  & Specification  $\psi$

Constraint  
Generation

**pCSP** Constraints  $c$  on Predicate Variables

Constraint  
Solving

**New method based on  
Probabilistic Inference**

$c$  is **Sat** ( $P$  satisfies  $\psi$ ),  
 $c$  is **Unsat** ( $P$  violates  $\psi$ ),  
or **Unknown**

# Challenges in pCSP Solving

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

We address these challenges by a novel combination of *probabilistic inference* with *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**
    - Enumerate candidate solutions  $\rho_1, \dots, \rho_n$  that satisfy  $\mathcal{E}$
  - **Validation Phase**
    - Check if there is a candidate  $\rho_i$  that also satisfies  $\mathcal{C}$ 
      - If yes, return  $\rho_i$  as a solution of  $\mathcal{C}$
      - If no, repeat the procedure with new example instances witnessing non-satisfaction of  $\mathcal{C}$  by  $\rho_1, \dots, \rho_n$  added



# Example Run of CEGIS

## Synthesizer

Example Instances  $\mathcal{E}$ :

$\emptyset$

Starting from  
the empty set

## Validator

pCSP Constraints  $\mathcal{C}$ :

- $I(x) \iff x > 0$
- $I(x - 1) \vee I(x + 1) \iff I(x) \wedge x \neq 0$
- $\perp \iff I(x) \wedge x = 0$

Is one of the following candidates genuine?  
 $\{I(x) \mapsto \top\}, \dots$

# Example Run of CEGIS

## Synthesizer

Example Instances  $\mathcal{E}$ :

$$\perp \Leftarrow I(0) \wedge 0 = 0$$

## Validator

pCSP Constraints  $\mathcal{C}$ :

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

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

# Example Run of CEGIS

## Synthesizer

Example Instances  $\mathcal{E}$ :

$$\neg I(0)$$

## Validator

pCSP Constraints  $\mathcal{C}$ :

- $I(x) \iff x > 0$
- $I(x - 1) \vee I(x + 1) \iff I(x) \wedge x \neq 0$
- $\perp \iff I(x) \wedge x = 0$

Is one of the following candidates genuine?  
 $\{I(x) \mapsto x < 0\}, \dots$

# Example Run of CEGIS

## Synthesizer

Example Instances  $\mathcal{E}$ :

$$\neg I(0)$$

$$I(1) \iff 1 > 0$$

## Validator

pCSP Constraints  $\mathcal{C}$ :

- $I(x) \iff x > 0$
- $I(x - 1) \vee I(x + 1) \iff I(x) \wedge x \neq 0$
- $\perp \iff I(x) \wedge x = 0$

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

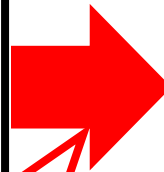
# Example Run of CEGIS

## Synthesizer

Example Instances  $\mathcal{E}$ :

$$\neg I(0)$$

$$I(1)$$



## Validator

pCSP Constraints  $\mathcal{C}$ :

- $I(x) \iff x > 0$
- $I(x - 1) \vee I(x + 1) \iff I(x) \wedge x \neq 0$
- $\perp \iff I(x) \wedge x = 0$

Is one of the following candidates genuine?

$$\{I(x) \mapsto x \geq 1\}, \dots$$

# Example Run of CEGIS

## Synthesizer

Example Instances  $\mathcal{E}$ :

$$\neg I(0)$$

$$I(1)$$

## Validator

pCSP Constraints  $\mathcal{C}$ :

- $I(x) \iff x > 0$
- $I(x - 1) \vee I(x + 1) \iff I(x) \wedge x \neq 0$
- $\perp \iff I(x) \wedge x = 0$

Yes.  $\{I(x) \mapsto x \geq 1\}$  is a solution of  $\mathcal{C}$ !

# Enumeration of Candidates $\rho_1, \dots, \rho_n$

- Challenges:

1. The number  $n$  of candidates must be **bounded**
2. Candidates must satisfy  $\mathcal{E}$  and should be **simpler** (per Occam's razor) and **essentially different** each other for **more chance to satisfy  $\mathcal{C}$**

- Our Solutions:

1. Compute **predicate abstraction** [Graf+ '97]  $\alpha(\mathcal{E})$  of  $\mathcal{E}$  using a finite set  $\mathcal{Q}$  of predicates to reduce the search space of candidates to the **finite set** of  $\mathcal{Q}$ -expressible solutions
2. Enumerate “**promising**” satisfying Boolean assignments of  $\alpha(\mathcal{E})$  via **survey inspired decimation (SID)** [Braunstein+ '05]

# Example: Predicate Abstraction

Example Instances  $\mathcal{E}$ :

- $\neg I(0)$
- $I(1)$

Predicates Set  $\mathcal{Q}$ :

$$\left\{ \begin{array}{l} \lambda x. \perp, \lambda x. \top, \\ \lambda x. x \geq 0, \lambda x. -x \geq 0, \\ \lambda x. x \geq 1, \lambda x. -x \geq -1 \end{array} \right\}$$



# Example: Predicate Abstraction

Example Instances  $\mathcal{E}$ :

- $\neg I(0)$
- $I(1)$

Predicates Set  $\mathcal{Q}$ :

$$\left\{ \begin{array}{l} \lambda x. \perp, \lambda x. \top, \\ \lambda x. x \geq 0, \lambda x. 0 \geq x, \\ \lambda x. x \geq 1, \lambda x. 1 \geq x \end{array} \right\}$$

Solution Template with Boolean parameters  $b_{\perp}, b_{\geq 1}, b_{0 \geq}, \dots$ :

$$\{I(x) \mapsto (b_{\perp} \Rightarrow \perp) \wedge (b_{\geq 1} \Rightarrow x \geq 1) \wedge (b_{0 \geq} \Rightarrow 0 \geq x) \wedge \dots\}$$

Predicate Abstraction  $\alpha(\mathcal{E}) \equiv (b_{\perp} \vee b_{\geq 1}) \wedge \neg b_{\perp} \wedge \neg b_{0 \geq}$

Satisfying Assignments for  $\alpha(\mathcal{E})$ : don't care

$\{b_{\perp} \mapsto \perp, b_{0 \geq} \mapsto \perp, b_{\geq 1} \mapsto \top, b_{\top} \mapsto *, b_{\geq 0} \mapsto *, b_{1 \geq} \mapsto *\}$ , indicating that  $\lambda x. \perp$  and  $\lambda x. 0 \geq x$  **must not be used** and  $\lambda x. x \geq 1$  **must be used**

$\therefore \mathcal{Q}$ -expressible sols. are  $\{I(x) \mapsto x \geq 1\}$  and  $\{I(x) \mapsto x \geq 1 \wedge 1 \geq x\}$

# SID-based Enumeration of “*Promising*” Satisfying Assignments for $\alpha(\mathcal{E})$

- To obtain a *simpler solution* (e.g.,  $\{I(x) \mapsto x \geq 1\}$ ), detect and assign  $\perp$  to the don't-care variables
- To *reduce similar solutions* while *preserving different ones* belonging to *different solution clusters*, use SID to iteratively assign a value  $v$  to a variable of  $\alpha(\mathcal{E})$  with the *highest bias* toward  $v$  (i.e., “sufficiently” determined)
  - The *biases* of each variable are computed via *probabilistic inference (survey propagation)* in a *graphical model (factor graph)* obtained from  $\alpha(\mathcal{E})$  (see the paper for details)

# Evaluation

- Implemented the presented method as a pCSP solver **PCSat** using **Z3** as the backend SMT solver
- Tested **PCSat** on the benchmark sets from
  - SyGuS-Comp 2017 and 2018 (Invariant Synthesis Track)
  - CHC-COMP 2019 (LIA-nonlin Track)
  - New pCSP benchmarks of branching-time safety verification that *go beyond the scope of existing SyGuS and CHC solvers*

# Results on 127 Benchmarks from SyGuS-Comp 2018

|               | #SAT | #UNSAT |
|---------------|------|--------|
| SID(#cand=1)  | 91   | 8      |
| SID(#cand=2)  | 92   | 8      |
| SID(#cand=4)  | 96   | 8      |
| SID(#cand=8)  | 100  | 8      |
| SID(#cand=16) | 96   | 8      |
| SID(#cand=32) | 95   | 7      |
| SID(#cand=64) | 94   | 6      |
| SAT(#cand=1)  | 94   | 8      |
| SAT(#cand=2)  | 92   | 8      |
| SAT(#cand=4)  | 91   | 8      |
| SAT(#cand=8)  | 89   | 7      |

# Summary: Program Verification via **Predicate Constraint Satisfaction Problem (pCSP)**

Target Program  $P$  & Specification  $\psi$

Support *Branching-Time*  
Safety Verification 😊

Constraint  
Generation

New method for Looping  
& Recursive Programs

**pCSP** Constraints  $c$  on Predicate Variables

Verification Intermediary  
Independent of Particular  
Target and Method 😊

Constraint  
Solving

New method based on  
CEGIS, Pred. Abstraction,  
Survey Inspired Decimation

$c$  is **Sat** ( $P$  satisfies  $\psi$ ),  
 $c$  is **Unsat** ( $P$  violates  $\psi$ ),  
or **Unknown**

# Ongoing and Future Work

- ✓ Extend pCSP to support branching-time *liveness* verification of (possibly) *infinitely* branching programs
  - *Liveness: “something good will eventually happen”*
- Develop constraint generation tools for C, Java, OCaml,  $\mu$ CLP
- Extend **PCSat** to support more theories (Arrays, ADTs, heaps, ...)
- Apply *other constraint satisfaction methods* to enumerate sols.
- Apply *other probabilistic inference* like *variational inference* and *approximate model counting* by directly *modeling pCSP as factor graphs* representing joint probability distributions over random predicate variables