Constraint-based Relational Verification

Hiroshi Unno[1], **Tachio Terauchi[2]**, Eric Koskinen[3]

[1] University of Tsukuba [2]Waseda University [3]Stevens Institute of Technology

Appeared in CAV 2021

Automated Verification of Relational Properties

1. ***k-safety*** $\subset$ ***Hypersafety***
   - ***Termination-Insensitive Non-Interference (TI-NI)*** $\in$ ***2-safety***
     - A well-studied security property formalizing the absence of information leakage [Goguen+ '82]
2. ***Co-termination*** $\in$ ***Hyperliveness***
   - Formalize that multiple programs agree on termination
   - ***Termination-Sensitive NI (TS-NI)*** = ***TI-NI*** $\wedge$ ***Co-termination***
3. ***(TI-/TS-) Generalized NI (GNI)*** $\in$ $\forall\exists$ ***hyperproperties***
   - A generalization of ***NI*** for non-deterministic programs

Main Contributions

- ***Sound*** and ***complete*** reduction to ***constraint solving***
  - Key observation: relational verification amounts to synthesis of ***relational invariants*** and ***fair & semantic schedulers***
- New expressive class of predicate constraints **pfwCSP**
  - Extend the class of CHCs with ***non-Horn clauses*** and ***functional*** / ***well-founded*** predicate variables
- ***Stratified CEGIS*** for **pfwCSP *solving***
  - Combine CEGIS with ***stratified families of templates*** for achieving ***completeness*** and ***efficiency***

Main Contributions

- ***Sound*** and ***complete*** reduction to ***constraint solving***
  - Key observation: relational verification amounts to synthesis of ***relational invariants*** and ***fair & semantic schedulers***
- New expressive class of predicate constraints **pfwCSP**
  - Extend the class of CHCs with ***non-Horn clauses*** and ***functional*** / ***well-founded*** predicate variables
- ***Stratified CEGIS*** for **pfwCSP *solving***
  - Combine CEGIS with ***stratified families of templates*** for achieving ***completeness*** and ***efficiency***

Relational Verification via Synthesis of Relational Invariants and Fair & Semantic Schedulers

1. ***k-safety*** $\subset$ ***Hypersafety***
   - ***Termination-Insensitive Non-Interference (TI-NI)*** $\in$ ***2-safety***
     - A well-studied security property formalizing the absence of information leakage [Goguen+ '82]
2. ***Co-termination*** $\in$ ***Hyperliveness***
   - Formalize that multiple programs agree on termination
   - ***Termination-Sensitive NI (TS-NI)*** = ***TI-NI*** $\wedge$ ***Co-termination***
3. ***(TI-/TS-) Generalized NI (GNI)*** $\in$ $\forall\exists$ ***hyperproperties***
   - A generalization of ***NI*** for non-deterministic programs

Relational Verification via Synthesis of Relational Invariants and Fair & Semantic Schedulers

1. ***k-safety*** $\subset$ ***Hypersafety***
   - ***Termination-Insensitive Non-Interference (TI-NI)*** $\in$ ***2-safety***
     - A well-studied security property formalizing the absence of information leakage [Goguen+ '82]
2. ***Co-termination*** $\in$ ***Hyperliveness***
   - Formalize that multiple programs agree on termination
   - ***Termination-Sensitive NI (TS-NI)*** = ***TI-NI*** $\wedge$ ***Co-termination***
3. ***(TI-/TS-) Generalized NI (GNI)*** $\in \forall\exists$ ***hyperproperties***
   - A generalization of ***NI*** for non-deterministic programs

Example: Termination-Insensitive Non-Interference (TI-NI) ∈ 2-safety

- doubleSquare(h, x) [Shemer+ '19] computes $2 \cdot x^2$ in two different ways depending on the ***high security*** input h

```
doubleSquare(bool h,int x) {
  int z, y=0;
  if(h) { z=2*x }
  else { z=x }
  while(z>0) { z--; y=y+x }
  if(!h) { y=2*y }
  return y;
}
```

Can an attacker infer the value of h by observing the ***low security*** input x and the return value y?

No! Formally, **TI-NI** holds:

$\forall h_1, x_1, y_1, h_2, x_2, y_2.$
$\texttt{doubleSquare}(h_1, x_1) \rightsquigarrow y_1 \wedge$
$\texttt{doubleSquare}(h_2, x_2) \rightsquigarrow y_2 \wedge$
$x_1 = x_2 \Rightarrow y_1 = y_2$

Existing Approach to TI-NI (Self Composition)

- Find a ***safe rel. invariant*** of the parallel executions of 2 copies of the program under the ***lock-step scheduler***

```
doubleSquare(bool h,int x) {
  int z, y=0;
ℓ1: if(h) {z=2*x} else {z=x}
ℓ2: while(z>0) { z--; y=y+x }
ℓ3: if(!h) { y=2*y }
  return y;
}
```

Synchronize at ℓ_3

A run with ***lock-step scheduler***:

$(h_1, x_1, y_1, z_1, h_2, x_2, y_2, z_2)$

$\mapsto (\top, 2,0, \_, \bot, 2,0, \_)$ $x_1 = x_2$

$\rightarrow^{\ell_1}_{1,2} (\top, 2,0,4, \bot, 2,0,2)$ $y_1 = y_2$

$\rightarrow^{\ell_2}_{1,2} (\top, 2,2,3, \bot, 2,2,1)$ $y_1 = y_2$

$\rightarrow^{\ell_2}_{1,2} (\top, 2,4,2, \bot, 2,4,0)$ $y_1 = y_2$

$\rightarrow^{\ell_2}_{1} (\top, 2,6,1, \bot, 2,4,0)$ $y_1 \neq y_2$

$\rightarrow^{\ell_2}_{1} (\top, 2,8,0, \bot, 2,4,0)$ $y_1 \neq y_2$

$\rightarrow^{\ell_3}_{1,2} (\top, 2,8,0, \bot, 2,8,0)$ $y_1 = y_2$

Existing Approach to TI-NI (Self Composition)

- Find a ***safe rel. invariant*** of the parallel executions of 2 copies of the program under the ***lock-step scheduler***

```
doubleSquare(bool h,int x) {
  int z, y=0;
ℓ1: if(h) {z=2*x} else {z=x}
ℓ2: while(z>0) { z--; y=y+x }
ℓ3: if(!h) { y=2*y }
  return y;
}
```

A run with ***lock-step scheduler***:

$(h_1, x_1, y_1, z_1, h_2, x_2, y_2, z_2)$

$\mapsto (\top, 2,0, \_, \bot, 2,0, \_)$ $x_1 = x_2$

$\rightarrow^{\ell_1}_{1,2} (\top, 2,0,4, \bot, 2,0,2)$ $y_1 = y_2$

$\rightarrow^{\ell_2}_{1,2} (\top, 2,2,3, \bot, 2,2,1)$ $y_1 = y_2$

$\rightarrow^{\ell_2}_{1,2} (\top, 2,4,2, \bot, 2,4,0)$ $y_1 = y_2$

$\rightarrow^{\ell_2}_{1} (\top, 2,6,1, \bot, 2,4,0)$ $y_1 \neq y_2$

$\rightarrow^{\ell_2}_{1} (\top, 2,8,0, \bot, 2,4,0)$ $y_1 \neq y_2$

$\rightarrow^{\ell_3}_{1,2} (\top, 2,8,0, \bot, 2,8,0)$ $y_1 = y_2$

Any "simple" invariant to prove this?

↳ No! *Safe rel. invariant* for this TI-NI is ***not expressible*** in LIA [Shemer+'19]

Our Approach to TI-NI

Can choose which program to execute depending on the states

- Find a ***semantic scheduler*** Sch and a ***safe rel. invariant*** of the two program copies executed under Sch

```
doubleSquare(bool h,int x) {
  int z, y=0;
ℓ1: if(h) {z=2*x} else {z=x}
ℓ2: while(z>0) { z--; y=y+x }
ℓ3: if(!h) { y=2*y }
  return y;
}
```

Found scheduler Sch dictates:

both copies move if $z_1 = 2 \cdot z_2$,

1st copy moves if $z_1 = 2 \cdot z_2 + 1$

An example run under Sch:

$(h_1, x_1, y_1, z_1, h_2, x_2, y_2, z_2)$

$\mapsto (\top, 2, 0, \_, \bot, 2, 0, \_)$

$\rightarrow^{\ell_1} (\top, 2, 0, 4, \bot, 2, 0, 2)$

$\rightarrow^{\ell_2}_{1,2} (\top, 2, 2, 3, \bot, 2, 2, 1)$

$\rightarrow^{\ell_2}_{1} (\top, 2, 4, 2, \bot, 2, 2, 1)$

$\rightarrow^{\ell_2}_{1,2} (\top, 2, 6, 1, \bot, 2, 4, 0)$

$\rightarrow^{\ell_2}_{1} (\top, 2, 8, 0, \bot, 2, 4, 0)$

$\rightarrow^{\ell_3} (\top, 2, 8, 0, \bot, 2, 8, 0)$

Our Approach to TI-NI

Can choose which program to execute depending on the states

- Find a ***semantic scheduler*** Sch and a ***safe rel. invariant*** of the two program copies executed under Sch

Found invariant expressible in LIA:

$$h_1 \wedge \neg h_2 \wedge x_1 = x_2 \wedge \left(\begin{array}{l} y_1 = 2 \cdot y_2 \wedge z_1 = 2 \cdot z_2 \vee \cdots \wedge \\ y_1 = 2 \cdot y_2 - x_2 \wedge z_1 = 2 \cdot z_2 + 1 \end{array}\right) \vee h_1 \wedge h_2 \wedge \cdots \vee \neg h_1 \wedge h_2 \wedge \cdots \vee \neg h_1 \wedge \neg h_2 \wedge \cdots$$

```
ℓ2: while(z>0) { z--; y=y+x }
ℓ3: if(!h) { y=2*y }
  return y;
}
```

Found scheduler Sch dictates:

both copies move if $z_1 = 2 \cdot z_2$,

1st copy moves if $z_1 = 2 \cdot z_2 + 1$

An example run under Sch:

$(h_1, x_1, y_1, z_1, h_2, x_2, y_2, z_2)$

$\mapsto (\top, 2, 0, \_, \bot, 2, 0, \_)$ $x_1 = x_2$

$\rightarrow^{\ell_1} (\top, 2, 0, 4, \bot, 2, 0, 2)$ $y_1 = 2 \cdot y_2$

$z_1 = 2 \cdot z_2$ $\rightarrow^{\ell_2}_{1,2} (\top, 2, 2, 3, \bot, 2, 2, 1)$ $y_1 = 2 \cdot y_2 - x_2$

$z_1 = 2 \cdot z_2 + 1$ $\rightarrow^{\ell_2}_{1} (\top, 2, 4, 2, \bot, 2, 2, 1)$ $y_1 = 2 \cdot y_2$

$z_1 = 2 \cdot z_2$ $\rightarrow^{\ell_2}_{1,2} (\top, 2, 6, 1, \bot, 2, 4, 0)$ $y_1 = 2 \cdot y_2 - x_2$

$z_1 = 2 \cdot z_2 + 1$ $\rightarrow^{\ell_2}_{1} (\top, 2, 8, 0, \bot, 2, 4, 0)$ $y_1 = 2 \cdot y_2$

$\rightarrow^{\ell_3} (\top, 2, 8, 0, \bot, 2, 8, 0)$ $y_1 = y_2$

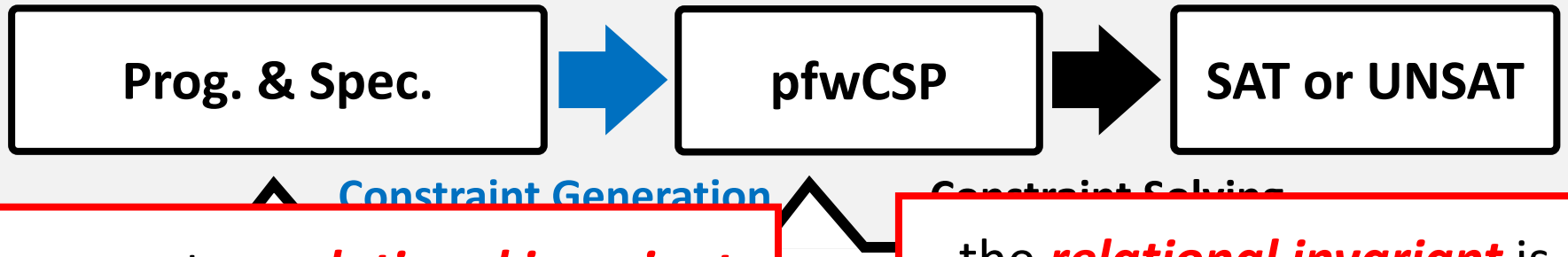
Input:

```
doubleSquare
  (bool h, int x) {
  int z, y=0;
  if(h) { z=2*x }
  else { z=x }
  while(z>0)
    { z--; y=y+x }
  if(!h) { y=2*y }
  return y;
}
```

Output $\mathcal{C}$ (with $h_1 = \top \wedge h_2 = \bot$ fixed):

- $I(V_1, V_2) \Leftarrow x_1 = x_2 \wedge y_1 = y_2 = 0 \wedge z_1 = 2 \cdot x_1 \wedge z_2 = x_2'$
- $I(x_1, y_1', z_1', V_2) \Leftarrow I(V_1, V_2) \wedge Sch_1(V_1, V_2) \wedge \begin{pmatrix} z_1 > 0 \wedge z_1' = z_1 - 1 \wedge y_1' = y_1 + x_1 \vee \\ z_1 \leq 0 \wedge z_1' = z_1 \wedge y_1' = y_1 \end{pmatrix}, \ldots,$
- $y_1 = 2 \cdot y_2 \Leftarrow I(V_1, V_2) \wedge z_1 \leq 0 \wedge z_2 \leq 0,$
- $\left(Sch_1(V_1, V_2) \vee Sch_2(V_1, V_2) \vee Sch_{1,2}(V_1, V_2)\right) \Leftarrow I(V_1, V_2) \wedge (z_1 > 0 \vee z_2 > 0),$
- $z_1 > 0 \Leftarrow I(V_1, V_2) \wedge Sch_1(V_1, V_2) \wedge z_2 > 0,$
- $z_2 > 0 \Leftarrow I(V_1, V_2) \wedge Sch_2(V_1, V_2) \wedge z_1 > 0,$

where $V_1 = x_1, y_1, z_1$ and $V_2 = x_2, y_2, z_2$



represents a ***relational invariant*** preserved by a ***semantic scheduler***

the ***relational invariant*** is inductive and implies **TI-NI**

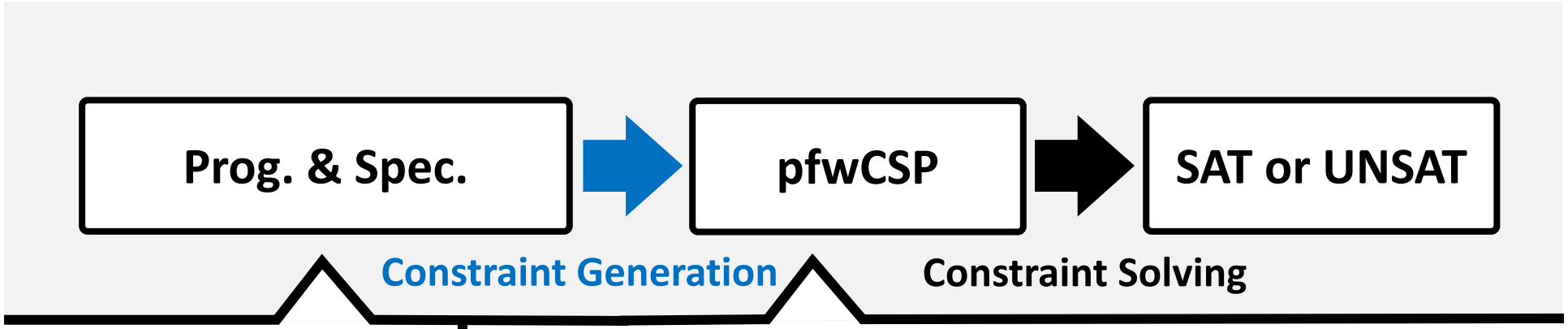
```
doubleSquare
  (bool h, int x) {
  int y=0, z=x;
  while(z>0)
    { z--; y=y+x }
  if(!h) { y=2*y }
  return y;
}
```

represent the ***semantic scheduler***: if Sch_1 holds, 1st copy is scheduled

th h_1

- $I(V_1, V_2) \Leftarrow x_1 = x_2 \wedge y_1 = y_2 = 0 \wedge z_1 = 2 \cdot x_1 \wedge z_2 = x_2'$
- $I(x_1, y_1', z_1', V_2) \Leftarrow I(V_1, V_2) \wedge Sch_1(V_1, V_2) \wedge \begin{pmatrix} z_1 > 0 \wedge z_1' = z_1 - 1 \wedge y_1' = y_1 + x_1 \vee \\ z_1 \leq 0 \wedge z_1' = z_1 \wedge y_1' = y_1 \end{pmatrix}, \ldots,$
- $y_1 = 2 \cdot y_2 \Leftarrow I(V_1, V_2) \wedge z_1 \leq 0 \wedge z_2 \leq 0,$
- $\left(Sch_1(V_1, V_2) \vee Sch_2(V_1, V_2) \vee Sch_{1,2}(V_1, V_2)\right) \Leftarrow I(V_1, V_2) \wedge (z_1 > 0 \vee z_2 > 0),$
- $z_1 > 0 \Leftarrow I(V_1, V_2) \wedge Sch_1(V_1, V_2) \wedge z_2 > 0,$
- $z_2 > 0 \Leftarrow I(V_1, V_2) \wedge Sch_2(V_1, V_2) \wedge z_1 > 0,$

where $V_1 = x_1, y_1, z_1$ and $V_2 = x_2, y_2, z_2$



Input:

```
doubleSquare
  (bool h, int x) {
  int z, y=0;
  if(h) { z=2*x }
  else { z=x }
  while(z>0)
    { z--; y=y+x }
  if(!h) { y=2*y }
```

non-Horn clause that goes beyond **CHCs** !

Output $\mathcal{C}$ (with $h_1 = \top \wedge h_2 = \bot$ fixed):

- $I(V_1, V_2) \Leftarrow x_1$
- $\wedge$
- $I(x_1, y_1', z_1', V_2)$
- $\left(z_1 > 0 \wedge z_1'\right.$
- $\left.z_1 \leq\right.$
- $y_1 = 2 \cdot y_2 \Leftarrow I(V_1, V_2) \wedge z_1 \leq 0 \wedge z_2 \leq 0,$
- $\Big(Sch_1(V_1, V_2) \vee Sch_2(V_1, V_2) \vee Sch_{1,2}(V_1, V_2)\Big)$
  $\Leftarrow I(V_1, V_2) \wedge (z_1 > 0 \vee z_2 > 0),$
  $z_1 > 0 \Leftarrow I(V_1, V_2) \wedge Sch_1(V_1, V_2) \wedge z_2 > 0,$
- $z_2 > 0 \Leftarrow I(V_1, V_2) \wedge Sch_2(V_1, V_2) \wedge z_1 > 0,$

where $V_1 = x_1, y_1, z_1$ and $V_2 = x_2, y_2, z_2$

the scheduler is ***fair***: at least one unfinished program must be scheduled if there is any ***(necessary for the soundness)***

Relational Verification via Synthesis of Relational Invariants and Fair & Semantic Schedulers

1. ***k*-safety ⊂ Hypersafety**
   - **Termination-Insensitive Non-Interference (TI-NI) ∈ 2-safety**
     - A well-studied security property formalizing the absence of information leakage [Goguen+ '82]
2. **Co-termination ∈ Hyperliveness**
   - Formalize that multiple programs agree on termination
   - **Termination-Sensitive NI (TS-NI) = TI-NI ∧ Co-termination**
3. **(TI-/TS-) Generalized NI (GNI) ∈ ∀∃ hyperproperties**
   - A generalization of ***NI*** for non-deterministic programs

Example: Co-Termination ∈ Hyperliveness

```
prog1(int x, int y) { while(x>0) { x=x-y; } }
prog2(int x, int y) { while(x>0) { x=x-2*y; } }
```

Do $\texttt{prog1}(x_1, y_1)$ and $\texttt{prog2}(x_2, y_2)$ agree on termination under the precondition $x_1 = x_2 \wedge y_1 = y_2$?

Yes! Formally, the following holds:

$$\forall x_1, y_1, x_2, y_2.\ (x_1 = x_2 \wedge y_1 = y_2) \Rightarrow \left(\begin{array}{l}\exists z_1.\ \mathbf{prog1}(x_1, y_1) \rightsquigarrow z_1 \\ \Rightarrow \neg(\mathbf{prog2}(x_2, y_2) \rightsquigarrow \bot)\end{array}\right) \wedge \left(\begin{array}{l}\exists z_2.\ \mathbf{prog2}(x_2, y_2) \rightsquigarrow z_2 \\ \Rightarrow \neg(\mathbf{prog1}(x_1, y_1) \rightsquigarrow \bot)\end{array}\right)$$

One ***symmetric*** co-termination problem boils down to two ***asymmetric*** co-termination problems

Our Approach to Asymmetric Co-Termination

- Find a ***fair semantic scheduler*** Sch, a ***relational invariant***, and a ***well-founded relation*** under Sch

```
prog1(int x, int y) {
  while(x>0) { x=x-y; }
}
prog2(int x, int y) {
  while(x>0) { x=x-y; }
}
```

$x_1 > 0 \wedge x_2 > 0$

$x_1 > 0 \wedge x_2 > 0$

$x_1 > 0 \wedge x_2 \leq 0$

$x_1 > 0 \wedge x_2 \leq 0$

An example run under Sch:

(x_1, y_1, x_2, y_2)

$\mapsto (4,1,4,1)$ $x_1 = x_2 \wedge y_1 = y_2$

$\rightarrow_{1,2} (3,1,2,1)$

$\rightarrow_{1,2} (2,1,0,1)$ prog2 terminated

$\rightarrow_{1} (1,1,0,1)$

$\rightarrow_{1} (0,1,0,1)$ prog1 terminated

Found scheduler Sch dictates:

both programs move if $x_1 > 0 \wedge x_2 > 0$,

prog1 moves if $x_1 > 0 \wedge x_2 \leq 0$

Our Approach to Asymmetric Co-Termination

- Find a ***fair semantic scheduler*** Sch, a ***relational invariant***, and a ***well-founded relation*** under Sch

```
prog1(int x, int y) {
  while(x>0) { x=x-y; }
```

Found relational invariant implies:

$x_1 > 0 \wedge x_2 \leq 0 \Rightarrow y_1 \geq 1$

Found well-founded relation witnesses:

if $x_1 > 0 \wedge x_2 \leq 0$, then prog1 ***decreases x_1***

but x_1 is lower bounded by 0

Found scheduler Sch dictates:

both programs move if $x_1 > 0 \wedge x_2 > 0$,

prog1 moves if $x_1 > 0 \wedge x_2 \leq 0$

An example run under Sch:

(x_1, y_1, x_2, y_2)

$\rightarrow (4,1,4,1)$ $x_1 = x_2 \wedge y_1 = y_2$

$\rightarrow_{1,2} (3,1,2,1)$

$\rightarrow_{1,2} (2,1,0,1)$ prog2 terminated

$\rightarrow_{1} (1,1,0,1)$

$\rightarrow_{1} (0,1,0,1)$ prog1 terminated

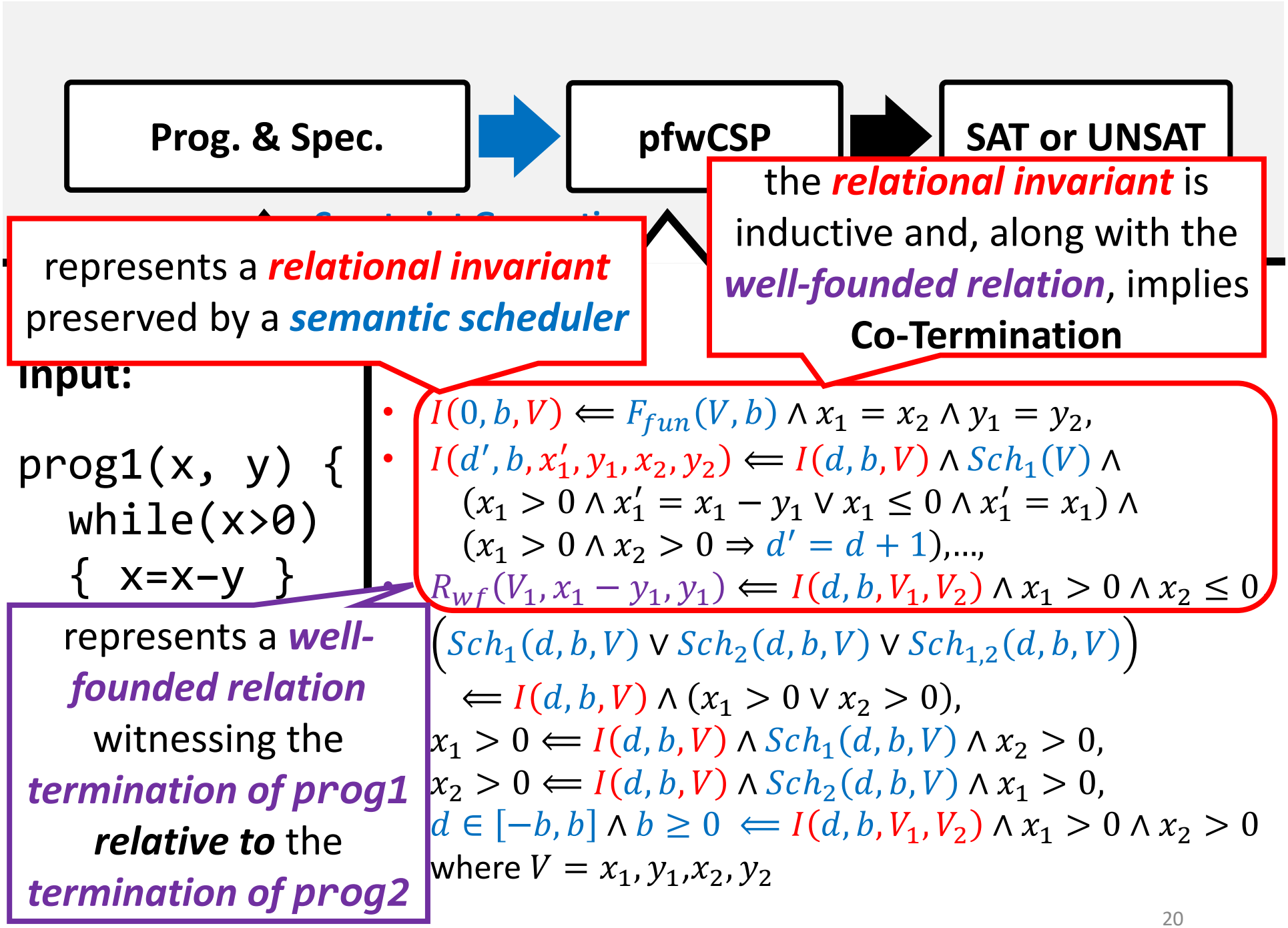
Input:

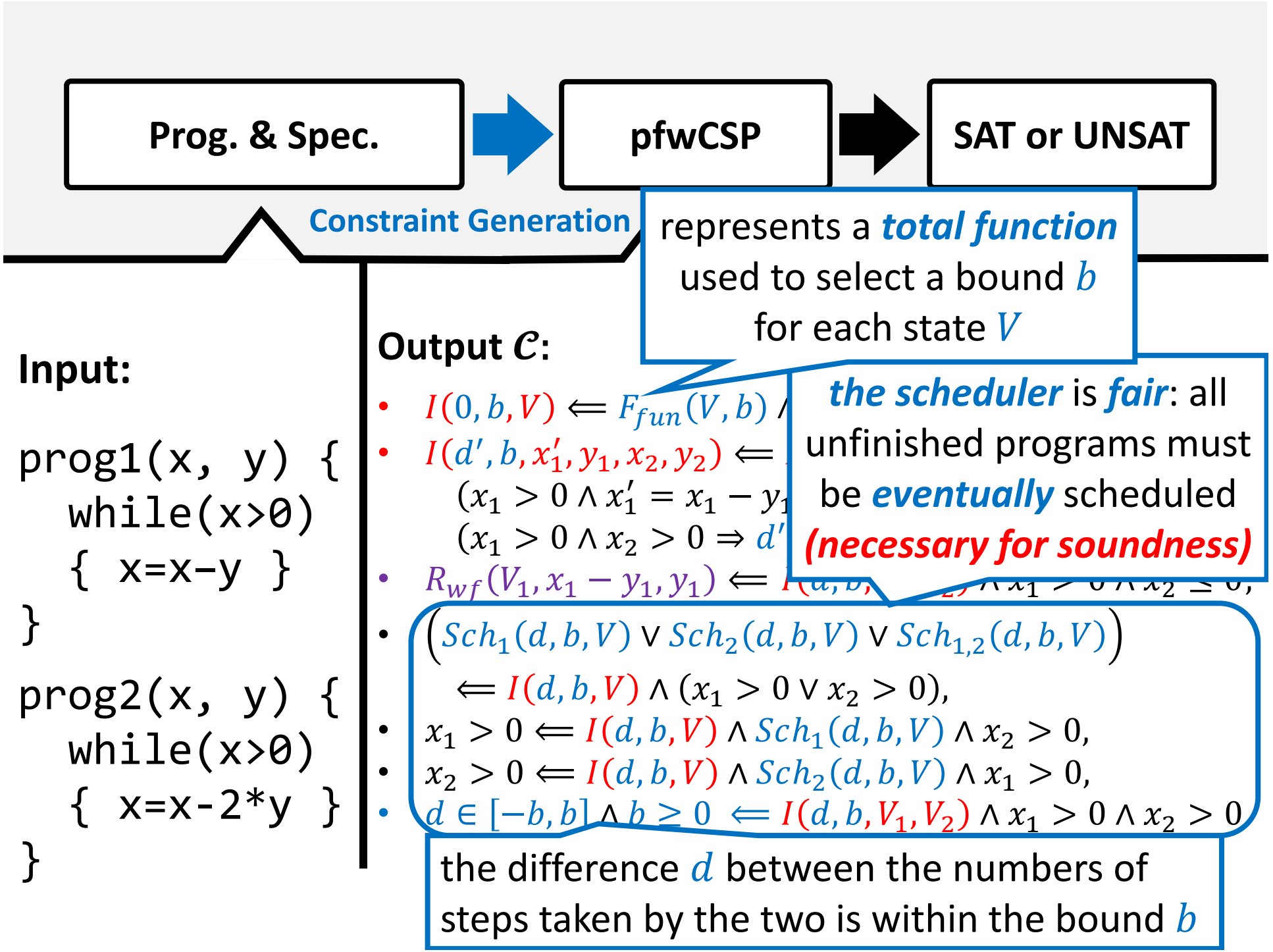
```
prog1(x, y) {
  while(x>0)
  { x=x-y }
}

prog2(x, y) {
  while(x>0)
  { x=x-2*y }
}
```

Output $\mathcal{C}$:

- $I(0,b,V) \Leftarrow F_{fun}(V,b) \wedge x_1 = x_2 \wedge y_1 = y_2,$
- $I(d',b,x_1',y_1,x_2,y_2) \Leftarrow I(d,b,V) \wedge Sch_1(V) \wedge$
  $(x_1 > 0 \wedge x_1' = x_1 - y_1 \vee x_1 \leq 0 \wedge x_1' = x_1) \wedge$
  $(x_1 > 0 \wedge x_2 > 0 \Rightarrow d' = d + 1)$,...,
- $R_{wf}(V_1, x_1 - y_1, y_1) \Leftarrow I(d,b,V_1,V_2) \wedge x_1 > 0 \wedge x_2 \leq 0,$
- $\big(Sch_1(d,b,V) \vee Sch_2(d,b,V) \vee Sch_{1,2}(d,b,V)\big)$
  $\Leftarrow I(d,b,V) \wedge (x_1 > 0 \vee x_2 > 0),$
- $x_1 > 0 \Leftarrow I(d,b,V) \wedge Sch_1(d,b,V) \wedge x_2 > 0,$
- $x_2 > 0 \Leftarrow I(d,b,V) \wedge Sch_2(d,b,V) \wedge x_1 > 0,$
- $d \in [-b,b] \wedge b \geq 0 \Leftarrow I(d,b,V_1,V_2) \wedge x_1 > 0 \wedge x_2 > 0$

where $V = x_1, y_1, x_2, y_2$


Prog. & Spec.
pfwCSP
SAT or UNSAT
represents a relational invariant preserved by a semantic scheduler
the relational invariant is inductive and, along with the well-founded relation, implies Co-Termination
input:
prog1(x, y) {
while(x>0)
{ x=x-y }
represents a well-founded relation witnessing the termination of prog1 relative to the termination of prog2
• $I(0,b,V) \Leftarrow F_{fun}(V,b) \wedge x_1 = x_2 \wedge y_1 = y_2$,
• $I(d',b,x_1',y_1,x_2,y_2) \Leftarrow I(d,b,V) \wedge Sch_1(V) \wedge (x_1 > 0 \wedge x_1' = x_1 - y_1 \vee x_1 \leq 0 \wedge x_1' = x_1) \wedge (x_1 > 0 \wedge x_2 > 0 \Rightarrow d' = d+1)$,...,
$R_{wf}(V_1, x_1 - y_1, y_1) \Leftarrow I(d,b,V_1,V_2) \wedge x_1 > 0 \wedge x_2 \leq 0$
$\left(Sch_1(d,b,V) \vee Sch_2(d,b,V) \vee Sch_{1,2}(d,b,V)\right) \Leftarrow I(d,b,V) \wedge (x_1 > 0 \vee x_2 > 0)$,
$x_1 > 0 \Leftarrow I(d,b,V) \wedge Sch_1(d,b,V) \wedge x_2 > 0$,
$x_2 > 0 \Leftarrow I(d,b,V) \wedge Sch_2(d,b,V) \wedge x_1 > 0$,
$d \in [-b,b] \wedge b \geq 0 \Leftarrow I(d,b,V_1,V_2) \wedge x_1 > 0 \wedge x_2 > 0$
where $V = x_1, y_1, x_2, y_2$


Prog. & Spec.
pfwCSP
SAT or UNSAT
Constraint Generation
Input:
prog1(x, y) {
while(x>0)
{ x=x-y }
}
prog2(x, y) {
while(x>0)
{ x=x-2*y }
}
Output $\mathcal{C}$:
represents a total function
used to select a bound b
for each state V
the scheduler is fair: all
unfinished programs must
be eventually scheduled
(necessary for soundness)
$I(0,b,V) \Leftarrow F_{fun}(V,b) \wedge$
$I(d',b,x_1',y_1,x_2,y_2) \Leftarrow$
$(x_1 > 0 \wedge x_1' = x_1 - y_1$
$(x_1 > 0 \wedge x_2 > 0 \Rightarrow d'$
$R_{wf}(V_1, x_1 - y_1, y_1) \Leftarrow$
$\left(Sch_1(d,b,V) \vee Sch_2(d,b,V) \vee Sch_{1,2}(d,b,V)\right)$
$\Leftarrow I(d,b,V) \wedge (x_1 > 0 \vee x_2 > 0)$,
$x_1 > 0 \Leftarrow I(d,b,V) \wedge Sch_1(d,b,V) \wedge x_2 > 0$,
$x_2 > 0 \Leftarrow I(d,b,V) \wedge Sch_2(d,b,V) \wedge x_1 > 0$,
$d \in [-b,b] \wedge b \geq 0 \Leftarrow I(d,b,V_1,V_2) \wedge x_1 > 0 \wedge x_2 > 0$
the difference d between the numbers of
steps taken by the two is within the bound b

Relational Verification via Synthesis of Relational Invariants and Fair & Semantic Schedulers

1. ***k*-safety ⊂ Hypersafety**
   - ***Termination-Insensitive Non-Interference (TI-NI)* ∈ 2-safety**
     - A well-studied security property formalizing the absence of information leakage [Goguen+ '82]
2. **Co-termination ∈ Hyperliveness**
   - Formalize that multiple programs agree on termination
   - ***Termination-Sensitive NI (TS-NI)* = *TI-NI* ∧ *Co-termination***
3. ***(TI-/TS-) Generalized NI (GNI)* ∈ ∀∃ *hyperproperties***
   - A generalization of ***NI*** for non-deterministic programs

Example: (TI-/TS-)GNI ∈ ∀∃hyperproperties

- gniEx(h, l) ***non-deterministically*** returns a value $x \geq l$ in two different ways depending on the ***high security*** input h

```
gniEx(bool high, int low) {
  if(high) {
    int x = nondet_int();
    if(x >= low) { return x }
    else { while(true) {} }
  } else {
    int x = low;
    while(nondet_bool()){x++}
    return x;
  }
}
```

Can an attacker infer the value of h by observing the ***low security*** input l and the return value x?

No! Forma…

The ***demonic*** (∀) side

$\forall h_1, h_2, l, x_1.\ \mathbf{gniEx}(h_1, l) \rightsquigarrow x_1 \Rightarrow$
$\mathbf{gniEx}(h_2, l) \rightsquigarrow \bot \vee$
$\exists x_2.\ \mathbf{gniEx}(h_2, l) \rightsquigarrow x_2 \wedge x_1 = x_2$

gniEx also…

The ***angelic*** (∃) side

$\forall h_1, h_2, l, x_1.\ \mathbf{gniEx}(h_1, l) \rightsquigarrow x_1 \Rightarrow$
$\exists x_2.\ \mathbf{gniEx}(h_2, l) \rightsquigarrow x_2 \wedge x_1 = x_2$

Our Approach to (TI-/TS-)GNI

- Find a ***fair semantic scheduler***, a ***relational invariant***, a ***well-founded relation***, and ***strategies for the non-deterministic choices of the angelic side***
- Augment the encodings for **TI-NI** and **Co-Term** with
  - ***predicate variables*** that represent the ***strategies: total functions from states to choices of the angelic side***
  - ***prophecy variables*** that represent the final outputs of ***the demonic side (necessary for the completeness)***
- Please refer to the CAV'21 paper for details

Main Contributions

- ***Sound*** and ***complete*** reduction to ***constraint solving***
  - Key observation: relational verification amounts to synthesis of ***relational invariants*** and ***fair & semantic schedulers***
- New expressive class of predicate constraints **pfwCSP**
  - Extend the class of CHCs with ***non-Horn clauses*** and ***functional*** / ***well-founded*** predicate variables
- ***Stratified CEGIS*** for **pfwCSP *solving***
  - Combine CEGIS with ***stratified families of templates*** for achieving ***completeness*** and ***efficiency***

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

- Generalize CHCs [Bjørner+ '15] with ***non-Horn clauses***
- A finite set $\mathcal{C}$ of ***clauses*** of the form:

$$\left(X_1(\tilde{t_1}) \vee \cdots \vee X_\ell(\tilde{t_\ell})\right) \Leftarrow \left(X_{\ell+1}(\tilde{t_{\ell+1}}) \wedge \cdots \wedge X_m(\tilde{t_m}) \wedge \phi\right)$$

$\mathcal{C}$ is ***CHCs*** if $\ell \leq 1$ for all clause in $\mathcal{C}$

where $X_1, \ldots, X_\ell, X_{\ell+1}, \ldots, X_m$ are predicate variables,
$\tilde{t_1}, \ldots, \tilde{t_m}$ 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 $\rho \vDash \bigwedge \mathcal{C}$

pfwCSP: pCSP with Functional & Well-founded Predicate Variables

- **pfwCSP** is pCSP with ***predicate variable kinds:***
  - ***Ordinary, Functional, Well-founded***
- A finite set $\mathcal{C}$ of pfwCSP clauses is ***satisfiable*** (modulo T) if there is an interpretation ρ of predicate variables such that
  - $\rho \vDash \bigwedge \mathcal{C}$
  - $\forall X \in$ ***Functional*** predicate variables.
    $\rho(X)$ is a ***total function***
  - $\forall X \in$ ***Well–founded*** predicate variables.
    $\rho(X)$ is a ***well-founded relation***

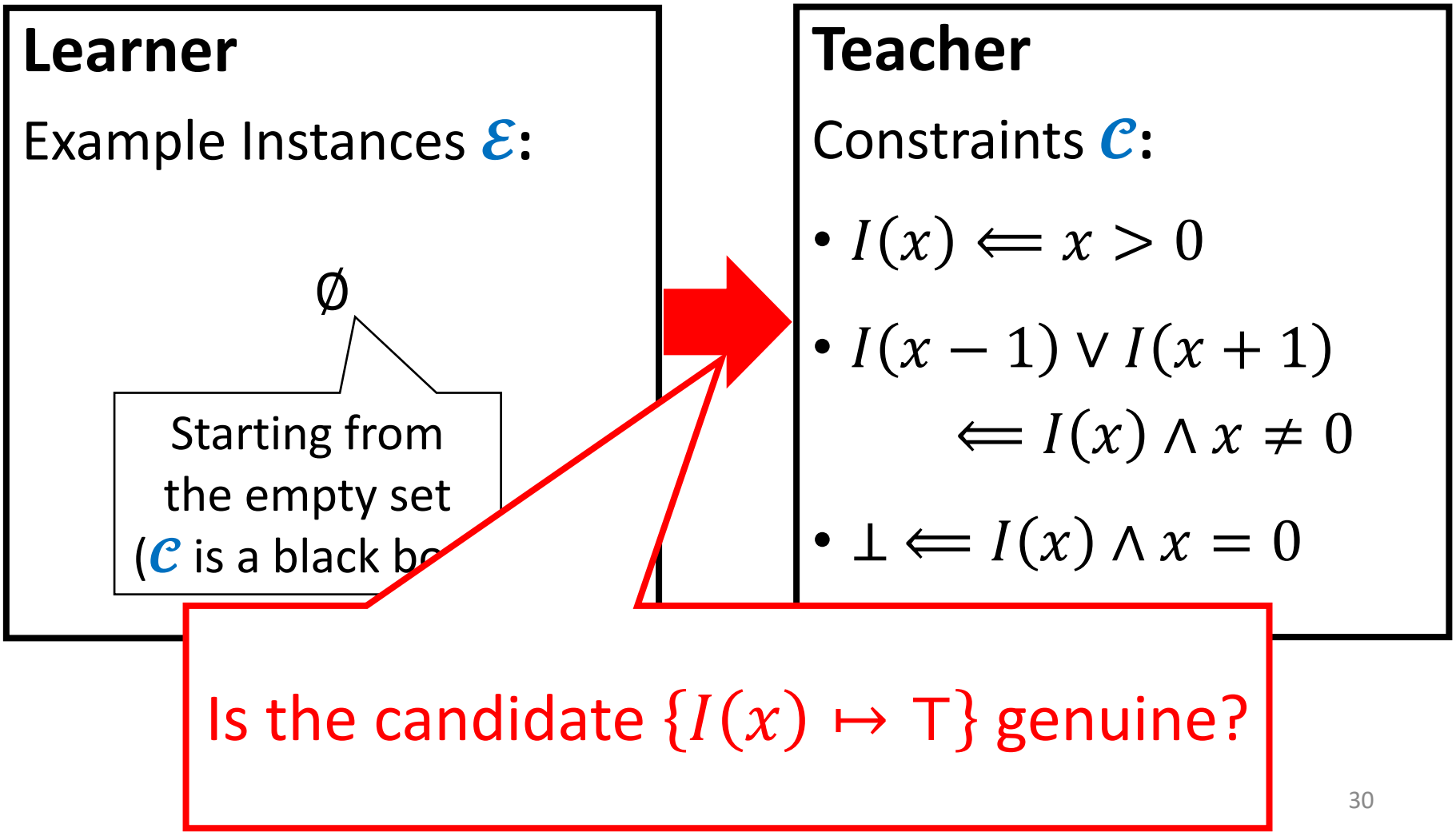
Main Contributions

- ***Sound*** and ***complete*** reduction to ***constraint solving***
  - Key observation: relational verification amounts to synthesis of ***relational invariants*** and ***fair & semantic schedulers***
- New expressive class of predicate constraints **pfwCSP**
  - Extend the class of CHCs with ***non-Horn clauses*** and ***functional*** / ***well-founded*** predicate variables
- ***Stratified CEGIS*** for **pfwCSP *solving***
  - Combine CEGIS with ***stratified families of templates*** for achieving ***completeness*** and ***efficiency***

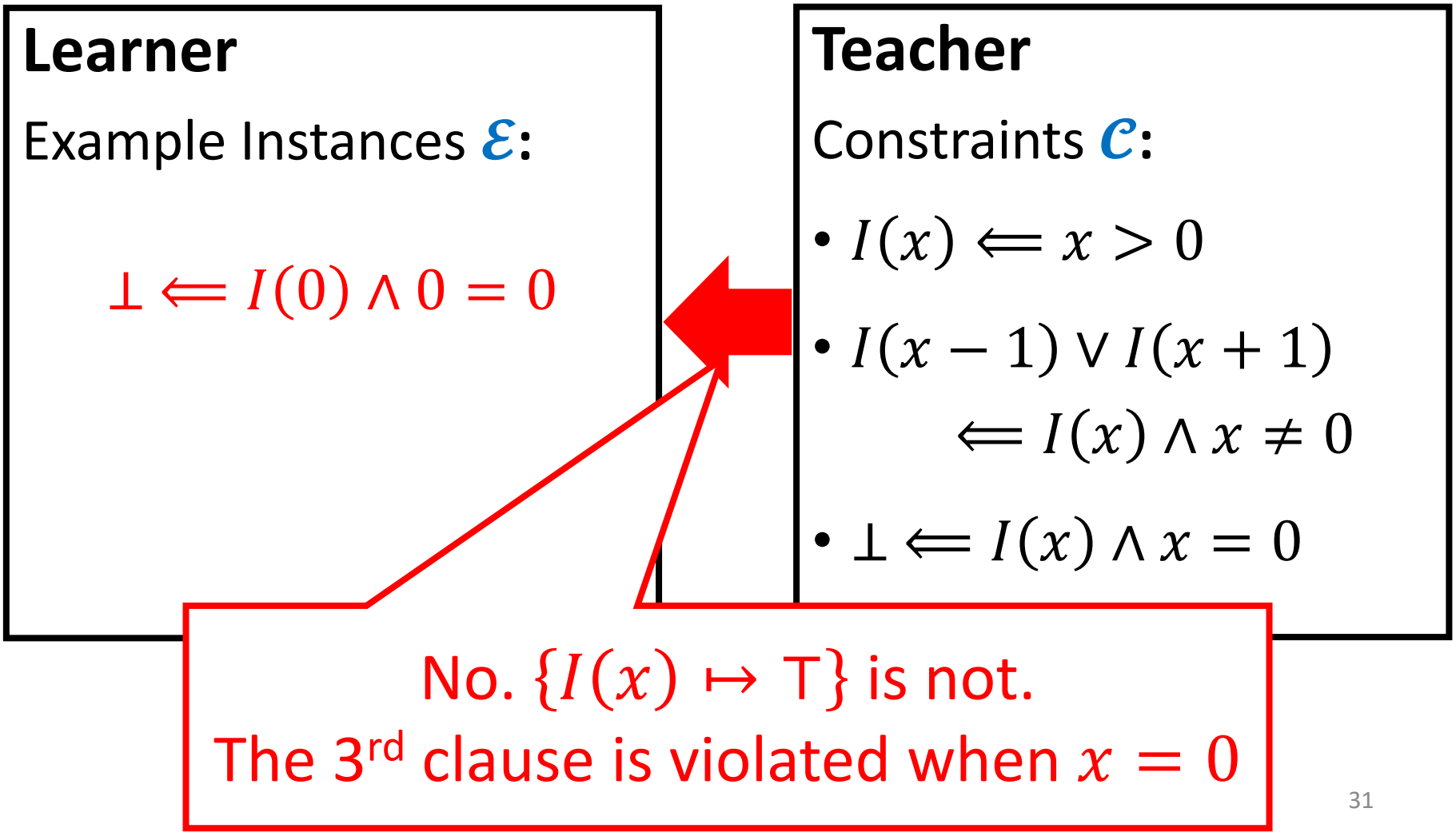
CounterExample Guided Inductive Synthesis (CEGIS) [Solar-Lezama+ '06]

- 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 $\mathcal{C}$
      - If yes, return ρ as a genuine solution of $\mathcal{C}$
      - If no, repeat the procedure by adding new example instances witnessing non-satisfaction of $\mathcal{C}$ by ρ

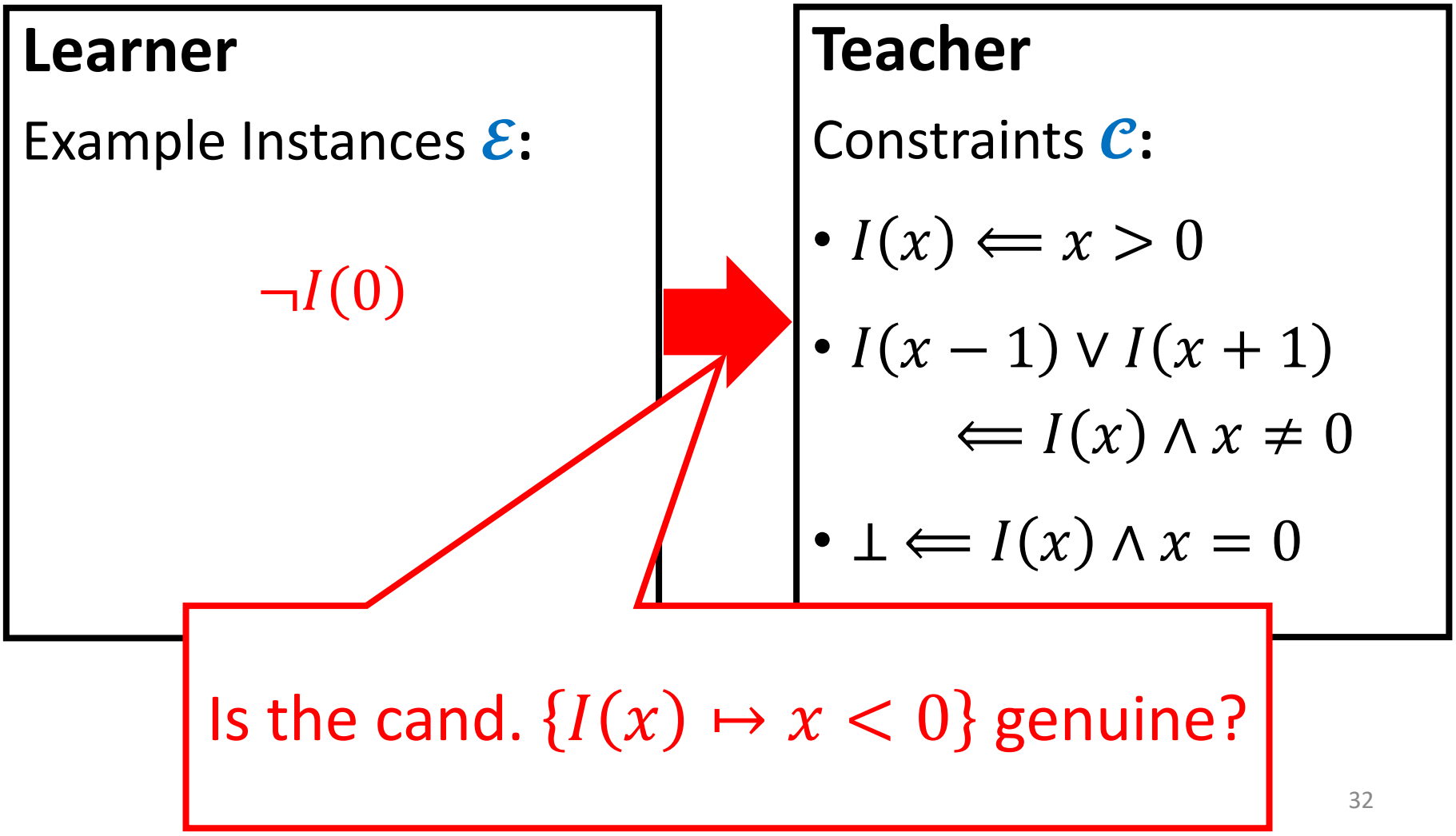
Example Run of CEGIS



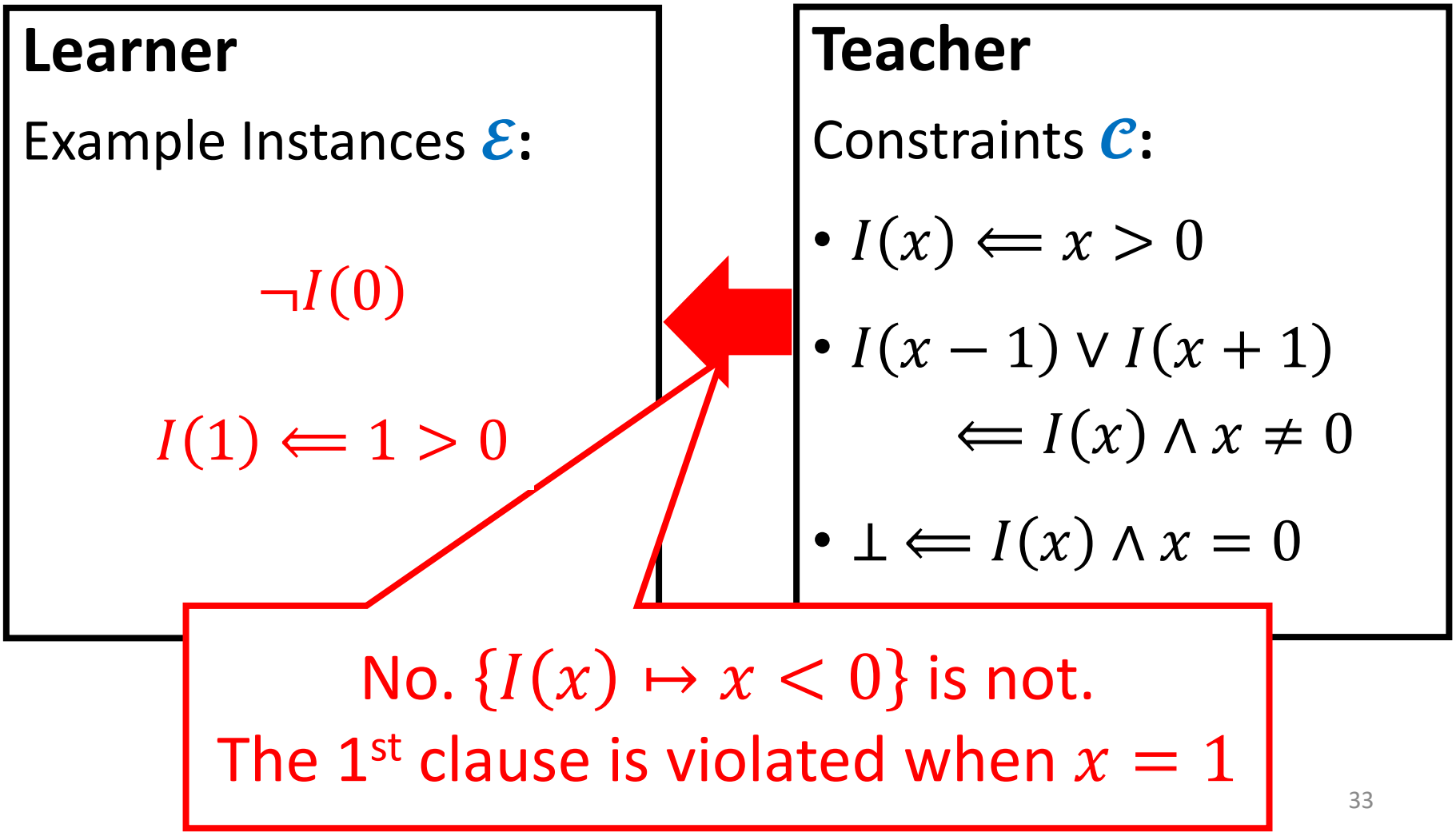
Example Run of CEGIS



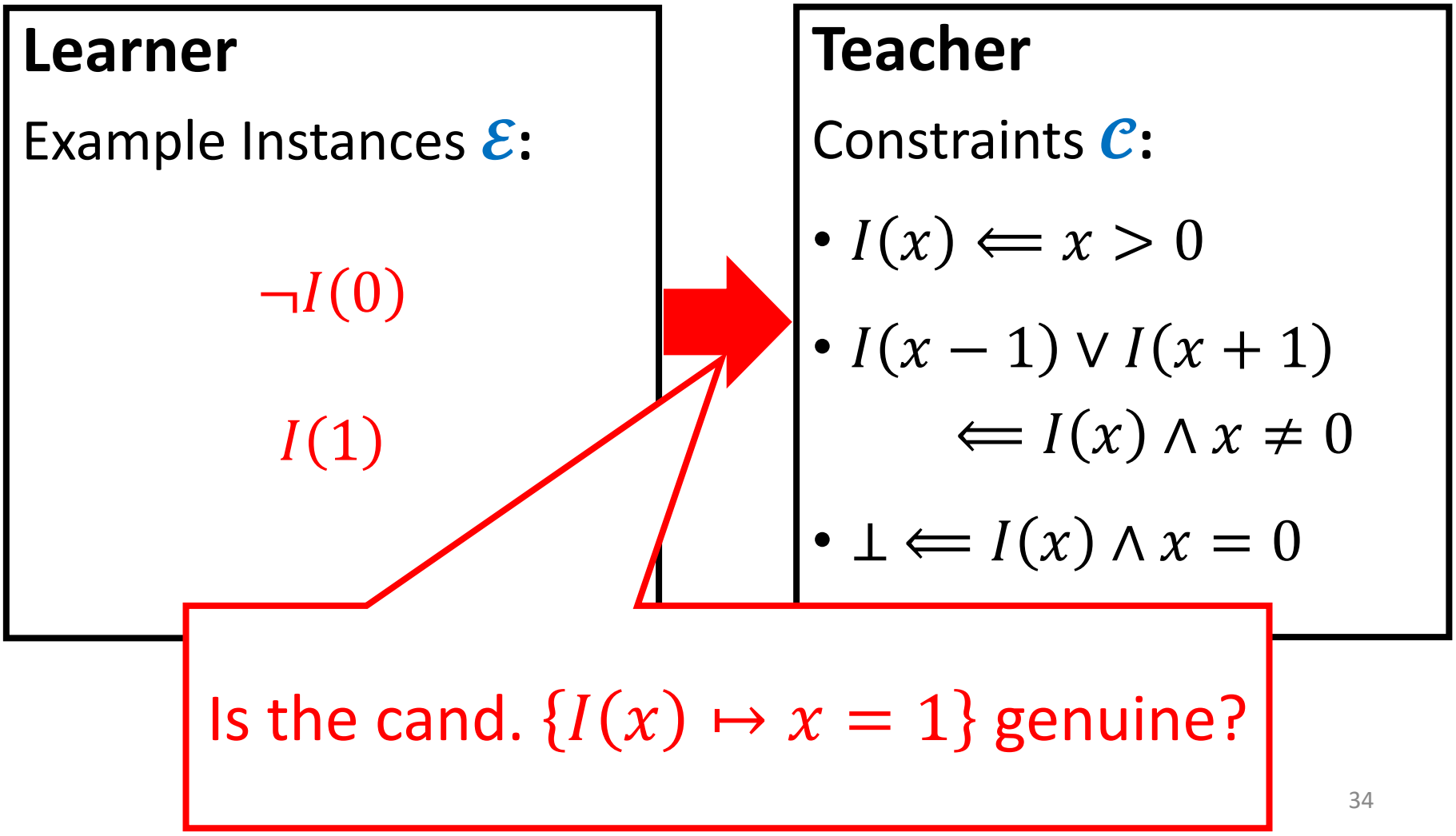
Example Run of CEGIS



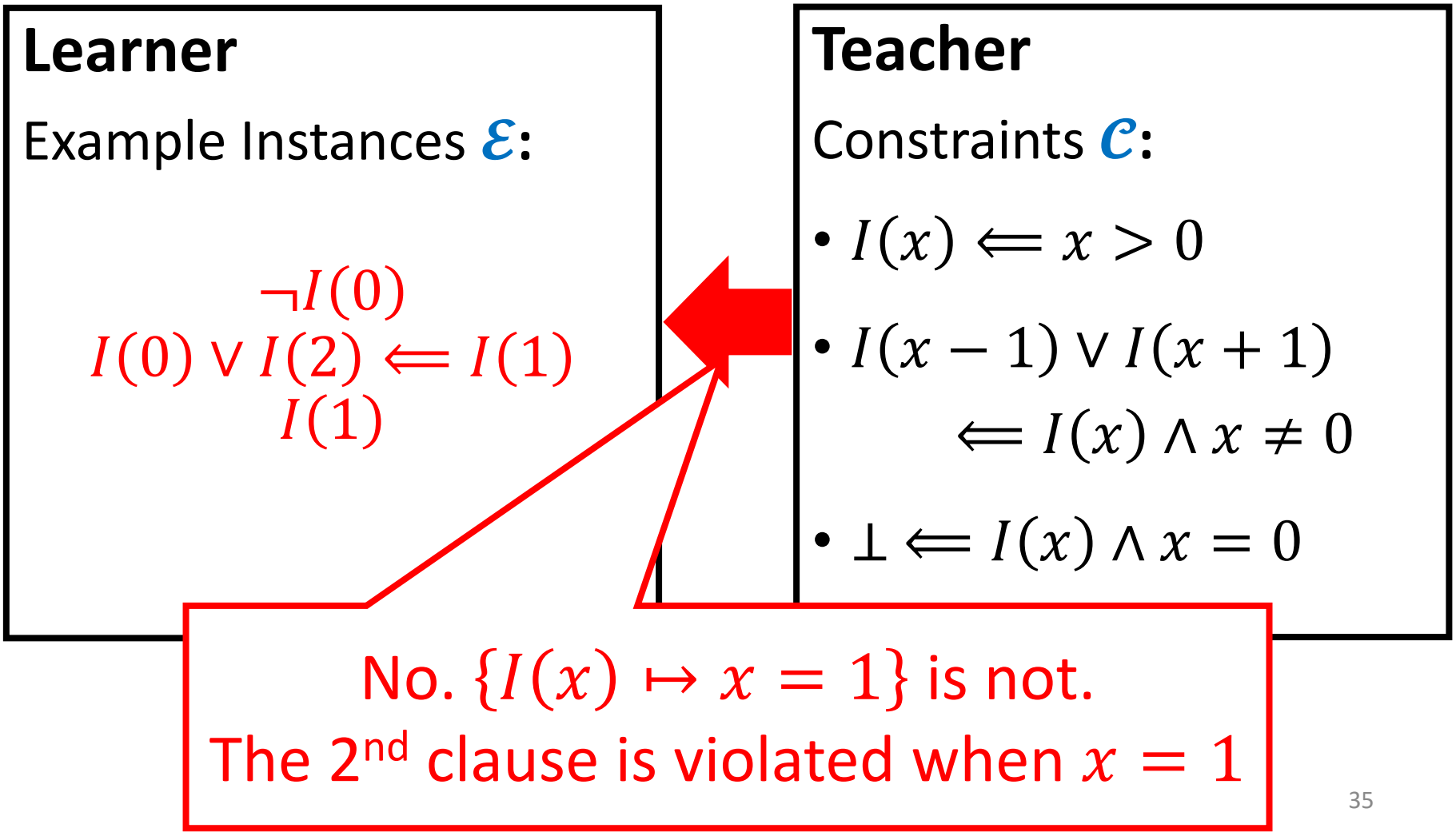
Example Run of CEGIS



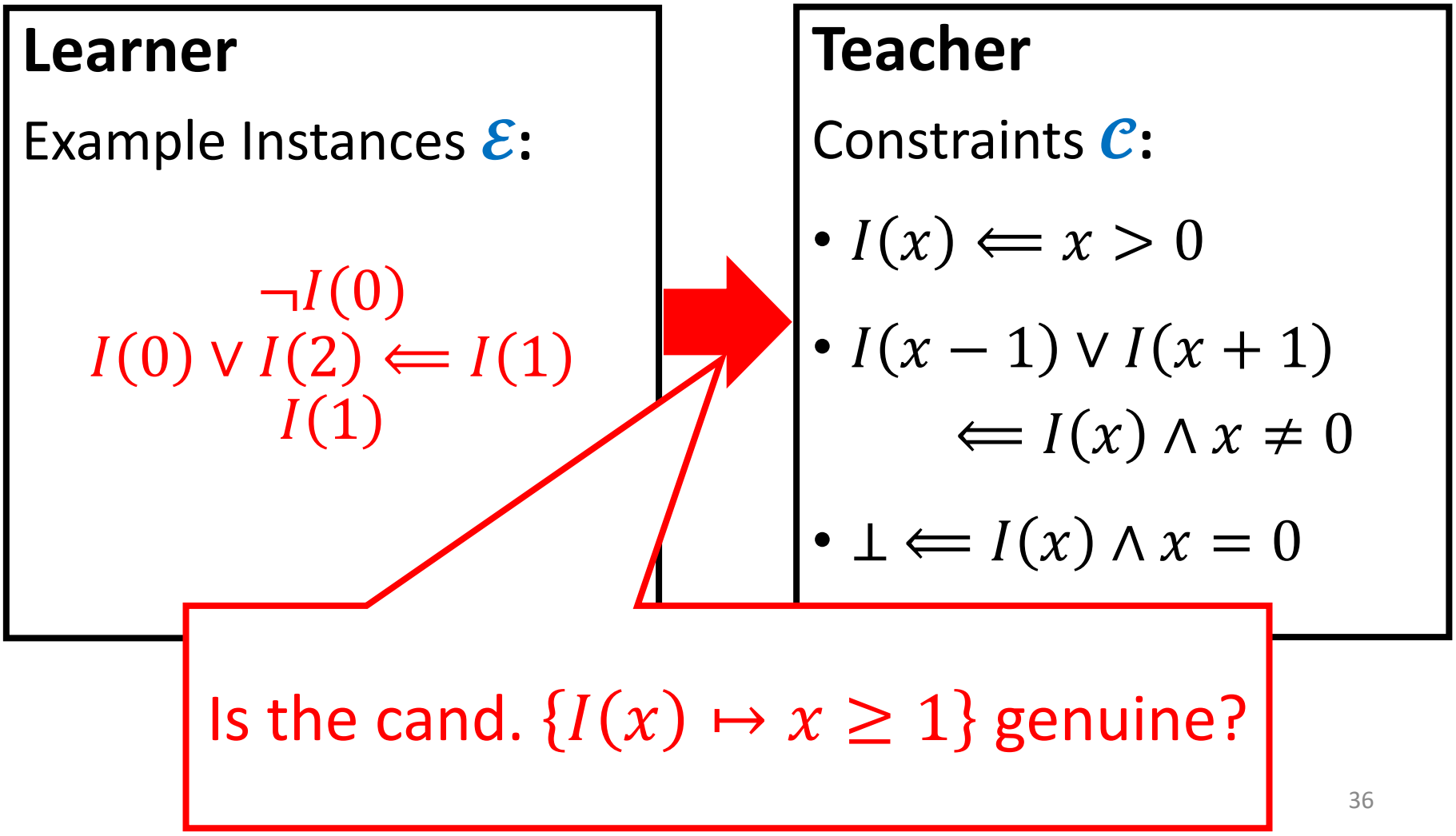
Example Run of CEGIS



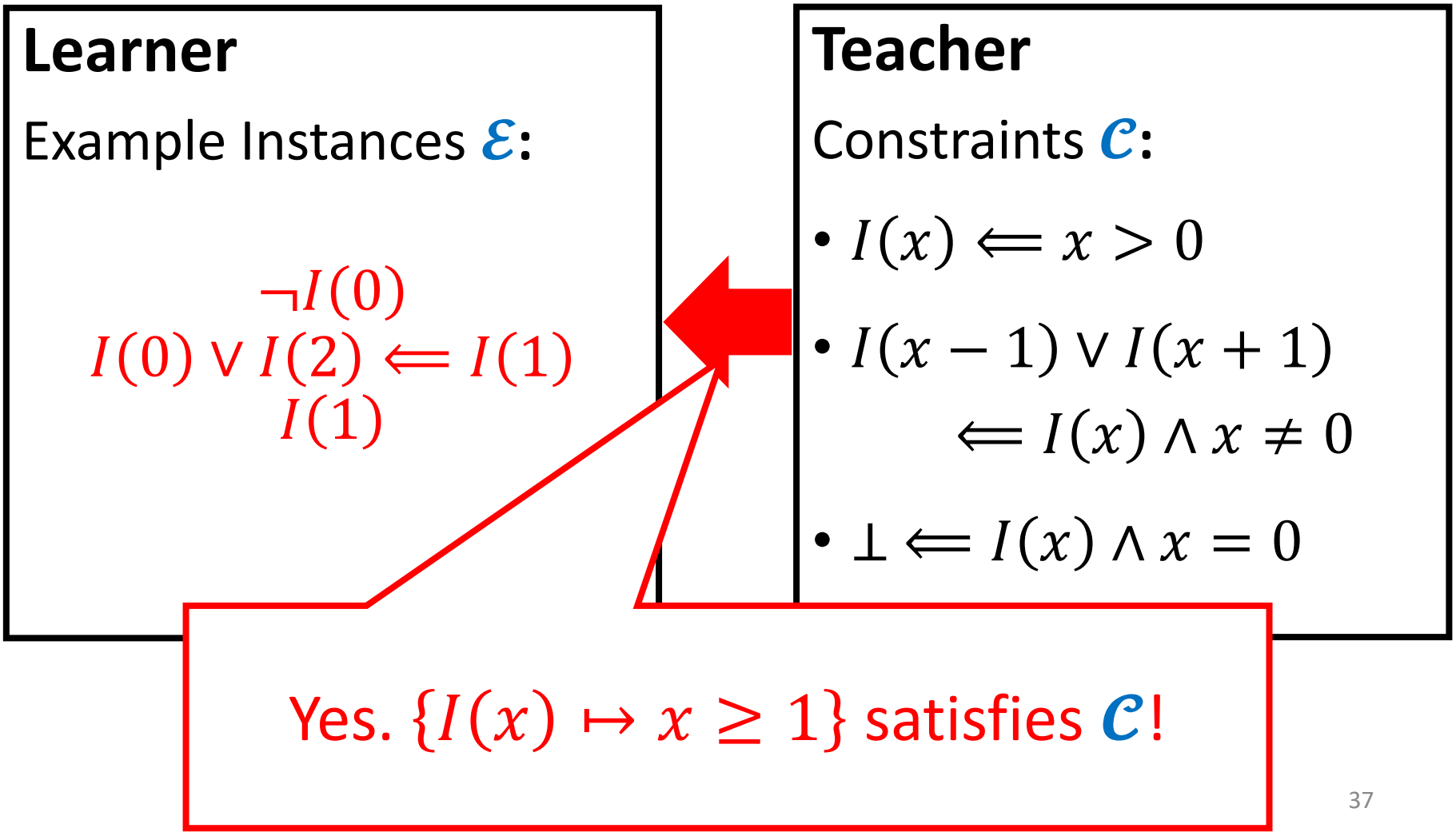
Example Run of CEGIS



Example Run of CEGIS



Example Run of CEGIS



Template-based Learner

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

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

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

Stratified Template Families

- ***Stratified template family*** was proposed for CEGAR-based *invariant synthesis* [Terauchi+ '15]
  - To achieve ***completeness***, often at the price of ***efficiency***
  - Search for solutions in a stratified manner: Iteratively update templates to more expressive ones in the family as needed.
- Our **pfwCSP *solving*** method combines CEGIS with ***stratified template families*** of ***ordinary/functional/well-founded*** predicates
  - To achieve not only ***completeness*** but also ***efficiency***
  - The ***data-driven nature*** of CEGIS is a good match: the stratified search accelerates the convergence by ***avoiding the overfitting problem*** [Padhi+ '19] of expressive templates to examples

Implementation and Evaluation

- Evaluated our solver **PCSat** for **pfwCSP** on 20 relational verification problems:
  - ➢15 solved fully automatically, 5 required small hints

Program	Time (s)	#Iters	Program	Time (s)	#Iters
DoubleSquareNI_hFT	17.762	42	HalfSquareNI	11.853	35
DoubleSquareNI_hTF	26.495	55	ArrayInsert‡	118.671	73
DoubleSquareNI_hFF	2.944	9	SquareSum†‡	337.596	117
DoubleSquareNI_hTT	4.055	11	SimpleTS_GNI1	5.397	14
CotermIntro1	19.322	80	SimpleTS_GNI2	8.919	26
CotermIntro2	15.871	73	InfBranchTS_GNI	2.607	4
TS_GNI_hFT†	47.083	78	TI_GNI_hFT†	4.389	16
TS_GNI_hTF	5.076	17	TI_GNI_hTF	2.277	6
TS_GNI_hFF	7.174	24	TI_GNI_hFF	2.968	6
TS_GNI_hTT†	23.495	53	TI_GNI_hTT	4.148	22

Summary

- New class **pfwCSP** of predicate constraints, generalizing **CHCs** with ***non-Horn clauses*** and ***functional/well-founded*** predicates
- ***Sound*** and ***complete*** reduction of ***relational verification*** (of k-safety, co-termination, and GNI) to **pfwCSP *solving***
  - ***relational verification*** amounts to synthesis of ***relational invariants*** and ***fair & semantic schedulers***, which can be characterized by **pfwCSP**
- ***Stratified CEGIS*** for **pfwCSP *solving***
  - Combine CEGIS with ***stratified families of templates*** for achieving ***completeness*** and ***efficiency***
- Implementation of a solver **PCSat** and Evaluation
  - Available from: https://github.com/hiroshi-unno/coar