Constraint-based Relational Verification

To appear in CAV 2021

Joint work with Hiroshi Unno and Tachio Terauchi.

```
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;
}
```

```
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;
}
```

- *Termination-sensitive non-interference*: can't infer the value of secrets (variable h) from observing the output (variable y).

```
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;
}
```

- *Termination-sensitive non-interference*: can't infer the value of secrets (variable h) from observing the output (variable y).

- 2-safety property.

```
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;
}
```

- *Termination-sensitive non-interference*: can't infer the value of secrets (variable h) from observing the output (variable y).

- 2-safety property.

- If two executions agree on x (*Pre relation*: $x_1 == x_2$), do they agree on resulting y (*Post relation*: $y_1 == y_2$)?

Strategy 1: Sequential Composition

```
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;
}
```

Strategy 1: Sequential Composition

```
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;
}
```


```
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;
}
```

Strategy 1: Sequential Composition

```
doubleSquare(bool h1, int x1) {
  int z1, y1=0;
  if (h1) { z1 = 2*x1; } else { z1 = x1; }
  while (z1>0) { z1--; y1 = y1+x1; }
  if (!h1) { y1 = 2*y1; }
  return y1;
}
```

```
doubleSquare(bool h2, int x2) {
  int z2, y2=0;
  if (h2) { z2 = 2*x2; } else { z2 = x2; }
  while (z2>0) { z2--; y2 = y2+x2; }
  if (!h2) { y2 = 2*y2; }
  return y2;
}
```

Strategy 1: Sequential Composition

```
main {
    bool h1, h2, int x1,x2;
    assume(x1 == x2);

      int z1, y1=0;
      if (h1) { z1 = 2*x1; } else { z1 = x1; }
      while (z1>0) { z1--; y1 = y1+x1; }
      if (!h1) { y1 = 2*y1; }
```

First copy has completed here.

```
      int z2, y2=0;
      if (h2) { z2 = 2*x2; } else { z2 = x2; }
      while (z2>0) { z2--; y2 = y2+x2; }
      if (!h2) { y2 = 2*y2; }

    assert(y1 == y2);
}
```

Second copy has completed here.

Strategy 1: Sequential Composition

```
main {
    bool h1, h2, int x1,x2;
    assume(x1 == x2);

      int z1, y1=0;
      if (h1) { z1 = 2*x1; } else { z1 = x1; }
      while (z1>0) { z1--; y1 = y1+x1; }
      if (!h1) { y1 = 2*y1; }
```

First copy has completed here.

```
      int z2, y2=0;
      if (h2) { z2 = 2*x2; } else { z2 = x2; }
      while (z2>0) { z2--; y2 = y2+x2; }
      if (!h2) { y2 = 2*y2; }

    assert(y1 == y2);
}
```

Second copy has completed here.

To prove this, need to know the full specification of doubleSquare!

Strategy 2: Self-composition or product program:

```
int z1, y1=0;
if (h1) { z1 = 2*x1; } else { z1 = x1; }
while (z1>0) { z1--; y1 = y1+x1; }
if (!h1) { y1 = 2*y1; }
```

```
int z2, y2=0;
if (h2) { z2 = 2*x2; } else { z2 = x2; }
while (z2>0) { z2--; y2 = y2+x2; }
if (!h2) { y2 = 2*y2; }
```

Strategy 2: Self-composition or product program:

```
int z1, y1=0;

if (h1) { z1 = 2*x1; } else { z1 = x1; }

while (z1>0) { z1--; y1 = y1+x1; }

if (!h1) { y1 = 2*y1; }
```

```
int z2, y2=0;

if (h2) { z2 = 2*x2; } else { z2 = x2; }

while (z2>0) { z2--; y2 = y2+x2; }

if (!h2) { y2 = 2*y2; }
```

Strategy 2: Self-composition or product program:

```
int z1, y1=0;

if (h1) { z1 = 2*x1; } else { z1 = x1; }

while (z1>0) { z1--; y1 = y1+x1; }

int z2, y2=0;

if (!h1) { y1 = 2*y1; }

if (h2) { z2 = 2*x2; } else { z2 = x2; }

while (z2>0) { z2--; y2 = y2+x2; }

if (!h2) { y2 = 2*y2; }
```

Strategy 2: Self-composition or product program:

```
int z1, y1=0;
int z2, y2=0;

if (h1) { z1 = 2*x1; } else { z1 = x1; }
if (h2) { z2 = 2*x2; } else { z2 = x2; }

while (z1>0) { z1--; y1 = y1+x1; }
while (z2>0) { z2--; y2 = y2+x2; }

if (!h1) { y1 = 2*y1; }
if (!h2) { y2 = 2*y2; }
```

Strategy 2: Self-composition or product program:

```
int z1, y1=0;
int z2, y2=0;
if (h1) { z1 = 2*x1; } else { z1 = x1; }
if (h2) { z2 = 2*x2; } else { z2 = x2; }
while (z1>0 && z2>0) {
    z1–;
    z2–;
    y1 = y1+x1;
    y2 = y2+x2;


}
if (!h1) { y1 = 2*y1; }
if (!h2) { y2 = 2*y2; }
```

Strategy 2: Self-composition or product program:

```
main {
    bool h1, h2, int x1,x2;
    assume(x1 == x2);


     int z1, y1=0;
     int z2, y2=0;
     if (h1) { z1 = 2*x1; } else { z1 = x1; }
     if (h2) { z2 = 2*x2; } else { z2 = x2; }
     while (z1>0 && z2>0) {
        z1–;
        z2–;
        y1 = y1+x1;
        y2 = y2+x2;


    }
    if (!h1) { y1 = 2*y1; }
    if (!h2) { y2 = 2*y2; }
```

Strategy 2: Self-composition or product program:

```
main {
    bool h1, h2, int x1,x2;
    assume(x1 == x2);


     int z1, y1=0;
     int z2, y2=0;
     if (h1) { z1 = 2*x1; } else { z1 = x1; }
     if (h2) { z2 = 2*x2; } else { z2 = x2; }
     while (z1>0 && z2>0) {
         z1–;
         z2–;
         y1 = y1+x1;
         y2 = y2+x2;


     }
     if (!h1) { y1 = 2*y1; }
     if (!h2) { y2 = 2*y2; }

    assert(y1 > 0 <==> y2 > 0);
}
```

Strategy 2: Self-composition or product program:

```
main {
    bool h1, h2, int x1,x2;
    assume(x1 == x2);

    { x1 > 0 <==> x2 > 0 }

    int z1, y1=0;
    int z2, y2=0;
    if (h1) { z1 = 2*x1; } else { z1 = x1; }
    if (h2) { z2 = 2*x2; } else { z2 = x2; }
    while (z1>0 && z2>0) {
        z1−;
        z2−;
        y1 = y1+x1;
        y2 = y2+x2;


    }
    if (!h1) { y1 = 2*y1; }
    if (!h2) { y2 = 2*y2; }

    assert(y1 > 0 <==> y2 > 0);
}
```

Strategy 2: Self-composition or product program:

```
main {
    bool h1, h2, int x1,x2;
    assume(x1 == x2);

    { x1 > 0 <==> x2 > 0 }

    int z1, y1=0;
    int z2, y2=0;
    if (h1) { z1 = 2*x1; } else { z1 = x1; }
    if (h2) { z2 = 2*x2; } else { z2 = x2; }
    while (z1>0 && z2>0) {
        z1−;
        z2−;
        y1 = y1+x1;
        y2 = y2+x2;

        { y1 > 0 <==> y2 > 0 }
    }
    if (!h1) { y1 = 2*y1; }
    if (!h2) { y2 = 2*y2; }

    assert(y1 > 0 <==> y2 > 0);
}
```

Strategy 2: Self-composition or product program:

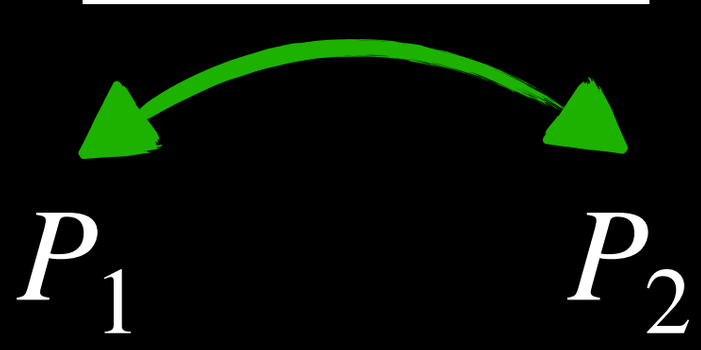
```
main {
    bool h1, h2, int x1,x2;
    assume(x1 == x2);

    { x1 > 0 <==> x2 > 0 }

    int z1, y1=0;
    int z2, y2=0;
    if (h1) { z1 = 2*x1; } else { z1 = x1; }
    if (h2) { z2 = 2*x2; } else { z2 = x2; }
    while (z1>0 && z2>0) {
        z1−;
        z2−;
        y1 = y1+x1;
        y2 = y2+x2;

        { y1 > 0 <==> y2 > 0 }
    }
    if (!h1) { y1 = 2*y1; }
    if (!h2) { y2 = 2*y2; }

    assert(y1 == y2);
}
```

Impossible to prove this with “lock-step” scheduling and only linear arithmetic invariants.

— Schemer et al, CAV 2019.

Semantic Scheduler

Examine the states σ_1, σ_2

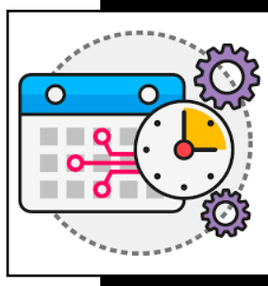


Decide whether P_1 or P_2 should go next and how many steps to take.

Schemer et al, CAV 2019.

```
pre(x1 == x2)

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;
}

post(y1 == y2)
```

```
predicates:
  h1, h2, x1 > 0, y1 ≥ 0, y2 ≥ 0, z2 ≥ 0,
  z2 ≥ 0, x1 = x2, y1 = y2, y1 = 2y2, y2 = 2y1,
  z1 = z2, z1 = 2z2, z2 = 2z1, z1 = 2z2 − 1,
  z2 = 2z1 − 1, y1 = 2y2 + x2, y2 = 2y1 + x1

composition =
if(((z0 > 0 & z2 > 0 | (z1 ≤ 0 & z2 ≤ 0))
  && (h1 & z1 == 2 * z2)
  && !(h1 == h2 || (z1 == 0 & z2 == 0)))
    || (!(z1 > 0 & z2 > 0 | (z1 ≤ 0 & z2 ≤ 0))
      & z2 ≤ 0 & z1 > 0))
    step (1);
else if (((z1 > 0 & z2 > 0 | (z1 ≤ 0 & z2 ≤ 0))
      && !(h1 == h2 | (z1 == 0 & z2 == 0))
      && !(h1 & z1 == 2 * z2) & (z2 == 2 * z1))
        || !(z1 > 0 & z2 > 0 | (z1 ≤ 0 & z2 ≤ 0)))
      step (2);
else
    step(1,2);
```



- **Pro**: Composition can now be proved.
- **Con**: Manually provided predicates, limited to k-safety.

*There are limited existing techniques for **automatically** verifying such relational problems that require such semantic schedulers.*

First Key Idea

1. **Represent the scheduler** as a series of predicate variables:

$$\textbf{sch}_{\{1\}}, \textbf{sch}_{\{2\}}, \textbf{sch}_{\{1,2\}}$$

First Key Idea

2nd copy takes a step alone

1. **Represent the scheduler** as a series of predicate variables:

$$\textbf{sch}_{\{1\}}, \textbf{sch}_{\{2\}}, \textbf{sch}_{\{1,2\}}$$

First Key Idea

2nd copy takes a step alone

1. **Represent the scheduler** as a series of predicate variables:

$$\textbf{sch}_{\{1\}}, \textbf{sch}_{\{2\}}, \textbf{sch}_{\{1,2\}}$$

2. Pose the k-safety problem as constraints: over these **sch** scheduling predicates.

$$\textbf{inv}(V_1, V_2) \wedge \textbf{sch}_{\{1\}} \wedge T_1(V_1, V_1') \wedge V_2 = V_2' \Rightarrow \textbf{inv}(V_1', V_2')$$

First Key Idea

2nd copy takes a step alone

1. **Represent the scheduler** as a series of predicate variables:

$$\textbf{sch}_{\{1\}}, \textbf{sch}_{\{2\}}, \textbf{sch}_{\{1,2\}}$$

2. Pose the k-safety problem as constraints: over these **sch** scheduling predicates.

$$\textbf{inv}(V_1, V_2) \wedge \textbf{sch}_{\{1\}} \wedge T_1(V_1, V_1') \wedge V_2 = V_2' \Rightarrow \textbf{inv}(V_1', V_2')$$

3. More generally over k-tuples of programs, we have

$$\textbf{sch}_A \qquad\qquad \forall A \in \mathscr{P}^+[k]$$

First Key Idea

2nd copy takes a step alone

1. **Represent the scheduler** as a series of predicate variables:

$$\textbf{sch}_{\{1\}}, \textbf{sch}_{\{2\}}, \textbf{sch}_{\{1,2\}}$$

2. Pose the k-safety problem as constraints: over these **sch** scheduling predicates.

$$\textbf{inv}(V_1, V_2) \wedge \textbf{sch}_{\{1\}} \wedge T_1(V_1, V_1') \wedge V_2 = V_2' \Rightarrow \textbf{inv}(V_1', V_2')$$

3. More generally over k-tuples of programs, we have

$$\textbf{sch}_A \qquad\qquad \forall A \in \mathscr{P}^+[k]$$

So then search for a schedule is relegated to the constraint solver. (Move the problem elsewhere :-))

First Key Idea

Predicate solutions found by our tool:

$$\textbf{sch}_{\{1\}} = \lambda\tilde{V}\,.\, h_1 \wedge \neg h_2 \wedge z_1 + 1 = 2z_2$$
$$\textbf{sch}_{\{2\}} = \lambda\tilde{V}\,.\, \neg h_1 \wedge z_2 + 1 = 2z_1$$
$$\textbf{sch}_{\{1,2\}} = \lambda\tilde{V}\,.\, (h_1 \wedge \neg h_2 \Rightarrow z_1 = 2z_2) \wedge (\neg h_1 \wedge h_2 \wedge z_2 = 2z_1)$$

That is, *when the copy with* **h=true** *is scheduled, execute the loop two times per loop iteration with* **h=false***.*

Problem 2: Co-termination

```
P1: while (x1>0) { x1 = x1 - y1; }

P2: while (x2>0) { x2 = x2 - y2; }
```

- **Question**: do they agree on termination?
  - In general, no. eg when $x_1<0$ initially and $x_2>0 \wedge y_2=0$
    - But they do if *Pre*: $x_1=x_2 \wedge y_1=y_2$
- Not k-safety. Cannot be refuted by finite traces.

Problem 2: Co-termination

```
P1: while (x1>0) { x1 = x1 - y1; }

P2: while (x2>0) { x2 = x2 - y2; }
```

- **Question**: do they agree on termination?
  - In general, no. eg when $x_1<0$ initially and $x_2>0 \wedge y_2=0$
  - But they do if *Pre*: $x_1=x_2 \wedge y_1=y_2$
- Not k-safety. Cannot be refuted by finite traces.

Problem 3: TS/TI Generalized Non-interference

(Will discuss later)

Contributions / Outline

- ***k-Safety***: Pose the scheduling problem with *predicate variables*.
- ***Co-termination*** via *well-founded predicate variables*
- ***Generalized N.I.*** (∀∃ properties) via *functional predicate variables (for prophecy)*

- A more expressive CSP language: pfw-CSP (beyond CHCs and pCSP)
- Soundness and completeness of these encodings.
- Solve pfw-CSP with *stratified CEGIS*
- Implementation and evaluation

Predicate Variables

In first order logic, predicate variables are “meta” variables:

$$\forall x, y \,.\, P(x, y)$$

In higher order logic, they are propositional variables, placeholders for expressions in the logic:

$$\forall x, y \,.\, \exists P \,.\, P(x, y)$$

Showing this formula is satisfiable means finding such a P.

What does this have to do with programs?

Predicate Variables

Program:

```
while (x ≥ 0) do
  if ∗ then x := x − 1 else x := x + 1
```

Goal:

Prove that there is a
non-terminating execution.

Predicate Variables

Program:

```
while (x ≥ 0) do
  if ∗ then x := x − 1 else x := x + 1
```

Goal:

Prove that there is a non-terminating execution.

Constraint Satisfaction Problem (CSP):

$$\left(\begin{array}{l} x \geq 0 \Rightarrow I(x), \\ (I(x) \wedge x \geq 0) \Rightarrow (I(x-1) \vee I(x+1)), \\ (I(x) \wedge x < 0) \Rightarrow \bot \end{array} \right)$$

Predicate Variables

Program:

```
while (x ≥ 0) do
  if ∗ then x := x − 1 else x := x + 1
```

Goal:

Prove that there is a non-terminating execution.

Constraint Satisfaction Problem (CSP):

$$\exists I.\left(\begin{array}{l} x \geq 0 \Rightarrow I(x), \\ (I(x) \wedge x \geq 0) \Rightarrow (I(x-1) \vee I(x+1)), \\ (I(x) \wedge x < 0) \Rightarrow \bot \end{array}\right)$$

Predicate Variable

Predicate Constraint Satisfaction

Recent works are able to verify SAT via synthesis of predicate variables.

The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

Probabilistic Inference for Predicate Constraint Satisfaction

Yuki Satake,[1] Hiroshi Unno,[1,2] Hinata Yanagi[1]
[1]University of Tsukuba, [2]RIKEN AIP
{satake, uhiro, hinata}@logic.cs.tsukuba.ac.jp

Abstract

In this paper, we present a novel constraint solving method for a class of predicate Constraint Satisfaction Problems (pCSP) where each constraint is represented by an *arbitrary* clause of first-order predicate logic over predicate variables. The class of pCSP properly subsumes the well-studied class of Constrained Horn Clauses (CHCs) where each constraint is restricted to a *Horn* clause. The class of CHCs has been widely applied to verification of *linear-time* safety properties of programs in different paradigms. In this paper, we show that pCSP further widens the applicability to verification of *branching-time* safety properties of programs that exhibit finitely-branching non-determinism. Solving pCSP (and CHCs) however is challenging because the search space of solutions is often very large (or unbounded), high-dimensional, and non-smooth. To address these challenges, our method naturally combines techniques studied separately in different literatures: counterexample guided inductive synthesis (CEGIS) and probabilistic inference in graphical models. We have implemented the presented method and obtained

This paper studies a generalization of CHCs called predicate Constraint Satisfaction Problems (pCSP) where each constraint is represented by an *arbitrary* (i.e., possibly non-Horn) clause. We show that this generalization further widens the applicability to verification of *branching-time* safety properties of programs that exhibit finitely branching non-determinism. In other words, this paves the way to use pCSP as a common intermediate language for verification of branching-time safety properties in place of CHCs that is limited to a strict subclass (i.e., linear-time safety properties).[1] One of the notable instances of branching-time safety verification is non-termination verification where the goal is to check whether *there is* a non-terminating execution of the given program (Gupta et al. 2008). For example, consider the following program c_{nt}:

```
while (x ≥ 0) do
  if * then x := x − 1 else x := x + 1
```

The program repeatedly and *non-deterministically* (indi-

Problem 1: k-Safety

```
int z1, y1=0;
if (h1) { z1 = 2*x1; } else { z1 = x1; }
while (z1>0) { z1--; y1 = y1+x1; }
if (!h1) { y1 = 2*y1; }
```

```
int z2, y2=0;
if (h2) { z2 = 2*x2; } else { z2 = x2; }
while (z2>0) { z2--; y2 = y2+x2; }
if (!h2) { y2 = 2*y2; }
```

Encoding in pCSP:

$$
\begin{aligned}
\mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \Leftarrow\ & x_1 = x_2 \wedge \\
& y_1 = 0 \wedge (h_1 \wedge z_1 = 2 \times x_1 \vee \neg h_1 \wedge z_1 = x_1) \wedge \\
& y_2 = 0 \wedge (h_2 \wedge z_2 = 2 \times x_2 \vee \neg h_2 \wedge z_2 = x_2) \\
\mathsf{inv}(\widetilde{V}_1', \widetilde{V}_2) \Leftarrow\ & \mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \wedge \mathsf{sch}_{\mathsf{TF}}(\widetilde{V}_1, \widetilde{V}_2) \wedge \\
& (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) \\
\mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2') \Leftarrow\ & \mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \wedge \mathsf{sch}_{\mathsf{FT}}(\widetilde{V}_1, \widetilde{V}_2) \wedge \\
& (z_2 > 0 \wedge z_2' = z_2 - 1 \wedge y_2' = y_2 + x_2 \vee z_2 \leq 0 \wedge z_2' = z_2 \wedge y_2' = y_2) \\
\mathsf{inv}(\widetilde{V}_1', \widetilde{V}_2') \Leftarrow\ & \mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \wedge \mathsf{sch}_{\mathsf{TT}}(\widetilde{V}_1, \widetilde{V}_2) \wedge \\
& (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) \wedge \\
& (z_2 > 0 \wedge z_2' = z_2 - 1 \wedge y_2' = y_2 + x_2 \vee z_2 \leq 0 \wedge z_2' = z_2 \wedge y_2' = y_2) \\
z_1 > 0 \Leftarrow\ & \mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \wedge \mathsf{sch}_{\mathsf{TF}}(\widetilde{V}_1, \widetilde{V}_2) \wedge z_2 > 0 \\
z_2 > 0 \Leftarrow\ & \mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \wedge \mathsf{sch}_{\mathsf{FT}}(\widetilde{V}_1, \widetilde{V}_2) \wedge z_1 > 0 \\
\mathsf{sch}_{\mathsf{TF}}(\widetilde{V}_1, \widetilde{V}_2) \vee \mathsf{sch}_{\mathsf{FT}}(\widetilde{V}_1, \widetilde{V}_2) \vee \mathsf{sch}_{\mathsf{TT}}(\widetilde{V}_1, \widetilde{V}_2) \Leftarrow\ & \mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \wedge (z_1 > 0 \vee z_2 > 0) \\
y_1' = y_2' \Leftarrow\ & \mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \wedge z_1 \leq 0 \wedge z_2 \leq 0 \wedge \\
& (h_1 \wedge y_1' = y_1 \vee \neg h_1 \wedge y_1' = 2 \times y_1) \wedge \\
& (h_2 \wedge y_2' = y_2 \vee \neg h_2 \wedge y_2' = 2 \times y_2)
\end{aligned}
$$

Problem 1: k-Safety

Complete Solution

$$\textbf{sch}_{\{1\}} = \lambda\tilde{V}.\, h_1 \wedge \neg h_2 \wedge z_1 + 1 = 2z_2$$
$$\textbf{sch}_{\{2\}} = \lambda\tilde{V}.\, \neg h_1 \wedge z_2 + 1 = 2z_1$$
$$\textbf{sch}_{\{1,2\}} = \lambda\tilde{V}.\, (h_1 \wedge \neg h_2 \Rightarrow z_1 = 2z_2) \wedge (\neg h_1 \wedge h_2 \wedge z_2 = 2z_1)$$

$$\mathsf{inv}(\widetilde{V}_1, \widetilde{V}_2) \equiv \begin{pmatrix} \neg h_1 \wedge \neg h_2 \wedge \begin{pmatrix} x_1 = x_2 \wedge y_1 = y_2 \wedge z_1 = z_2 \vee \\ x_2 + y_2 = 0 \wedge z_1 < 0 \wedge \\ 1 + 2 \cdot x_2 = 2 \cdot z_2 \wedge 2 + y_1 = y_2 \end{pmatrix} \vee \\ \neg h_1 \wedge h_2 \wedge \begin{pmatrix} x_1 = x_2 \wedge 1 + 2 \cdot x_1 \neq z_2 \wedge 2 \cdot y_1 = y_2 \wedge 2 \cdot z_1 = z_2 \vee \\ x_1 = x_2 \wedge 2 \cdot x_1 \neq z_2 \wedge z_1 \geq 1 \wedge z_2 \geq 1 \wedge \\ x_1 + 2 \cdot y_1 = y_2 \wedge 2 \cdot z_1 = 1 + z_2 \end{pmatrix} \vee \\ h_1 \wedge \neg h_2 \wedge \begin{pmatrix} x_1 = x_2 \wedge y_1 = 2 \cdot y_2 \wedge z_1 = 2 \cdot z_2 \vee \\ x_1 = x_2 \wedge 2 \cdot x_1 \neq z_1 \wedge z_1 \geq 1 \wedge z_2 \geq 1 \wedge \\ y_1 = x_2 + 2 \cdot y_2 \wedge 1 + z_1 = 2 \cdot z_2 \end{pmatrix} \vee \\ h_1 \wedge h_2 \wedge x_1 = x_2 \wedge y_1 = y_2 \wedge z_2 = z_2 \end{pmatrix}$$

Problem 1: k-Safety - Generalized to k-Safety

(1) $Pre(\widetilde{V}) \Rightarrow \mathsf{inv}(\widetilde{V})$

(2) $\mathsf{inv}(\widetilde{V}) \wedge \bigwedge_{i\in[k]} F_i(\widetilde{V}_i) \Rightarrow Post(\widetilde{V})$

(3) For each $A \in \mathcal{P}^+[k]$,

$$\mathsf{inv}(\widetilde{V}) \wedge \mathsf{sch}_A(\widetilde{V}) \wedge \bigwedge_{i\in A} T_i(\widetilde{V}_i, \widetilde{V}_i{}') \wedge \bigwedge_{i\in[k]\setminus A} \widetilde{V}_i = \widetilde{V}_i{}' \Rightarrow \mathsf{inv}(\widetilde{V}')$$

(4) For each $A \in \mathcal{P}^+[k]$, $\mathsf{inv}(\widetilde{x}) \wedge \mathsf{sch}_A(V) \wedge \bigvee_{i\in[k]} \neg F_i(V_i) \Rightarrow \bigvee_{i\in A} \neg F_i(V_i)$

(5) $\mathsf{inv}(\widetilde{V}) \wedge \bigvee_{i\in[k]} \neg F_i(\widetilde{V}_i) \Rightarrow \bigvee_{A\in\mathcal{P}^+[k]} \mathsf{sch}_A(\widetilde{V})$.

Problem 1: k-Safety - Generalized to k-Safety

Invariant must hold initially

(1) $Pre(\widetilde{V}) \Rightarrow \mathsf{inv}(\widetilde{V})$

(2) $\mathsf{inv}(\widetilde{V}) \wedge \bigwedge_{i\in[k]} F_i(\widetilde{V}_i) \Rightarrow Post(\widetilde{V})$

(3) For each $A \in \mathcal{P}^+[k]$,

$$\mathsf{inv}(\widetilde{V}) \wedge \mathsf{sch}_A(\widetilde{V}) \wedge \bigwedge_{i\in A} T_i(\widetilde{V}_i, \widetilde{V}_i{}') \wedge \bigwedge_{i\in[k]\setminus A} \widetilde{V}_i = \widetilde{V}_i{}' \Rightarrow \mathsf{inv}(\widetilde{V}')$$

(4) For each $A \in \mathcal{P}^+[k]$, $\mathsf{inv}(\widetilde{x}) \wedge \mathsf{sch}_A(V) \wedge \bigvee_{i\in[k]} \neg F_i(V_i) \Rightarrow \bigvee_{i\in A} \neg F_i(V_i)$

(5) $\mathsf{inv}(\widetilde{V}) \wedge \bigvee_{i\in[k]} \neg F_i(\widetilde{V}_i) \Rightarrow \bigvee_{A\in\mathcal{P}^+[k]} \mathsf{sch}_A(\widetilde{V})$.

Problem 1: k-Safety - Generalized to k-Safety

Invariant must hold initially

Post condition must hold when all have finished.

(1) $Pre(\widetilde{V}) \Rightarrow \mathsf{inv}(\widetilde{V})$

(2) $\mathsf{inv}(\widetilde{V}) \wedge \bigwedge_{i\in[k]} F_i(\widetilde{V}_i) \Rightarrow Post(\widetilde{V})$

(3) For each $A \in \mathcal{P}^+[k]$,

$$\mathsf{inv}(\widetilde{V}) \wedge \mathsf{sch}_A(\widetilde{V}) \wedge \bigwedge_{i\in A} T_i(\widetilde{V}_i, \widetilde{V}_i{}') \wedge \bigwedge_{i\in[k]\setminus A} \widetilde{V}_i = \widetilde{V}_i{}' \Rightarrow \mathsf{inv}(\widetilde{V}')$$

(4) For each $A \in \mathcal{P}^+[k]$, $\mathsf{inv}(\widetilde{x}) \wedge \mathsf{sch}_A(V) \wedge \bigvee_{i\in[k]} \neg F_i(V_i) \Rightarrow \bigvee_{i\in A} \neg F_i(V_i)$

(5) $\mathsf{inv}(\widetilde{V}) \wedge \bigvee_{i\in[k]} \neg F_i(\widetilde{V}_i) \Rightarrow \bigvee_{A\in\mathcal{P}^+[k]} \mathsf{sch}_A(\widetilde{V})$.

Problem 1: k-Safety - Generalized to k-Safety

Invariant must hold initially

Post condition must hold when all have finished.

The scheduled sub-tuple takes a step

(1) $Pre(\widetilde{V}) \Rightarrow \mathsf{inv}(\widetilde{V})$

(2) $\mathsf{inv}(\widetilde{V}) \wedge \bigwedge_{i\in[k]} F_i(\widetilde{V}_i) \Rightarrow Post(\widetilde{V})$

(3) For each $A \in \mathcal{P}^+[k]$,

$$\mathsf{inv}(\widetilde{V}) \wedge \mathsf{sch}_A(\widetilde{V}) \wedge \bigwedge_{i\in A} T_i(\widetilde{V}_i, \widetilde{V}_i{}') \wedge \bigwedge_{i\in[k]\setminus A} \widetilde{V}_i = \widetilde{V}_i{}' \Rightarrow \mathsf{inv}(\widetilde{V}')$$

(4) For each $A \in \mathcal{P}^+[k]$, $\mathsf{inv}(\widetilde{x}) \wedge \mathsf{sch}_A(V) \wedge \bigvee_{i\in[k]} \neg F_i(V_i) \Rightarrow \bigvee_{i\in A} \neg F_i(V_i)$

(5) $\mathsf{inv}(\widetilde{V}) \wedge \bigvee_{i\in[k]} \neg F_i(\widetilde{V}_i) \Rightarrow \bigvee_{A\in\mathcal{P}^+[k]} \mathsf{sch}_A(\widetilde{V})$.

Problem 1: k-Safety - Generalized to k-Safety

Invariant must hold initially

Post condition must hold when all have finished.

(1) $Pre(\widetilde{V}) \Rightarrow \mathsf{inv}(\widetilde{V})$

(2) $\mathsf{inv}(\widetilde{V}) \wedge \bigwedge_{i\in[k]} F_i(\widetilde{V}_i) \Rightarrow Post(\widetilde{V})$

The scheduled sub-tuple takes a step

(3) For each $A \in \mathcal{P}^+[k]$,

$$\mathsf{inv}(\widetilde{V}) \wedge \mathsf{sch}_A(\widetilde{V}) \wedge \bigwedge_{i\in A} T_i(\widetilde{V}_i, \widetilde{V}_i{}') \wedge \bigwedge_{i\in[k]\setminus A} \widetilde{V}_i = \widetilde{V}_i{}' \Rightarrow \mathsf{inv}(\widetilde{V}')$$

(4) For each $A \in \mathcal{P}^+[k]$, $\mathsf{inv}(\widetilde{x}) \wedge \mathsf{sch}_A(V) \wedge \bigvee_{i\in[k]} \neg F_i(V_i) \Rightarrow \bigvee_{i\in A} \neg F_i(V_i)$

(5) $\mathsf{inv}(\widetilde{V}) \wedge \bigvee_{i\in[k]} \neg F_i(\widetilde{V}_i) \Rightarrow \bigvee_{A\in\mathcal{P}^+[k]} \mathsf{sch}_A(\widetilde{V})$.

If A is scheduled, and there is some unfinished copy, then must be an unfinished copy in one in A.

Problem 1: k-Safety - Generalized to k-Safety

Invariant must hold initially

Post condition must hold when all have finished.

(1) $Pre(\widetilde{V}) \Rightarrow \mathsf{inv}(\widetilde{V})$

(2) $\mathsf{inv}(\widetilde{V}) \wedge \bigwedge_{i \in [k]} F_i(\widetilde{V_i}) \Rightarrow Post(\widetilde{V})$

The scheduled sub-tuple takes a step

(3) For each $A \in \mathcal{P}^+[k]$,

$$\mathsf{inv}(\widetilde{V}) \wedge \mathsf{sch}_A(\widetilde{V}) \wedge \bigwedge_{i \in A} T_i(\widetilde{V_i}, \widetilde{V_i}') \wedge \bigwedge_{i \in [k] \setminus A} \widetilde{V_i} = \widetilde{V_i}' \Rightarrow \mathsf{inv}(\widetilde{V'})$$

(4) For each $A \in \mathcal{P}^+[k]$, $\mathsf{inv}(\widetilde{x}) \wedge \mathsf{sch}_A(V) \wedge \bigvee_{i \in [k]} \neg F_i(V_i) \Rightarrow \bigvee_{i \in A} \neg F_i(V_i)$

(5) $\mathsf{inv}(\widetilde{V}) \wedge \bigvee_{i \in [k]} \neg F_i(\widetilde{V_i}) \Rightarrow \bigvee_{A \in \mathcal{P}^+[k]} \mathsf{sch}_A(\widetilde{V})$.

If A is scheduled, and there is some unfinished copy, then must be an unfinished copy in one in A.

If there’s an unfinished copy, then there must be some group A that’s scheduled.

Problem 1: k-Safety - Generalized to k-Safety

Invariant must hold initially

Post condition must hold when all have finished.

The scheduled sub-tuple takes a step

(1) $Pre(\widetilde{V}) \Rightarrow \mathsf{inv}(\widetilde{V})$

(2) $\mathsf{inv}(\widetilde{V}) \wedge \bigwedge_{i\in[k]} F_i(\widetilde{V}_i) \Rightarrow Post(\widetilde{V})$

(3) For each $A \in \mathcal{P}^+[k]$,

$$\mathsf{inv}(\widetilde{V}) \wedge \mathsf{sch}_A(\widetilde{V}) \wedge \bigwedge_{i\in A} T_i(\widetilde{V}_i, \widetilde{V}_i{}') \wedge \bigwedge_{i\in[k]\setminus A} \widetilde{V}_i = \widetilde{V}_i{}' \Rightarrow \mathsf{inv}(\widetilde{V}')$$

(4) For each $A \in \mathcal{P}^+[k]$, $\mathsf{inv}(\widetilde{x}) \wedge \mathsf{sch}_A(V) \wedge \bigvee_{i\in[k]} \neg F_i(V_i) \Rightarrow \bigvee_{i\in A} \neg F_i(V_i)$

(5) $\mathsf{inv}(\widetilde{V}) \wedge \bigvee_{i\in[k]} \neg F_i(\widetilde{V}_i) \Rightarrow \bigvee_{A\in\mathcal{P}^+[k]} \mathsf{sch}_A(\widetilde{V})$.

If A is scheduled, and there is some unfinished copy, then must be an unfinished copy in one in A.

If there's an unfinished copy, then there must be some group A that's scheduled.

What is the language of these constraints?

The Constraint Language pCSP [Satake et al, AAAI 2020]

1. Start from a first-order theory T, with formulas and terms:

$$\phi ::= X(\tilde{t}) \mid p(\tilde{t}) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$$

$$t ::= x \mid f(\tilde{t})$$

The Constraint Language pCSP [Satake et al, AAAI 2020]

1. Start from a first-order theory T, with formulas and terms:

Predicate variable

Predicates in the theory

$$\phi ::= X(\tilde{t}) \mid p(\tilde{t}) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$$

$$t ::= x \mid f(\tilde{t})$$

Term variable

Functions in the theory

The Constraint Language pCSP [Satake et al, AAAI 2020]

1. Start from a first-order theory T, with formulas and terms:

Predicate variable

Predicates in the theory

$$\phi ::= X(\tilde{t}) \mid p(\tilde{t}) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$$

$$t ::= x \mid f(\tilde{t})$$

Term variable

Functions in the theory

2. Define pCSP (without wfr and functional predicate vars)

$$\mathscr{C} = \varphi \vee \left(\bigvee_{i=1}^{\ell} X_i(\tilde{t}_i) \right) \vee \left(\bigvee_{i=\ell+1}^{m} \neg X_i(\tilde{t}_i) \right)$$

The Constraint Language pCSP [Satake et al, AAAI 2020]

1. Start from a first-order theory T, with formulas and terms:

Predicate variable

Predicates in the theory

$$\phi ::= X(\tilde{t}) \mid p(\tilde{t}) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$$

$$t ::= x \mid f(\tilde{t})$$

Term variable

Functions in the theory

2. Define pCSP (without wfr and functional predicate vars)

$$\mathscr{C} = \varphi \vee \left(\bigvee_{i=1}^{\ell} X_i(\tilde{t}_i) \right) \vee \left(\bigvee_{i=\ell+1}^{m} \neg X_i(\tilde{t}_i) \right)$$

Portion without predicate variables

The Constraint Language pCSP [Satake et al, AAAI 2020]

1. Start from a first-order theory T, with formulas and terms:

Predicate variable

Predicates in the theory

$$\phi ::= X(\tilde{t}) \mid p(\tilde{t}) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$$

$$t ::= x \mid f(\tilde{t})$$

Term variable

Functions in the theory

2. Define pCSP (without wfr and functional predicate vars)

$$\mathscr{C} = \varphi \vee \left(\bigvee_{i=1}^{\ell} X_i(\tilde{t}_i) \right) \vee \left(\bigvee_{i=\ell+1}^{m} \neg X_i(\tilde{t}_i) \right)$$

Portion without predicate variables

Also define syntactic substitutions and semantic solutions (see the paper).

Generalizing pCSP to pfwCSP

3. Generalize with a kinding function:

$$(\mathscr{C}, \mathscr{K}) \textbf{ with } \mathscr{K} : X \in fpv(\mathscr{C}) \to \{\textbf{Ord}, \textbf{Wfr}, \textbf{Fpv}\}$$

A predicate interpretation ρ is a semantic solution if

$$\rho \vDash FN(X) \textbf{ when } \mathscr{K}(X) = \textbf{Fpv}$$

$$\rho \vDash WF(X) \textbf{ when } \mathscr{K}(X) = \textbf{Wfr}$$

Generalizing pCSP to pfwCSP

3. Generalize with a kinding function:

$$(\mathscr{C}, \mathscr{K}) \textbf{ with } \mathscr{K} : X \in fpv(\mathscr{C}) \to \{\textbf{Ord}, \textbf{Wfr}, \textbf{Fpv}\}$$

4. Semantics via interpretation. First we write:

$\rho \vDash WF(X)$ if interpretation is well-founded
i.e. sort $(\tilde{s}, \tilde{s}) \to \star$ and no inf. sequences)

A predicate interpretation ρ is a semantic solution if

$\rho \vDash FN(X)$ **when** $\mathscr{K}(X) = \textbf{Fpv}$

$\rho \vDash WF(X)$ **when** $\mathscr{K}(X) = \textbf{Wfr}$

Generalizing pCSP to pfwCSP

3. Generalize with a kinding function:

$$(\mathscr{C}, \mathscr{K}) \textbf{ with } \mathscr{K} : X \in fpv(\mathscr{C}) \to \{\textbf{Ord}, \textbf{Wfr}, \textbf{Fpv}\}$$

4. Semantics via interpretation. First we write:

$\rho \vDash WF(X)$ if interpretation is well-founded
i.e. sort $(\tilde{s}, \tilde{s}) \to \star$ and no inf. sequences)

$\rho \vDash FN(X)$ if interpretation is functional
i.e. sort $(\tilde{s}, s) \to \star$ and

$$\rho \vDash \forall \tilde{x} : \tilde{s} \,.\, (\exists y : s \,.\, X(\tilde{x}, y))$$
$$\wedge\, (\forall y_1, y_2 : s \,.\, X(\tilde{x}, y_1) \wedge X(\tilde{x}, y_2) \Rightarrow y_1 = y_2$$

A predicate interpretation ρ is a semantic solution if

$$\rho \vDash FN(X) \textbf{ when } \mathscr{K}(X) = \textbf{Fpv}$$
$$\rho \vDash WF(X) \textbf{ when } \mathscr{K}(X) = \textbf{Wfr}$$

Problem 2: Co-termination

```
P1: while (x1>0) { x1 = x1 - y1; }

P2: while (x2>0) { x2 = x2 - y2; }
```

- **Question**: do they agree on termination?
  - In general, no. eg when $x_1<0$ initially and $x_2>0 \wedge y_2=0$
  - But they do if *Pre*: $x_1=x_2 \wedge y_1=y_2$
- Formally: $\forall \tilde{v}_1, \tilde{v}_2$ **s.t.** $Pre(\tilde{v}_1, \tilde{v}_2)$. **if** $\tilde{v}_1 \rightsquigarrow_1 \tilde{v}_1'$ **then** $\neg(\tilde{v}_2 \rightsquigarrow \bot)$.
- Not k-safety. Cannot be refuted by finite traces.

Problem 2: Co-termination

Specialize to $\tilde{V} = (\tilde{V}_1, \tilde{V}_2)$

Similar to k-safety
(3x) “inv” and “sch" interaction
(4x-5) sch fairness

$$\text{(3a)}\quad \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{FT}}(d, b, \widetilde{V}) \wedge T_2(\widetilde{V_2}, \widetilde{V_2}') \wedge (F_1(\widetilde{V_1}) \vee F_2(\widetilde{V_2}) \Rightarrow \mathsf{inv}(d', b, \widetilde{V_1}, \widetilde{V_2}')$$

$$\text{(3b)}\quad \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{TF}}(d, b, \widetilde{V}) \wedge T_1(\widetilde{V_1}, \widetilde{V_1}') \wedge (F_1(\widetilde{V_1}) \vee F_2(\widetilde{V_2}) \Rightarrow \mathsf{inv}(d', b, \widetilde{V_1}', \widetilde{V_2})$$

$$\text{(3c)}\quad \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{TT}}(d, b, \widetilde{V}) \wedge T_1(\widetilde{V_1}, \widetilde{V_1}') \wedge T_2(\widetilde{V_2}, \widetilde{V_2}') \Rightarrow \mathsf{inv}(d, b, \widetilde{V_1}', \widetilde{V_2}')$$

$$\text{(4a)}\quad \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{FT}}(d, b, \widetilde{V}) \wedge \neg F_1(\widetilde{V_1}) \Rightarrow \neg F_2(\widetilde{V_2})$$

$$\text{(4b)}\quad \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{TF}}(d, b, \widetilde{V}) \wedge \neg F_2(\widetilde{V_2}) \Rightarrow \neg F_1(\widetilde{V_1})$$

$$\text{(5)}\quad \mathsf{inv}(d, b, \widetilde{V}) \wedge (\neg F_1(\widetilde{V_1}) \vee \neg F_2(\widetilde{V_2})) \Rightarrow \bigvee_{a \in \{\mathsf{TT}, \mathsf{FT}, \mathsf{TF}\}} \mathsf{sch}_a(d, b, \widetilde{V})$$

Problem 2: Co-termination

"fnb" is a functional predicate, asserting a bound b

Difference d in steps taken is with b, when neither has terminated.

$$(1)\ \ Pre(\widetilde{V}) \wedge \mathsf{fnb}(\widetilde{V}, b) \Rightarrow \mathsf{inv}(0, b, \widetilde{V})$$

$$(2)\ \ \mathsf{inv}(d, b, V) \wedge \neg F_1(V_1) \wedge \neg F_2(V_2) \Rightarrow (-b \leq d \wedge d \leq b \wedge b \geq 0)$$

$$(3a)\ \ \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{FT}}(d, b, \widetilde{V}) \wedge T_2(\widetilde{V_2}, \widetilde{V_2}') \wedge (F_1(\widetilde{V_1}) \vee F_2(\widetilde{V_2}) \vee d' = d - 1) \Rightarrow \mathsf{inv}(d', b, \widetilde{V_1}, \widetilde{V_2}')$$

$$(3b)\ \ \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{TF}}(d, b, \widetilde{V}) \wedge T_1(\widetilde{V_1}, \widetilde{V_1}') \wedge (F_1(\widetilde{V_1}) \vee F_2(\widetilde{V_2}) \vee d' = d + 1) \Rightarrow \mathsf{inv}(d', b, \widetilde{V_1}', \widetilde{V_2})$$

$$(3c)\ \ \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{TT}}(d, b, \widetilde{V}) \wedge T_1(\widetilde{V_1}, \widetilde{V_1}') \wedge T_2(\widetilde{V_2}, \widetilde{V_2}') \Rightarrow \mathsf{inv}(d, b, \widetilde{V_1}', \widetilde{V_2}')$$

$$(4a)\ \ \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{FT}}(d, b, \widetilde{V}) \wedge \neg F_1(\widetilde{V_1}) \Rightarrow \neg F_2(\widetilde{V_2})$$

$$(4b)\ \ \mathsf{inv}(d, b, \widetilde{V}) \wedge \mathsf{sch}_{\mathsf{TF}}(d, b, \widetilde{V}) \wedge \neg F_2(\widetilde{V_2}) \Rightarrow \neg F_1(\widetilde{V_1})$$

$$(5)\ \ \mathsf{inv}(d, b, \widetilde{V}) \wedge (\neg F_1(\widetilde{V_1}) \vee \neg F_2(\widetilde{V_2})) \Rightarrow \bigvee_{a \in \{\mathsf{TT}, \mathsf{FT}, \mathsf{TF}\}} \mathsf{sch}_a(d, b, \widetilde{V})$$

$$(6)\ \ \mathsf{inv}(d, b, \widetilde{V}) \wedge F_1(\widetilde{V_1}) \wedge \neg F_2(\widetilde{V_2}) \wedge T_2(\widetilde{V_2}, \widetilde{V_2}') \Rightarrow \mathsf{wfr}(\widetilde{V_2}, \widetilde{V_2}')$$

If P_1 terminated, then P_1 must eventually terminate.

"wfr" is a well-founded predicate variable

Problem 3: TS/TI Generalized Non-interference

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

Problem 3: TS/TI Generalized Non-interference

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

- **TI-GNI**: if two copies agree on low and one terminates returning v, then the other must also return v or else has a non-terminating execution.

Problem 3: TS/TI Generalized Non-interference

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

- **TI-GNI**: if two copies agree on low and one terminates returning v, then the other must also return v or else has a non-terminating execution.

Problem 3: TS/TI Generalized Non-interference

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

- **TI-GNI**: if two copies agree on low and one terminates returning v, then the other must also return v or else has a non-terminating execution.

$$\forall \tilde{v}_1, \tilde{v}_2 \textbf{ s.t. } \mathit{Pre}(\tilde{v}_1, \tilde{v}_2)\,.\ \textbf{if } \tilde{v}_1 \rightsquigarrow_1 \tilde{v}_1' \textbf{ then}$$
$$(\exists \tilde{v}_2'\,.\,\tilde{v}_2 \rightsquigarrow \tilde{v}_2' \wedge \mathit{Post}(\tilde{v}_1', \tilde{v}_2')) \vee \tilde{v}_2 \rightsquigarrow \bot$$

Problem 3: TS/TI Generalized Non-interference

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

- **TI-GNI:** if two copies agree on low and one terminates returning v, then the other must also return v or else has a non-terminating execution.

$$\forall \tilde{v}_1, \tilde{v}_2 \textbf{ s.t. } Pre(\tilde{v}_1, \tilde{v}_2)\,.\ \textbf{if } \tilde{v}_1 \twoheadrightarrow_1 \tilde{v}_1' \textbf{ then}$$
$$(\exists \tilde{v}_2'\,.\, \tilde{v}_2 \twoheadrightarrow \tilde{v}_2' \wedge Post(\tilde{v}_1', \tilde{v}_2')) \vee \tilde{v}_2 \twoheadrightarrow \bot$$

- **TS-GNI:** if two copies agree on low and one terminates returning v, then the other must also return v.

Problem 3: TS/TI Generalized Non-interference

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

- **TI-GNI:** if two copies agree on low and one terminates returning v, then the other must also return v or else has a non-terminating execution.

$$\forall \tilde{v}_1, \tilde{v}_2 \textbf{ s.t. } Pre(\tilde{v}_1, \tilde{v}_2)\,.\ \textbf{if } \tilde{v}_1 \twoheadrightarrow_1 \tilde{v}_1' \textbf{ then}$$
$$(\exists \tilde{v}_2'\,.\,\tilde{v}_2 \twoheadrightarrow \tilde{v}_2' \wedge Post(\tilde{v}_1', \tilde{v}_2')) \vee \tilde{v}_2 \twoheadrightarrow \bot$$

- **TS-GNI:** if two copies agree on low and one terminates returning v, then the other must also return v.

- For this program with *Pre*: low_1=low_2, TS-GNI does hold. (whatever I pick for x in P_1, there's a suitable choice in P_2)

Problem 3: TS/TI Generalized Non-interference

1. Decompose the transition relation:

$$T(\tilde{v}, \tilde{v}') \Leftrightarrow \exists r \,.\, U(r, \tilde{v}, \tilde{v}') \textbf{ and } U(r, \tilde{v}, \tilde{v}') \wedge U(r, \tilde{v}, \tilde{v}'') \Rightarrow \tilde{v}' = \tilde{v}''$$

Make nondet choices explicit.

ification (m6) adds functional predicate variables to express the angelic non-deterministic choices of P_2. The functional predicate variables shift the onus of making the right choices to the solver's task of discovering sufficient assignments to them. Importantly, the functional predicate takes the prophecy variables as parameter

Prophecy variables introduced

Can take *any* initial value.

(m1) The parameters representing the inputs and outputs of P_1 is extended with prophecy variables $\widetilde{p}$ where $|\widetilde{p}| = |\widetilde{V_1}|$. Accordingly, each occurrence of $\widetilde{V_1}$ is replaced by $\widetilde{p}, \widetilde{V_1}$, and each occurrence of $\widetilde{V_1}'$ is replaced by $\widetilde{p}', \widetilde{V_1}'$.

(m2) Pre is replaced by Pre' which is defined by $Pre'(\widetilde{p}, \widetilde{V_1}, \widetilde{V_2}) \Leftrightarrow Pre(\widetilde{V_1}, \widetilde{V_2})$, *i.e.*, the prophecy values are unconstrained in the precondition.

Propagate

(m3) F_1 is replaced by F_1' defined by $F_1'(\widetilde{p}, \widetilde{V_1}) \Leftrightarrow F_1(\widetilde{V_1})$.

(m4) T_1 is replaced by T_1' defined by $T_1'(\widetilde{p}, \widetilde{V_1}, \widetilde{p}', \widetilde{V_1}') \Leftrightarrow T_1(\widetilde{V_1}, \widetilde{V_1}') \wedge \widetilde{p} = \widetilde{p}'$.

(m5) $Post$ is replaced by $Post'$ defined by $Post'(\widetilde{p}, \widetilde{V_1}, \widetilde{V_2}) \Leftrightarrow (\widetilde{p} = \widetilde{V_1} \Rightarrow Post(\widetilde{V_1}, \widetilde{V_2}))$, *i.e.*, if the prophecy was correct then the original post condition must hold.

(m6) Each occurrence of $T_2(\widetilde{V_2}, \widetilde{V_2}')$ is replaced by $\mathsf{fnr}(\widetilde{p}, \widetilde{V_2}, r) \wedge U_2(r, \widetilde{V_2}, \widetilde{V_2}')$ where fnr is a functional predicate variable.

Output consistent

Choices of P2 based on functional predicate variable fnr

We now sho
augment t
predicate variab
deterministic ch
demonic side (*i.*

Problem 3: TS/TI Generalized Non-interference

Inferred Solution

$$\mathsf{fnb}(x_1, y_1, x_2, u_2, b) \equiv b = 0$$

$$\mathsf{fnr}(p, h_2, l_2, x_2) \equiv x_2 = p$$

$$\mathsf{inv}(d, b, \widetilde{V}_1, \widetilde{V}_2) \equiv \left(\begin{array}{l} \neg h_1 \wedge \neg h_2 \wedge d = 0 \wedge b = 0 \wedge b_2 \wedge x_2 \geq l_2 \wedge l_1 = l_2 \vee \\ \neg h_1 \wedge h_2 \wedge d = 0 \wedge b \geq 0 \wedge x_1 \geq l_1 \wedge p = x_2 \wedge l_1 = l_2 \vee \\ h_1 \wedge \neg h_2 \wedge \left(\begin{array}{l} d = 0 \wedge b = 0 \wedge b_2 \wedge l_1 = l_2 \vee \\ l_1 = x_2 \wedge p = x_1 \wedge x_1 \geq l_1 \wedge d \geq 1 + b \wedge \\ b = l_2 \wedge 1 + 2 \cdot x_1 + 2 \cdot x_2 = 0 \end{array} \right) \vee \\ h_1 \wedge h_2 \wedge \left(\begin{array}{l} d = 0 \wedge b = 0 \wedge b_1 \wedge l_1 = l_2 \wedge x_2 = p \vee \\ \neg b_1 \wedge x_1 \geq l_1 \wedge p = x_2 \wedge l_1 = l_2 \end{array} \right) \end{array} \right)$$

$$\mathsf{wfr}(x, h, l, x', h', l') \equiv \neg h \wedge l - x \geq 0 \wedge l - x > l' - x' \vee h \wedge x \geq 0 \wedge x > x'$$

where $\widetilde{V}_1 = p, b_1, x_1, h_1, l_1$ and $\widetilde{V}_2 = b_2, x_2, h_2, l_2$.

Problem 3: TS/TI Generalized Non-interference

Functional predicate for bound

Functional predicate for prophecy

Inferred Solution

$$\mathsf{fnb}(x_1, g_1, x_2, u_2, b) \equiv b = 0$$

$$\mathsf{fnr}(p, h_2, l_2, x_2) \equiv x_2 = p$$

$$\mathsf{inv}(d, b, \widetilde{V}_1, \widetilde{V}_2) \equiv \left(\begin{array}{l} \neg h_1 \wedge \neg h_2 \wedge d = 0 \wedge b = 0 \wedge b_2 \wedge x_2 \geq l_2 \wedge l_1 = l_2 \vee \\ \neg h_1 \wedge h_2 \wedge d = 0 \wedge b \geq 0 \wedge x_1 \geq l_1 \wedge p = x_2 \wedge l_1 = l_2 \vee \\ h_1 \wedge \neg h_2 \wedge \left(\begin{array}{l} d = 0 \wedge b = 0 \wedge b_2 \wedge l_1 = l_2 \vee \\ l_1 = x_2 \wedge p = x_1 \wedge x_1 \geq l_1 \wedge d \geq 1 + b \wedge \\ b = l_2 \wedge 1 + 2 \cdot x_1 + 2 \cdot x_2 = 0 \end{array}\right) \vee \\ h_1 \wedge h_2 \wedge \left(\begin{array}{l} d = 0 \wedge b = 0 \wedge b_1 \wedge l_1 = l_2 \wedge x_2 = p \vee \\ \neg b_1 \wedge x_1 \geq l_1 \wedge p = x_2 \wedge l_1 = l_2 \end{array}\right) \end{array}\right)$$

$$\mathsf{wfr}(x, h, l, x', h', l') \equiv \neg h \wedge l - x \geq 0 \wedge l - x > l' - x' \vee h \wedge x \geq 0 \wedge x > x'$$

where $\widetilde{V}_1 = p, b_1, x_1, h_1, l_1$ and $\widetilde{V}_2 = b_2, x_2, h_2, l_2$.

Relational invariant

Well-founded relation predicate

Problem 3: TS/TI Generalized Non-interference

Thm. Encoding is sound and complete.

- By determining the angelic choices via solutions to the functional predicate variables and reducing the argument to k-safety and co-termination.
- Completeness by synthesizing sufficient anglic choice functions from program executions.

TODO - show sound and complete for other problems.

Solving pfwCSP problems

- Iterates two phases until convergence.
- Build a sequence σ of candidate solutions and a sequence $\mathscr{E}$ of example instances.
- Example instance $\mathscr{E}^{(i)}$ is an instantiation of term variables and provides a counterexample to $\sigma^{(i-1)}$. Start with $\mathscr{E}^{(0)} = \varnothing$.
- ***Synthesis phase***: Check if $(\mathscr{E}^{(i)}, \mathscr{K})$ is unsat. If so, done. Else synthesized a new solution $\sigma^{(i)}$.
- ***Validation phase***: Use an SMT solver to see if $\sigma^{(i)}$ is a solution.
    - If not, for each unsat clause, obtain substitution θ_c such that $\not\models \theta_c(\sigma^{(i)}(c))$. Update examples: $\mathscr{E}^{(i+1)} = \mathscr{E}^{(i)} \cup \{\theta_c(c) \mid \not\models \sigma^{(i)}(c)\}$.

Synthesis is via stratified templates:

Number of conjuncts, disjuncts

Bound sum of absolute values of coefficients

Stratified Template Family for Ordinary Predicate Variables:

$$T^{\bullet}_X(nd, nc, ac, ad) \triangleq \lambda(x_1, \ldots, x_{\mathrm{ar}(X)}).\bigvee_{i=1}^{nd}\bigwedge_{j=1}^{nc} c_{i,j,0} + \sum_{k=1}^{\mathrm{ar}(X)} c_{i,j,k}\cdot x_k \geq 0$$

$$\phi^{\bullet}_X(nd, nc, ac, ad) \triangleq \bigwedge_{i=1}^{nd}\bigwedge_{j=1}^{nc}(\sum_{k=1}^{\mathrm{ar}(X)}|c_{i,j,k}| \leq ac) \wedge |c_{i,j,0}| \leq ad$$

Stratified Template Family for Well-Founded Predicate Variables:

$$T^{\Downarrow}_X(np, nl, nc, rc, rd, dc, dd) \triangleq \lambda(\widetilde{x}, \widetilde{y}).\bigwedge_{i=1}^{np}\bigwedge_{k=1}^{nl} r_{i,k}(\widetilde{x}) \geq 0 \wedge (\bigvee_{i=1}^{np} D_i(\widetilde{x}))\wedge (\bigvee_{j=1}^{np} D_j(\widetilde{y})) \wedge (\bigvee_{i=1}^{np} D_i(\widetilde{x}) \wedge \bigwedge_{j=1}^{np}(D_j(\widetilde{y}) \Rightarrow DEC_{i,j}(\widetilde{x}, \widetilde{y})))$$

$$\phi^{\Downarrow}_X(np, nl, nc, rc, rd, dc, dd) \triangleq \bigwedge_{i=1}^{np}\bigwedge_{k=1}^{nl}(\sum_{\ell=1}^{\mathrm{ar}(X)/2}|c_{i,k,\ell}| \leq rc) \wedge |c_{i,k,0}| \leq rd \wedge \bigwedge_{i=1}^{np}\bigwedge_{k=1}^{nc}(\sum_{\ell=1}^{\mathrm{ar}(X)/2}|c'_{i,k,\ell}| \leq dc) \wedge |c'_{i,k,0}| \leq dd$$

$$DEC_{i,j}(\widetilde{x}, \widetilde{y}) \triangleq \bigvee_{k=1}^{nl}(r_{i,k}(\widetilde{x}) > r_{j,k}(\widetilde{y}) \wedge \bigwedge_{\ell=1}^{k-1} r_{i,\ell}(\widetilde{x}) \geq r_{j,\ell}(\widetilde{y}))$$

$$r_{i,k}(\widetilde{x}) \triangleq c_{i,k,0} + \sum_{\ell=1}^{\mathrm{ar}(X)/2} c_{i,k,\ell}\cdot x_\ell \qquad D_i(\widetilde{x}) \triangleq \bigwedge_{k=1}^{nc} c'_{i,k,0} + \sum_{\ell=1}^{\mathrm{ar}(X)/2} c'_{i,k,\ell}\cdot x_\ell \geq 0$$

Stratified Template Family for Functional Predicate Variables:

$$T^{\lambda}_X(nd, nc, dc, dd, ec, ed) \triangleq \lambda(\widetilde{x}, r).r = \textsf{if } D_1(\widetilde{x}) \textsf{ then } e_1(\widetilde{x}) \textsf{ else if } D_2(\widetilde{x}) \textsf{ then } e_2(\widetilde{x}) \cdots \textsf{ else if } D_{nd-1}(\widetilde{x}) \textsf{ then } e_{nd-1}(\widetilde{x}) \textsf{ else } e_{nd}(\widetilde{x})$$

$$\phi^{\lambda}_X(nd, nc, ec, ed, dc, dd) \triangleq \bigwedge_{i=1}^{nd}(\sum_{j=1}^{\mathrm{ar}(X)-1}|c_{i,j}| \leq ec) \wedge |c_{i,0}| \leq ed \wedge \bigwedge_{i=1}^{nd-1}\bigwedge_{j=1}^{nc}(\sum_{k=1}^{\mathrm{ar}(X)-1}|c'_{i,j,k}| \leq dc) \wedge |c'_{i,j,0}| \leq dd$$

$$e_i(\widetilde{x}) \triangleq c_{i,0} + \sum_{j=1}^{\mathrm{ar}(X)-1} c_{i,j}\cdot x_j \qquad D_i(\widetilde{x}) \triangleq \bigwedge_{j=1}^{nc} c'_{i,j,0} + \sum_{k=1}^{\mathrm{ar}(X)-1} c'_{i,j,k}\cdot x_k \geq 0$$

Synthesis is via stratified templates:

Number of conjuncts, disjuncts

Bound sum of absolute values of coefficients

Piecewise-defined lexicographic affine ranking function

Stratified Template Family for Ordinary Predicate Variables:

$$T^{\bullet}_X(nd, nc, ac, ad) \triangleq \lambda(x_1, \ldots, x_{\mathrm{ar}(X)}).\ \bigvee_{i=1}^{nd} \bigwedge_{j=1}^{nc} c_{i,j,0} + \sum_{k=1}^{\mathrm{ar}(X)} c_{i,j,k} \cdot x_k \geq 0$$

$$\phi^{\bullet}_X(nd, nc, ac, ad) \triangleq \bigwedge_{i=1}^{nd} \bigwedge_{j=1}^{nc} \left(\sum_{k=1}^{\mathrm{ar}(X)} |c_{i,j,k}| \leq ac\right) \wedge |c_{i,j,0}| \leq ad$$

Stratified Template Family for Well-Founded Predicate Variables:

$$T^{\Downarrow}_X(np, nl, nc, rc, rd, dc, dd) \triangleq \lambda(\widetilde{x}, \widetilde{y}).\ \bigwedge_{i=1}^{np} \bigwedge_{k=1}^{nl} r_{i,k}(\widetilde{x}) \geq 0 \wedge \left(\bigvee_{i=1}^{np} D_i(\widetilde{x})\right) \wedge \left(\bigvee_{j=1}^{np} D_j(\widetilde{y})\right) \wedge \left(\bigvee_{i=1}^{np} D_i(\widetilde{x}) \wedge \bigwedge_{j=1}^{np} (D_j(\widetilde{y}) \Rightarrow DEC_{i,j}(\widetilde{x}, \widetilde{y}))\right)$$

$$\phi^{\Downarrow}_X(np, nl, nc, rc, rd, dc, dd) \triangleq \bigwedge_{i=1}^{np} \bigwedge_{k=1}^{nl} \left(\sum_{\ell=1}^{\mathrm{ar}(X)/2} |c_{i,k,\ell}| \leq rc\right) \wedge |c_{i,k,0}| \leq rd \wedge \bigwedge_{i=1}^{np} \bigwedge_{k=1}^{nc} \left(\sum_{\ell=1}^{\mathrm{ar}(X)/2} |c'_{i,k,\ell}| \leq dc\right) \wedge |c'_{i,k,0}| \leq dd$$

$$DEC_{i,j}(\widetilde{x}, \widetilde{y}) \triangleq \bigvee_{k=1}^{nl} \left(r_{i,k}(\widetilde{x}) > r_{j,k}(\widetilde{y}) \wedge \bigwedge_{\ell=1}^{k-1} r_{i,\ell}(\widetilde{x}) \geq r_{j,\ell}(\widetilde{y})\right)$$

$$r_{i,k}(\widetilde{x}) \triangleq c_{i,k,0} + \sum_{\ell=1}^{\mathrm{ar}(X)/2} c_{i,k,\ell} \cdot x_\ell \qquad D_i(\widetilde{x}) \triangleq \bigwedge_{k=1}^{nc} c'_{i,k,0} + \sum_{\ell=1}^{\mathrm{ar}(X)/2} c'_{i,k,\ell} \cdot x_\ell \geq 0$$

Stratified Template Family for Functional Predicate Variables:

$$T^{\lambda}_X(nd, nc, dc, dd, ec, ed) \triangleq \lambda(\widetilde{x}, r).r = \textsf{if } D_1(\widetilde{x}) \textsf{ then } e_1(\widetilde{x}) \textsf{ else if } D_2(\widetilde{x}) \textsf{ then } e_2(\widetilde{x}) \cdots \textsf{ else if } D_{nd-1}(\widetilde{x}) \textsf{ then } e_{nd-1}(\widetilde{x}) \textsf{ else } e_{nd}(\widetilde{x})$$

$$\phi^{\lambda}_X(nd, nc, ec, ed, dc, dd) \triangleq \bigwedge_{i=1}^{nd} \left(\sum_{j=1}^{\mathrm{ar}(X)-1} |c_{i,j}| \leq ec\right) \wedge |c_{i,0}| \leq ed \wedge \bigwedge_{i=1}^{nd-1} \bigwedge_{j=1}^{nc} \left(\sum_{k=1}^{\mathrm{ar}(X)-1} |c'_{i,j,k}| \leq dc\right) \wedge |c'_{i,j,0}| \leq dd$$

$$e_i(\widetilde{x}) \triangleq c_{i,0} + \sum_{j=1}^{\mathrm{ar}(X)-1} c_{i,j} \cdot x_j \qquad D_i(\widetilde{x}) \triangleq \bigwedge_{j=1}^{nc} c'_{i,j,0} + \sum_{k=1}^{\mathrm{ar}(X)-1} c'_{i,j,k} \cdot x_k \geq 0$$

Implementation and Evaluation

- CEGIS iterations
- Some required small hints for invariants (could be done with other tools)
- Otherwise, all *fully automatic*.

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

Implementation and Evaluation

- CEGIS iterations
- Some required small hints for invariants (could be done with other tools)
- Otherwise, all *fully automatic*.

k-Safety

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

Implementation and Evaluation

- CEGIS iterations
- Some required small hints for invariants (could be done with other tools)
- Otherwise, all *fully automatic*.

k-Safety

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

Beyond *k*-Safety

Thank you!