Quantitative Relational Synthesis With Semantic Preference Objectives

Presented By Sumit Lahiri[1]

[1]PRAISE Group, CSE Dept. IIT Kanpur

M.Tech Defense

Relational Properties

A property that relates multiple executions of (the same or different) program(s) is referred to as a *relational property*.

A property that relates multiple executions of (the same or different) program(s) is referred to as a *relational property*.

Hoare Triple.

$$\{\mathsf{Pre}(\vec{x})\}\ \mathcal{P}\ \{\mathsf{Post}(y)\}$$

A property that relates multiple executions of (the same or different) program(s) is referred to as a *relational property*.

Hoare Triple.

$$\{\mathsf{Pre}(\vec{x})\}\ \mathcal{P}\ \{\mathsf{Post}(y)\}$$

Relational Property as a Hoare Triple.

$$\{\mathsf{Pre}(\vec{x_1},\ \vec{x_2})\}\ \mathcal{P}_1, \mathcal{P}_2\ \{\mathsf{Post}(y_1,\ y_2)\}$$

A property that relates multiple executions of (the same or different) program(s) is referred to as a *relational property*.

Hoare Triple.

$$\{\mathsf{Pre}(\vec{x})\}\ \mathcal{P}\ \{\mathsf{Post}(y)\}$$

Relational Property as a Hoare Triple.

$$\{\mathsf{Pre}(\vec{x_1},\ \vec{x_2})\}\ \mathcal{P}_1, \mathcal{P}_2\ \{\mathsf{Post}(y_1,\ y_2)\}$$

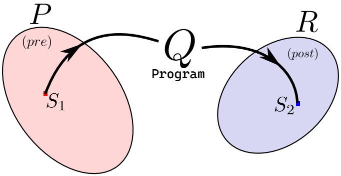
When both programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are the same programs, i.e $\mathcal{P}$, *relational properties* become **hyper-properties**!.

What is program sketching?

A partial program (referred to as a *sketch*), which leaves out certain $\boxed{holes}$ for the synthesizer to fill such that the completed program satisfies the required specification.

A partial program (referred to as a *sketch*), which leaves out certain $\boxed{holes}$ for the synthesizer to fill such that the completed program satisfies the required specification.

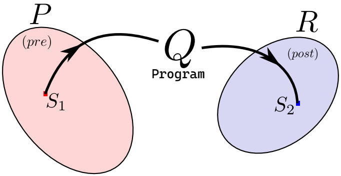
Specification as a **Hoare Triple**,
$$\{P\}Q\{R\}$$



What is program sketching?

A partial program (referred to as a *sketch*), which leaves out certain $\boxed{holes}$ for the synthesizer to fill such that the completed program satisfies the required specification.

Specification as a **Hoare Triple**,
$\{P\}Q\{R\}$




```
int P(int n){
    // PRE : assume(n > 1);
    int i = 0, x = 0;
    while(i < n){
        i = i + 1;
        x = ·
    }
    // POST : assert(x > 2 * n);
    return x;
}
```

Given partial programs $\mathcal{P}_1^{[\cdot]}$ and $\mathcal{P}_2^{[\cdot]}$, find completion $\mathcal{E}$, where $\mathcal{E}.H$ and $\mathcal{E}.G$ respective completions of $\mathcal{P}_1^{[\cdot]}$ and $\mathcal{P}_2^{[\cdot]}$ that satisfy a specification.

Given partial programs $\mathcal{P}_1^{[\cdot]}$ and $\mathcal{P}_2^{[\cdot]}$, find completion $\mathcal{E}$, where $\mathcal{E}.H$ and $\mathcal{E}.G$ respective completions of $\mathcal{P}_1^{[\cdot]}$ and $\mathcal{P}_2^{[\cdot]}$ that satisfy a specification.

Relational Synthesis

$$\exists \mathcal{E}.\ \{\mathsf{Pre}(\vec{x_1}, \vec{x_2})\}\ \mathcal{P}_1^{[\mathcal{E}.H]}, \mathcal{P}_2^{[\mathcal{E}.G]}\ \{\mathsf{Post}(y_1, y_2)\}$$

Program equivalence requires that, any two executions of a pair of programs, $\mathcal{P}_1$ and $\mathcal{P}_2$ on same the same input $\vec{x}$, must yield the same outputs.

Program equivalence requires that, any two executions of a pair of programs, $\mathcal{P}_1$ and $\mathcal{P}_2$ on same the same input $\vec{x}$, must yield the same outputs.

A Verification Problem

$$\forall \vec{x}.\ \mathcal{P}_1(\vec{x}) = \mathcal{P}_2(\vec{x})$$

Given a reference program $\mathcal{P}_1$ and a *partial* program $\mathcal{P}_2$ that has a hole $\boxed{\cdot}$, we are interested in *completing* the partial program by synthesizing an expression to fill the hole such that the two programs are rendered semantically equivalent.

Given a reference program $\mathcal{P}_1$ and a *partial* program $\mathcal{P}_2$ that has a hole $\boxed{\cdot}$, we are interested in *completing* the partial program by synthesizing an expression to fill the hole such that the two programs are rendered semantically equivalent.

Formal Definition

$$\exists \mathcal{E} \in \mathcal{L}(\mathcal{G}).\ \forall \vec{x}.\ \mathcal{P}_1(\vec{x}) = \mathcal{P}_2^{[\mathcal{E}]}(\vec{x})$$

```
int P1(int n){
    assume(n > 1);
    int i = 0, ans = 0;
    while(i < (n - 1)){
        i = i + 1;
        ans = ans + (5 * i) + 1;
    }
    return ans + 1;
}
```

(a) Program 1

```
int P2[·](int n){
    assume(n > 1);
    int x = 0, y = 0, z = n;
    while(z ≠ 0){
        z = z - 1;
        x = [·];
        y = y + 1;
    }
    return x + y;
}
```

(b) Program 2 (with hole $\boxed{\cdot}$)

Sketching for Program Equivalence

```
int P_1(int n){
    assume(n > 1);
    int i = 0, ans = 0;
    while(i < (n - 1)){
        i = i + 1;
        ans = ans + (5 * i) + 1;
    }
    return ans + 1;
}
```

(a) Program 1

```
int P_2^[·](int n){
    assume(n > 1);
    int x = 0, y = 0, z = n;
    while(z ≠ 0){
        z = z - 1;
        x = [·];
        y = y + 1;
    }
    return x + y;
}
```

(b) Program 2 (with hole $\boxed{\cdot}$)

Post condition for sketching.

$$\exists \mathcal{E} \in \mathcal{L}(\mathcal{G}).\ \forall \vec{x}.\ \mathcal{P}_1(\vec{x}) = \mathcal{P}_2^{[\mathcal{E}]}(\vec{x})$$

```
int P1(int n){
    assume(n > 1);
    int i = 0, ans = 0;
    while(i < (n - 1)){
        i = i + 1;
        ans = ans + (5 * i) + 1;
    }
    return ans + 1;
}
```

(a) Program 1

```
int P2[·](int n){
    assume(n > 1);
    int x = 0, y = 0, z = n;
    while(z ≠ 0){
        z = z - 1;
        x = [·];
        y = y + 1;
    }
    return x + y;
}
```

(b) Program 2 (with hole $\boxed{\cdot}$)

Post condition for sketching.

$$\exists \mathcal{E} \in \mathcal{L}(\mathcal{G}).\ \forall \vec{x}.\ \mathcal{P}_1(\vec{x}) = \mathcal{P}_2^{[\mathcal{E}]}(\vec{x})$$

$$assert(ans + 1 = x + y)$$

```
int P_1(int n){
    assume(n > 1);
    int i = 0, ans = 0;
    while(i < (n - 1)){
        i = i + 1;
        ans = ans + (5 * i) + 1;
    }
    return ans + 1;
}
```

(a) Program 1

```
int P_2^[.](int n){
    assume(n > 1);
    int x = 0, y = 0, z = n;
    while(z ≠ 0){
        z = z - 1;
        x = [x + 6 · y − n];
        y = y + 1;
    }
    return x + y;
}
```

(b) Program 2 (with hole $\boxed{\cdot}$)

Program equivalence posed as a **relational** property.

Program equivalence posed as a **relational** property.

Relational Synthesis for Program Equivalence

$$\exists \mathcal{E}.\ \{\vec{x_1} = \vec{x_2}\}\ \ \mathcal{P}_1^{[\mathcal{E}.H]}, \mathcal{P}_2^{[\mathcal{E}.G]}\ \ \{\mathcal{P}_1^{[\mathcal{E}.H]}(\vec{x_1}) = \mathcal{P}_2^{[\mathcal{E}.G]}(\vec{x_2})\}$$

Strict non-interference requires that program executions with the same public inputs $\vec{x_1} = \vec{x_2}$, irrespective of the secret inputs s_1 and s_2, must have identical responses; i.e., the program does not reveal any information about the secret input.

$$\exists \mathcal{E}.\ \{\vec{x_1} = \vec{x_2}\}\ \mathcal{P}^{[\mathcal{E}]}\ \ \{\mathcal{P}^{[\mathcal{E}]}(s_1, \vec{x_1}) = \mathcal{P}^{[\mathcal{E}]}(s_2, \vec{x_2})\}$$

Weak Equivalence

$$\exists \mathcal{E}.\ \{\vec{x_1} = \vec{x_2}\}\ \ \mathcal{P}_1^{[\mathcal{E}.H]}, \mathcal{P}_2^{[\mathcal{E}.G]}\ \{|\ \mathcal{P}_1^{[\mathcal{E}.H]}(\vec{x_1}) - \mathcal{P}_2^{[\mathcal{E}.G]}(\vec{x_2})\ | \leq c\}$$

Weak Non-Interference

$$\exists \mathcal{E}.\ \{\vec{x_1} = \vec{x_2}\}\ \mathcal{P}^{[\mathcal{E}]}\ \ \{||\mathcal{P}^{[\mathcal{E}]}(s_1, \vec{x_1}) - \mathcal{P}^{[\mathcal{E}]}(s_2, \vec{x_2})|| \leq c\}$$

Robustness requires that small changes in the inputs must not lead to large difference in the responses of the program. In this case, if the inputs are within a distance d_1, then the desired completion must not change the response of the program by more than a distance that is defined by a function over program inputs $\vec{x_1}$ and $\vec{x_2}$.

$$\exists \mathcal{E}.\ \{||\vec{x_1} - \vec{x_2}|| \leq d_1\}\ \mathcal{P}^{[\mathcal{E}]}\ \ \{||\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}) - \mathcal{P}^{[\mathcal{E}]}(\vec{x_2})|| \leq f(\vec{x_1}, \vec{x_2})\}$$

Group Fairness requires that for two individuals, one from a majority population ($s_1 = 1$) and another from the minority population ($s_2 = 0$), the decision of the program on a favorable decision (like hiring on a job) must not be disadvantageous to the individual from the minority population. That is, the program must not use the *sensitive attribute* (s) to be unfair to the minority population. The response is not necessarily Boolean; it may be a number that indicates the *suitability* of the candidate for the position (higher is better).

$$\exists \mathcal{E}.\ \{s_1 \leq s_2 \wedge \vec{x_2} \sqsubseteq \vec{x_1}\}\ \mathcal{P}^{[\mathcal{E}]}\ \ \{\mathcal{P}^{[\mathcal{E}]}(\vec{x_2}, s_2) \leq \mathcal{P}^{[\mathcal{E}]}(\vec{x_1}, s_1)\}$$

Monotonicity is a hyper-property that requires that for any two executions of the program, if the inputs are ordered, so must be the outputs. The completion would then need to satisfy the following:

$$\exists \mathcal{E}.\ \{\vec{x_1} \sqsubseteq \vec{x_2}\}\ \mathcal{P}^{[\mathcal{E}]}\ \ \{\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}) \sqsubseteq \mathcal{P}^{[\mathcal{E}]}(\vec{x_2})\}$$

Relational property with (semantic) quantitative objectives.

All Gammas!

Monotonicity, Robustness: A preference on a completion could be the one that minimizes the distance between any two responses of the program.

$$\Gamma(\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}), \mathcal{P}^{[\mathcal{E}]}(\vec{x_2})) \triangleq ||\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}) - \mathcal{P}^{[\mathcal{E}]}(\vec{x_2})||$$

Weak Non-Interference, Weak Equivalence: A preference on a completion could be the one that minimizes the distance between any two responses of the program.

$$\Gamma(\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}), \mathcal{P}^{[\mathcal{E}]}(\vec{x_2})) \triangleq ||\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}) - \mathcal{P}^{[\mathcal{E}]}(\vec{x_2})||$$

Group Fairness: One may design many preference metrics over completions. One metric could be to prefer completions where the deviation in responses between candidates of two populations is small for similar candidates.

$$\begin{aligned}\Gamma(\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}, s_1), \mathcal{P}^{[\mathcal{E}]}(\vec{x_2}, s_2)) \triangleq &\text{ if } (s_1 < s_2 \wedge \vec{x_1} \sim \vec{x_2}) \\ &\text{then } ||\mathcal{P}^{[\mathcal{E}]}(\vec{x_1}, y_1) - \mathcal{P}^{[\mathcal{E}]}(\vec{x_2}, y_2)|| \text{ else } 0\end{aligned}$$

```
int 𝒫^[·] (int a, int b){
    assume((0 < a) && (a < b));
    while (a < b) {
        c = c + [·];
        a = a + 1;
    }
    return c;
}
```

Figure: Program Sketch for Monotonicity

Monotonicity Example

```
int P^[·](int a1, int b1, int a2, int b2){
    assume((0 < a1) && (a1 < b1));
    assume((0 < a2) && (a2 < b2));
    int c1 = 0, c2 = 0;
    while ( (a1 < b1) || (a2 < b2) ) {
        if ( (a1 < b1) ) {
            c1 = c1 + [·];
            a1 = a1 + 1;
        }
        if ( (a2 < b2) ) {
            c2 = c2 + [·];
            a1 = a1 + 1;
        }
    }
    return c1, c2;
}
```

Figure: Product program for Monotonicity from Fig. 3.

Run Through: Monotonicity

Run Through: Monotonicity

Tool Architecture

Machine configuration, Benchmark Sources, Domain-specific Language.

Bench	Property	Time(s)
b26	**Strict Equivalence**	4
b10	**Strict Equivalence**	3
b18	**Strict Equivalence**	2
b16	**Strict Equivalence**	1
b21	**Strict Equivalence**	3
b27	**Strict Equivalence**	4
b04	**Strict Equivalence**	3
b34	**Strict Equivalence**	10
b05	**Strict Equivalence**	7
nonintf01	**Strict Non-Interference**	7
nonintf02	**Strict Non-Interference**	8
nonintf05	**Strict Non-Interference**	6

Bench	Property	Time(s)	Best?
mono01	**Monotonicity**	329	✔
mono02	**Monotonicity**	311	✔
mono02	**Monotonicity**	310	✔
weak01	**Weak Equivalence**	210	✔
weak02	**Weak Equivalence**	198	✔
weak03	**Weak Equivalence**	128	✔
weak04	**Weak Equivalence**	168	✔
robust01	**Robustness**	95	✔
robust02	**Robustness**	102	✔
fair01	**Group Fairness**	82	✔
nonintf03	**Weak Non-Interference**	70	✔
nonintf04	**Weak Non-Interference**	75	✔

Conclusion & Future Directions

Thank You!