Quantitative Relational Synthesis With Semantic Preference Objectives

Sumit Lahiri

PRAISE Group, CSE Dept. IIT Kanpur

M.Tech Defense

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.

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

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.

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

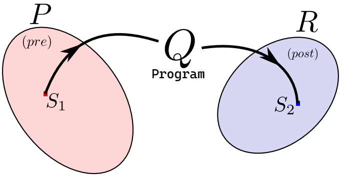
When both programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are the same programs, i.e $\mathcal{P}$, *relational properties* become **hyper-properties**.

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**,

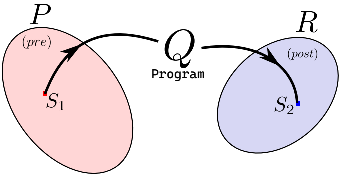
$$\{P\}Q\{R\}$$



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.

Lifting Program Sketching to Relational Sketching

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 $\boxed{\cdot}$)

```
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.

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

```
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 completion)

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

Given public inputs, $(\vec{x_1}, \vec{x_2})$ and secret inputs (s_1 , s_2), program $(\mathcal{P})$ does not reveal any information about the secret inputs.

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

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

Small changes in the inputs must not lead to large difference in the responses of the program, $\mathcal{P}$.

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

Program, $\mathcal{P}$ must not use the *sensitive attribute* (s) to be unfair to the minority population.

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

Monotonicity

For any two executions of the program, $\mathcal{P}$, if the inputs are ordered, so must be the outputs.

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

Quantitative objectives on the synthesis problem that defines the *preference* among the feasible completions, $\mathcal{E}$.

Quantitative objectives on the synthesis problem that defines the *preference* among the feasible completions, $\mathcal{E}$.

Quantitative Objective, Γ

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

Quantitative objectives on the synthesis problem that defines the *preference* among the feasible completions, $\mathcal{E}$.

Quantitative Objective, Γ

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

The most preferred completion can be found by **minimizing** the quantitative objective.

The ordering relation $\mathcal{E}$ is a partial order over the completions, defined by the relation, $\sqsubseteq$, and the *strict* ordering $\sqsubset$

The ordering relation $\mathcal{E}$ is a partial order over the completions, defined by the relation, $\sqsubseteq$, and the *strict* ordering $\sqsubset$

Dominance

A completion $\mathcal{E}'$ dominates a completion $\mathcal{E}$ if the value of Γ score with $\mathcal{E}'$ is smaller or equals to $\mathcal{E}$ across all input pairs $\langle \vec{x_1}, \vec{x_2} \rangle$

The ordering relation $\mathcal{E}$ is a partial order over the completions, defined by the relation, $\sqsubseteq$, and the *strict* ordering $\sqsubset$

Dominance

A completion $\mathcal{E}'$ dominates a completion $\mathcal{E}$ if the value of Γ score with $\mathcal{E}'$ is smaller or equals to $\mathcal{E}$ across all input pairs $\langle \vec{x_1}, \vec{x_2} \rangle$

Strict Dominance

Additionally requires the existence of at least one input-pair where the Γ score of $\mathcal{E}'$ is strictly smaller than that of $\mathcal{E}$

Monotonicity, Robustness, Weak Non-Interference, Weak Equivalence

Completions that minimize the distance between any two responses of the program.

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

Monotonicity, Robustness, Weak Non-Interference, Weak Equivalence

Completions that minimize the distance between any two responses of the program.

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

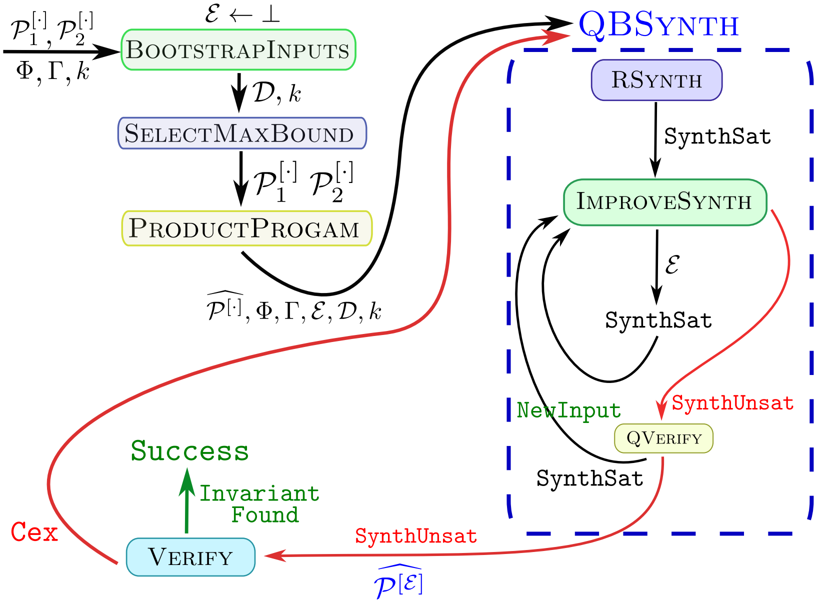
Group Fairness

Prefer completions where the deviation in responses between candidates of two populations is small for similar candidates.

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

Monotonicity Example

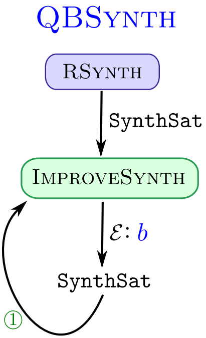
```
// PRE: (a2 < a1 ∧ b2 > b1)
int P^[·] (int a, int b){
    assume((0 < a) && (a < b));
    while (a < b) {
        c = c + [·];
        a = a + 1;
    }
    return c;
}
// POST: (c2 > c1)
```


$\mathcal{E} \leftarrow \perp$
$\mathcal{P}_1^{[\cdot]}, \mathcal{P}_2^{[\cdot]}$
Φ, Γ, k
BootstrapInputs
$\mathcal{D}, k$
SelectMaxBound
$\mathcal{P}_1^{[\cdot]}$ $\mathcal{P}_2^{[\cdot]}$
ProductProgam
$\widehat{\mathcal{P}^{[\cdot]}}, \Phi, \Gamma, \mathcal{E}, \mathcal{D}, k$
QBSynth
RSynth
SynthSat
ImproveSynth
$\mathcal{E}$
SynthSat
NewInput
SynthUnsat
QVerify
SynthSat
Success
Invariant Found
Cex
Verify
SynthUnsat
$\widehat{\mathcal{P}^{[\mathcal{E}]}}$

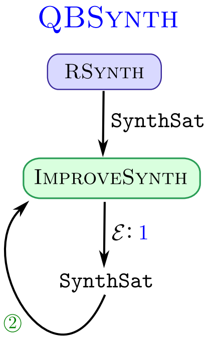
Monotonicity Example: Product Program

```
int P̂[·](int a1, int b1, int a2, int b2){

    assume((0 < a1) && (a1 < b1));
    assume((0 < a2) && (a2 < b2));
    assume((a2 < a1) && (b2 > b1));

    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;
}
```

```
// PRE: (a2 < a1 ∧ b2 > b1)
int P[·] (int a, int b){
    assume((0 < a) && (a < b));
    while (a < b) {
        c = c + [b];
        a = a + 1;
    }
    return c;
}
// POST: (c2 > c1)
```

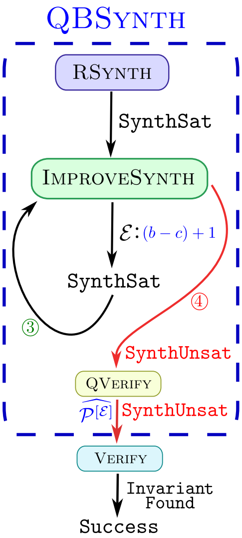


```
// PRE: (a2 < a1 ∧ b2 > b1)
int P[.] (int a, int b){
    assume((0 < a) && (a < b));
    while (a < b) {
        c = c + [1];
        a = a + 1;
    }
    return c;
}
// POST: (c2 > c1)
```



Monotonicity Example: Third Completion

```
// PRE: (a₂ < a₁ ∧ b₂ > b₁)
int P^[·] (int a, int b){
    assume((0 < a) && (a < b));
    while (a < b) {
        c = c + [(b - c) + 1];
        a = a + 1;
    }
    return c;
}
// POST: (c₂ > c₁)
```



Experiments

RQ1. Relational synthesis **without** quantitative objectives

RQ2. Relational synthesis **with** quantitative objectives

Grammar to describe the expressions to be instantiated for the holes in the program sketches; $\langle \texttt{var} \rangle$ and $\langle \texttt{num} \rangle$ refer to program variables and numbers respectively.

Grammar to describe the expressions to be instantiated for the holes in the program sketches; $\langle \texttt{var} \rangle$ and $\langle \texttt{num} \rangle$ refer to program variables and numbers respectively.

Domain Specific Language

$$E ::= E + E \mid E - E \mid E * E \mid \langle \texttt{var} \rangle \mid \langle \texttt{num} \rangle$$

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	✔

New Synthesis Problem!

Relational synthesis with semantic quantitative objectives—and designed an algorithm to solve the same.

New Synthesis Problem!

Relational synthesis with semantic quantitative objectives—and designed an algorithm to solve the same.

Conclusion

- QRS instantiates this algorithm and solves non-trivial problems in a reasonable time.
- Our solutions are optimal only for a k-bounded setting, however, we do achieve the globally optimal completions (empirically) for all benchmarks in our suite.

New Synthesis Problem!

Relational synthesis with semantic quantitative objectives—and designed an algorithm to solve the same.

Conclusion

- QRS instantiates this algorithm and solves non-trivial problems in a reasonable time.
- Our solutions are optimal only for a k-bounded setting, however, we do achieve the globally optimal completions (empirically) for all benchmarks in our suite.

Future Directions

- Synthesis for global optimality.
- Synthesis problems for those beyond 2-safety.

Thank You!

Questions?