Exploiting Synchrony and Symmetry in Relational Verification

Lauren Pick
Relational Verification
Relational Verification

Given:

- \( k (k > 1) \) programs (renamed so that they have independent sets of variables)

- a \textit{relational} specification (relating the variables) over the \( k \) programs

Prove that the relational specification holds for the programs
Example: Equivalence Checking

Given programs $P_1$, $P_2$, that respectively have inputs $x_1$, $x_2$ and outputs $y_1$, $y_2$, prove $x_1 = x_2 \Rightarrow y_1 = y_2$.

*Note: bold-faced variables are vectors*
Hyperproperty Verification

- A hyperproperty is a relational property over $k$ copies of the same program.

- The hyperproperty verification problem is the relational verification problem where all $k$ programs are copies of the same program.

- E.g. noninterference, monotonicity, transitivity.
Example: Noninterference

Security property for programs where variables have security types \{\text{low, high}\}

Given two copies of the same program $P_1, P_2$, that respectively have inputs $(l_{x1} : \text{low}, h_{x1} : \text{high}), (l_{x2} : \text{low}, h_{x2} : \text{high})$ and outputs $(l_{y1} : \text{low}, h_{y1} : \text{high}), (l_{y2} : \text{low}, h_{y2} : \text{high})$, prove $l_{x1} = l_{x2} \implies l_{y1} = l_{y2}$.
Example: Monotonicity

Given two copies of the same program $P_1$, $P_2$, that respectively have inputs $x_1$, $x_2$, and outputs $y_1$, $y_2$, prove $x_1 \leq x_2 \Rightarrow y_1 \leq y_2$. 
Composition

Sequential: \{pre\} P_1 ; \ldots ; P_k \{post\}

- Pros: Can easily apply standard verification techniques
- Cons: Inflexible, can result in more difficult verification problems

[Barthe et al., 2004]
[Terauchi and Aiken, 2005]
Composition

Parallel: \{pre\} P_1 \parallel \ldots \parallel P_k \{post\}

- **Pros**: Flexibility can let us pick easier verification subproblems

- **Cons**: Need to come up with new techniques

[Barthe et al., 2004]
[Terauchi and Aiken, 2005]
Synchrony and Symmetry

Two new techniques: Synchrony, Symmetry

Let’s consider the challenges that motivate them....
Challenge 1: Loops
Challenge 1: Loops

\{ x_1 < x_2 \land i_1 = i_2 \land x_1 > 0 \land i_1 > 0 \} 

L1: \begin{array}{l}
\text{while } (i_1 < 10) \{ x_1 \times = i_1; i_1++; \} \\
\} 
\end{array}

L2: \begin{array}{l}
\text{while } (i_2 < 10) \{ x_2 \times = i_2; i_2++; \} \\
\} 
\end{array}

\{ x_1 < x_2 \land i_1 = i_2 \land x_1 > 0 \land i_1 > 0 \} 

Nonlinear Invariants:

L1: \( x_1 = x_{1_{\text{init}}} \times i_1! / i_{1_{\text{init}}} \land \ldots \)

L2: \( x_2 = x_{2_{\text{init}}} \times i_2! / i_{2_{\text{init}}} \land \ldots \)
Challenge 1: Loops

\{ x_1 < x_2 \land i_1 = i_2 \land x_1 > 0 \land i_1 > 0 \}

\text{while (i_1 < 10) \{ x_1 *= i_1; i_1++; \}}

\parallel

\text{while (i_2 < 10) \{ x_2 *= i_2; i_2++; \}}

\{ x_1 < x_2 \land i_1 = i_2 \land x_1 > 0 \land i_1 > 0 \}

Consider the loops in parallel instead.
Challenge 1: Loops

\[
\{ x1 < x2 \land i1 = i2 \land x1 > 0 \land i1 > 0 \}
\]

while (i1 < 10 && i2 < 10) {
\[
x1 *= i1; i1++; x2 *= i2; i2++;
\]
}

\[
\{ x1 < x2 \land i1 = i2 \land x1 > 0 \land i1 > 0 \}
\]

(One) Relational Invariant: \( x1 < x2 \land i1 = i2 \land x1 > 0 \land i1 > 0 \)
Relational Verification

- Relating (i.e. synchronizing) intermediate points in programs to get \textit{intermediate relational specifications} can result in easier verification problems.

- In particular, synchronizing structurally similar parts of the different programs can yield simpler relational specifications.

\begin{itemize}
  \item say that \(s\) are structurally similar
  \item Relating (i.e. \textit{synchronizing}) intermediate points in programs to get \textit{intermediate relational specifications} can result in easier verification problems.
  \item In particular, synchronizing structurally similar parts of the different programs can yield simpler relational specifications.
\end{itemize}

\[\text{pre} \rightarrow \text{post}\]

[Barthe et al., 2011]
[Sousa and Dillig, 2016]
[De Angelis et al., 2016] and more
Lockstep Loops

Loops that iterate the same number of times are able to be executed in lockstep

[Barthe et al., 2011]
[Sousa and Dillig, 2016]
Challenge 1: Loops

Handling each loop individually can require the generation of potentially complicated loop invariants.

How can we maximize the number of loops over which we can compute simpler relational invariants?
Synchrony

Partition a set of loops into \textit{maximal} sets of loops that can be executed in lockstep
We assume we are given a relational invariant $I$. }

Note: You can use any of several existing techniques for invariant generation. The implementation (described later) uses a guess-and-check invariant generator.
Synchrony

When can we execute a set of loops in lockstep?

If any loop has terminated, all loops must have terminated.

\[ I \land (\neg c_1 \lor \neg c_2 \lor \ldots \lor \neg c_k) \Rightarrow (\neg c_1 \land \neg c_2 \land \ldots \land \neg c_k) \]

(check)

[Sousa and Dillig, 2016]
Maximal Lockstep Loop Detection

(check) \neg(I \land \neg c_1 \lor \neg c_2 \lor \ldots \lor \neg c_k) \Rightarrow \neg c_1 \land \neg c_2 \land \ldots \land \neg c_k

• If unsatisfiable, all loops can be executed in lockstep. (Done!)

• If satisfiable, then what?

(partition) • Use model to partition the set of loops into those that have terminated (\neg c_i holds in the model) and those that have not (c_i holds in the model)

(recurse) • Recurse on the two sets....
Maximal Lockstep Loop Example

\[ \neg (I \land (\neg c_1 \lor \neg c_2 \lor \ldots \lor \neg c_5) \Rightarrow (\neg c_1 \land \neg c_2 \land \ldots \land \neg c_5)) \]

(check) SAT: \( c_1, c_2, c_3, \neg c_4, \) and \( c_5 \) hold in model

(partition)
Maximal Lockstep Loop

Example

\[ \neg(I \land (\neg c_1 \lor \neg c_2 \lor \neg c_3 \lor \neg c_5) \Rightarrow (\neg c_1 \land \neg c_2 \land \neg c_3 \land \neg c_5)) \]

(check) SAT: \(c_1, \neg c_2, c_3, \neg c_5\) hold in model

(partition)
Summary: Maximal Lockstep Loop Detection

Step 1. Check if current set can be executed in lockstep

Step 2. Partition according to model (if necessary)

Step 3. Recurse
Challenge 2: Redundancy
Challenge 2: Redundancy

\[ \{ x_1 \neq x_2 \} \]

\[
\begin{align*}
\text{if } (x_1 > y_1) \text{ then } P_1 \text{ else } Q_1 \\
\| \\
\text{if } (x_2 > y_2) \text{ then } P_2 \text{ else } Q_2
\end{align*}
\]

\[ \{ x_1 \neq x_2 \} \]
Challenge 2: Redundancy

\[
\{ x_1 \neq x_2 \}
\]

if \((x_1 > y_1)\) then \(P_1\) else \(Q_1 \parallel \) if \((x_2 > y_2)\) then \(P_2\) else \(Q_2\)

\[
\{ x_1 \neq x_2 \}
\]

\[
\{ x_1 \neq x_2 \land x_1 > y_1 \land x_2 > y_2 \}
\]

\(P_1 \parallel P_2\)

\[
\{ x_1 \neq x_2 \land x_1 \leq y_1 \land x_2 > y_2 \}
\]

\(Q_1 \parallel P_2\)

\[
\{ x_1 \neq x_2 \land x_1 > y_1 \land x_2 \leq y_2 \}
\]

\(P_1 \parallel Q_2\)

\[
\{ x_1 \neq x_2 \land x_1 \leq y_1 \land x_2 \leq y_2 \}
\]

\(Q_1 \parallel Q_2\)
Challenge 2: Redundancy

Maybe for the given relational specification, $P_{1,2} P_{k,3}$ and $P_{1,3} P_{k,2}$ are symmetric over indices.

How can we identify and use symmetries in programs and in relational specifications to avoid solving redundant verification problems?
Symmetric Relational Verification Problems (RVPs)

If you permute indices, you get the same problem.

\[
\{ x_1 \neq x_2 \} \quad \text{if} \ (x_1 > y_1) \text{ then } P_1 \text{ else } Q_1 \]

\[
\{ x_2 \neq x_1 \} \quad \text{if} \ (x_2 > y_2) \text{ then } P_2 \text{ else } Q_2 \]

\[
\{ x_1 \neq x_2 \} \quad \text{if} \ (x_2 > y_2) \text{ then } P_2 \text{ else } Q_2 \]

\[
\{ x_2 \neq x_1 \} \quad \text{if} \ (x_1 > y_1) \text{ then } P_1 \text{ else } Q_1 \]

Need a permutation \( \pi \) of indices that is a symmetry of the formulas (pre- and postconditions) and of the programs.

(can e.g. check if at same program point for hyperproperties)
Leveraging Symmetry to Reduce Redundancies

- Find symmetries in formulas (permutation $\pi$)
- Find symmetric RVPs (make sure programs are symmetric, i.e. $\pi$ is a symmetry of the programs also)
- Prune (via symmetry-breaking, lifted from SAT)
Leveraging Symmetry to Reduce Redundancies

- **Find symmetries in formulas** (permutation $\pi$)
- Find symmetric RVPs (make sure programs are symmetric, i.e. $\pi$ is a symmetry of the programs also)
- Prune (via symmetry-breaking, lifted from SAT)
Finding Symmetries of a Formula

• Prior work for SAT formulas: based on finding automorphisms of a colored graph

• Our work: Lift SAT techniques to first-order theories (with equality, linear integer arithmetic)
Example: Finding Symmetries of a Formula

Step 1. Canonicalize

\[ \phi = x_1 \leq x_2 \land x_3 \leq x_4 \]

to CNF

\[ \phi' = ((x_1 < x_2) \lor (x_1 = x_2)) \land ((x_3 < x_4) \lor (x_3 = x_4)) \]

[Aloul et al., 2006]
[Crawford et al., 2005]
Example: Finding Symmetries of a Formula

Step 2. Create colored graph from AST

\[ \phi' = ((x_1 < x_2) \lor (x_1 = x_2)) \land ((x_3 < x_4) \lor (x_3 = x_4)) \]
Example: Finding Symmetries of a Formula

Step 2. Create colored graph from ASTs

Clauses: \( \{(x_1 < x_2) \lor (x_1 = x_2), (x_3 < x_4) \lor (x_3 = x_4)\} \)
Example: Finding Symmetries of a Formula

Step 2. Create graph from ASTs

$$x_1 x_2 = \lor (x_1, L)(x_2, R)$$

$$x_3 x_4 = \lor (x_3, L)(x_4, R)$$
Example: Finding Symmetries of a Formula

Step 2. Create graph from ASTs
Example: Finding Symmetries of a Formula

Step 2. Create graph from ASTs
Example: Finding Symmetries of a Formula

Step 2. Create graph from ASTs
Example: Finding Symmetries of a Formula

Step 2. Create graph from ASTs

\[
\begin{align*}
\text{Example: Finding Symmetries of a Formula} \\
\text{Step 2. Create graph from ASTs} \\
\end{align*}
\]
Example: Finding Symmetries of a Formula

Step 3. Find graph automorphisms

\[ \pi = \{1 \mapsto 3, 2 \mapsto 4, 3 \mapsto 1, 4 \mapsto 2\} \]

\[ x (x, R) \lor (x, L) < \lor < x (x, R) \]

\[ = x (x, L) \lor (x, R) = x (x, L) \lor (x, R) \]

\[ \text{Id} 1 \text{Id} 2 \text{Id} 3 \text{Id} 4 \]
Example: Finding Symmetries of a Formula

\[ \pi = \{1 \mapsto 3, 2 \mapsto 4, 3 \mapsto 1, 4 \mapsto 2\} \]

\[ \phi = x_1 \leq x_2 \land x_3 \leq x_4 \]

\[ \pi(\phi) = x_3 \leq x_4 \land x_1 \leq x_2 \]

\[ \phi \Leftrightarrow \pi(\phi) \]
Summary: Finding Symmetries of a Formula Automatically

Step 1. Canonicalize

Step 2. Create graph from AST

Step 3. Find graph automorphisms
How to apply symmetry?

- Aligning conditionals
- So far we have considered trying to align loops in a particular way
- We also would like to align conditional statements in order to give us more opportunities to exploit symmetry
Aligning Conditionals: Example

\{ \text{post}(\text{post}(x_1 \neq x_2, R_1), S_2) \} \\
R \text{ if } (x_1 > y_1) \text{ then } P_1 \text{ else } Q_1 \\
\parallel \\
S \text{ if } (x_2 > y_2) \text{ then } P_2 \text{ else } Q_2 \\
\{ x_1 \neq x_2 \}
Instantiation: Hyperproperty Verification
Instantiation: Hyperproperty Verification

Algorithm based on forward analysis where we maintain Hoare triples.

1: procedure VERIFY(pre, Current, Ifs, Loops, post)
2:   while Current ≠ ∅ do
3:     if PROCESSSTATEMENT(pre, P_i, Ifs, Loops, post) = safe then return safe
4:     if Loops ≠ ∅ then HANDLELOOPS(pre, Loops, post)
5:     else if Ifs ≠ ∅ then HANDLEIFS(pre, Ifs, Loops, post)
6:     else return unsafe

Maintain sets of program copies that begin with conditionals (Ifs) and loops (Loops).
Instantiation: Hyperproperty Verification

1: procedure VERIFY\((pre, Current, Ifs, Loops, post)\)
2: while Current \(\neq \emptyset\) do
3: if \(\text{PROCESSSTATEMENT}\((pre, P_i, Ifs, Loops, post)\) = safe\) then return safe
4: if Loops \(\neq \emptyset\) then HANDLELOOPS\((pre, Loops, post)\)
5: else if Ifs \(\neq \emptyset\) then HANDLEIFS\((pre, Ifs, Loops, post)\)
6: else return unsafe

Current  Ifs  Loops

\[
\begin{align*}
P_1,0 & \rightarrow P_1,1 & & \rightarrow P_1,2 & \rightarrow P_1,3 & & \rightarrow P_1,4 \\
P_2,0 & \rightarrow P_2,1 & & \rightarrow P_2,2 & \rightarrow P_2,3 & & \rightarrow P_2,4 \\
P_3,0 & \rightarrow P_3,1 & & \rightarrow P_3,2 & \rightarrow P_3,3 & & \rightarrow P_3,4
\end{align*}
\]
Instantiation: Hyperproperty Verification

1: procedure VERIFY(pre, Current, Ifs, Loops, post)
2:   while Current ≠ ∅ do
3:     if PROCESSSTATEMENT(pre, P_i, Ifs, Loops, post) = safe then return safe
4:     if Loops ≠ ∅ then HANDLELOOPS(pre, Loops, post)
5:     else if Ifs ≠ ∅ then HANDLEIFS(pre, Ifs, Loops, post)
6:     else return unsafe

Current          Ifs          Loops

P1,1  P1,2  P1,3  P1,4  P2,1  P2,2  P2,3  P2,4  P3,1  P3,2  P3,3  P3,4
Instantiation: Hyperproperty Verification

1: procedure Verify(pre, Current, Ifs, Loops, post)
2:   while Current \neq \emptyset do
3:     if ProcessStatement(pre, P_i, Ifs, Loops, post) = safe then return safe
4:     if Loops \neq \emptyset then HANDLELOOPS(pre, Loops, post)
5:     else if Ifs \neq \emptyset then HANDLEIFS(pre, Ifs, Loops, post)
6:     else return unsafe

Current       Ifs       Loops

\begin{center}
\begin{tikzpicture}[node distance=2cm, auto]
  \node (P1) at (0,0) {P_{1,1}};
  \node (P2) at (3,0) {P_{2,1}};
  \node (P3) at (6,0) {P_{3,1}};
  \node (P4) at (0,-3) {P_{1,2}};
  \node (P5) at (3,-3) {P_{2,2}};
  \node (P6) at (6,-3) {P_{3,2}};
  \node (P7) at (0,-6) {P_{1,3}};
  \node (P8) at (3,-6) {P_{2,3}};
  \node (P9) at (6,-6) {P_{3,3}};
  \node (P10) at (0,-9) {P_{1,4}};
  \node (P11) at (3,-9) {P_{2,4}};
  \node (P12) at (6,-9) {P_{3,4}};

  \draw [->] (P1) -- (P4);
  \draw [->] (P1) -- (P10);
  \draw [->] (P2) -- (P4);
  \draw [->] (P2) -- (P11);
  \draw [->] (P3) -- (P6);
  \draw [->] (P3) -- (P12);
  \draw [->] (P4) -- (P7);
  \draw [->] (P4) -- (P8);
  \draw [->] (P5) -- (P8);
  \draw [->] (P5) -- (P9);
  \draw [->] (P6) -- (P9);
  \draw [->] (P6) -- (P10);

  \node at (3,-12) {handle maximally in lockstep};
\end{tikzpicture}
\end{center}
Instantiation: Hyperproperty Verification

1: procedure VERIFY(pre, Current, Ifs, Loops, post)
2:   while Current ≠ ∅ do
3:     if PROCESSSTATEMENT(pre, P_i, Ifs, Loops, post) = safe then return safe
4:     if Loops ≠ ∅ then HANDLELOOPS(pre, Loops, post)
5:     else if Ifs ≠ ∅ then HANDLEIFS(pre, Ifs, Loops, post)
6:     else return unsafe

Current     Ifs     Loops

P1,2

P1,3
P1,4

P2,2

P2,3
P2,4

P3,2

P3,3
P3,4
Instantiation: Hyperproperty Verification

1: `procedure VERIFY(pre, Current, Ifs, Loops, post)`
2: `while Current ≠ ∅ do`
3: `if PROCESSSTATEMENT(pre, P_i, Ifs, Loops, post) = safe then return safe`
4: `if Loops ≠ ∅ then HANDLELOOPS(pre, Loops, post)`
5: `else if Ifs ≠ ∅ then HANDLEIFS(pre, Ifs, Loops, post)`
6: `else return unsafe`

*Current* | *Ifs* | *Loops*
---|---|---
P₁,₂ | | |
P₂,₂ | | |
P₃,₂ | | |

Avoid generating redundant RVPs.
Evaluation

Prototype

• Built on top of Descartes [Sousa and Dillig, 2016]

• Two variants:
  • Syn - uses synchrony
  • Synonym - uses synchrony and symmetry

Benchmarks

• 33 small Stackoverflow Java benchmarks (21-107 LOC) from original Descartes evaluation [Sousa and Dillig, 2016]

• 16 larger, modified Stackoverflow Java benchmarks (62-301 LOC)

All experiments conducted on a MacBook Pro with a 2.7GHz Intel Core i5 processor and 8GB RAM.
public class Match implements Comparator<Match> {
    int score;
    int seq1start;
    int seq2start;

    @Override
    public int compare(Match o1, Match o2) {
        // first compare scores
        if (o1.score > o2.score) return -1; /* higher score for o1 -> o1 */
        if (o1.score < o2.score) return 1; /* higher score for o2 -> o2 */

        // scores are equal, go on with the position
        if ((o1.seq1start + o1.seq2start) < (o2.seq1start+o2.seq2start))
            return -1; /* o1 farther left */
        if ((o1.seq1start + o1.seq2start) > (o2.seq1start+o2.seq2start))
            return 1; /* o2 farther left */

        // they're equally good
        return 0;
    }
}
Results: Small Stackoverflow Benchmarks

P1: ∀x,y. sgn(compare(x,y)) = -sgn(compare(y,x))
Results: Small Stackoverflow Benchmarks

P2: $\forall x, y, z. \ (\text{compare}(x, y) > 0 \land \text{compare}(y, z) > 0) \Rightarrow \text{compare}(x, z) > 0$

Times

HTCs

Syn vs. Descartes

Synonym vs. Descartes
Results: Small Stackoverflow Benchmarks

P3: \( \forall x, y, z. \, \text{compare}(x, y) = 0 \Rightarrow (\text{sgn}((\text{compare}(x, z))) = \text{sgn}((\text{compare}(y, z)))) \)

### Times

| Prop. P3 Times | Prop. P3 Times |
|----------------|----------------|
| Time (s) (SYN) | Time (s) (DESCARTES) |
| 0.01           | 0.01 |
| 0.1            | 1.0 |
| 1.0            | 10.0 |
| 10.0           | 100.0 |

### HTC

| Prop. P3 Hoare Triple Counts | Prop. P3 Hoare Triple Counts |
|------------------------------|------------------------------|
| HTC (SYN)                    | HTC (DESCARTES) |
| 10                           | 10 |
| 100                          | 100 |
| 1,000                        | 1,000 |

Syn
vs.
Descartes

Synonym
vs.
Descartes

57
Results: Modified Benchmarks

P13: $\forall x,y,z.\text{pick}(x,y,z) = \text{pick}(y,x,z)$

Syn vs. Descartes

Synonym vs. Descartes
Results: Modified Benchmarks

P13: $\forall x, y, z. \text{pick}(x, y, z) = \text{pick}(y, x, z)$

Synonym vs. Syn
Results: Modified Benchmarks

P14: $\forall x,y,z. \text{pick}(x,y,z) = \text{pick}(y,x,z) \land \text{pick}(x,y,z) = \text{pick}(z,y,x)$

**Times**

Syn vs. Descartes

Synonym vs. Descartes

**HTCs**

Descartes times out on all examples.
Results: Modified Benchmarks

P14: \( \forall x, y, z. \ pick(x, y, z) = \ pick(y, x, z) \land \ pick(x, y, z) = \ pick(z, y, x) \)

Synonym vs. Syn
Related Work

• Cartesian Hoare Logic and Cartesian Loop Logic for relational verification (most closely related)
  • [Sousa and Dillig, 2016]

• Exploiting synchrony (by constructing [some kind of] product program)
  • [Barthe et al. 2011; Lahiri et al. 2013; Strichman and Veitsman 2016; Felsing et al. 2014; Kiefer et al., 2016; De Angelis et al., 2016; Mordvinov and Fedyukovich, 2017]

• Exploiting symmetry in model checking
  • [Emerson and Sistla, 1993; Clarke et al., 1993; Ip and Dill, 1996; Donaldson et al., 2011]

• Without self-composition
  • [Antonopoulos et al., 2017]
Summary

We have seen approaches to addressing the following two challenges in relational verification:

1. How can we maximize the number of loops over which we can compute simpler relational invariants?

2. How can we identify and use symmetries in programs and relational specifications to avoid solving redundant verification problems?
Extra Slides
Composition

Can use standard verification techniques by applying *composition*. E.g. for equivalence-checking:

\[ \{ x_1 = x_2 \} \ P_1 \ c \ P_2 \ \{ y_1 = y_2 \} \]

where c is a composition operator (e.g. sequential composition or parallel composition)
Challenge: Loops

\{ x_1 < x_2 \land i_1 = i_2 \land i_3 > i_1 \land x_1 > 0 \land i_1 > 0 \}

while (i_1 < 10) \{ x_1 *= i_1; i_1++; \} ||
while (i_2 < 10) \{ x_2 *= i_2; i_2++; \} ||
while (i_3 < 10) \{ x_3 *= i_3; i_3++; \}

\{ x_1 < x_2 \land i_1 = i_2 \land x_1 > 0 \land i_1 > 0 \}

In this case, not all loops can be executed in lockstep, but we still want to execute the first and second loops together.
Symmetric Relational Verification Problems

Two relational verification problem \{pre\} $Ps$ \{post\} and \{pre\} $Ps'$ \{post\} are symmetric under a permutation $\pi$ iff

1. $\pi$ is a symmetry of formula $pre \land post'$

2. for every $P_i \in Ps$ and $P_j \in Ps'$, if $\pi(i) = j$, then $P_i$ and $P_j$ have the same number of inputs and outputs and have logically equivalent encodings for the same set of input variables and output variables
Symmetric Formulas

Let $\mathbf{x}_1, \ldots, \mathbf{x}_k$ be vectors of the same size over disjoint sets of variables.

A symmetry $\pi$ of a formula $F(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is a permutation of set $\{ \mathbf{x}_i | 1 \leq i \leq k \}$ s.t.

$$F(\mathbf{x}_1, \ldots, \mathbf{x}_k) \Leftrightarrow F(\pi(\mathbf{x}_1), \ldots, \pi(\mathbf{x}_k))$$
Symmetry-Breaking

We can construct the following symmetry-breaking predicate (SBPs) for the condition \((x_i > 5)\)

\[
p1 \land (p1 \Rightarrow (((x1 > 5) \Rightarrow (x3 > 5)) \land p2))) \land (p2 \Rightarrow ((x3 > 5) \Rightarrow (x1 > 5)) \Rightarrow ((x2 > 5) \Rightarrow (x4 > 5)))
\]

This is an adaptation of the SBPs constructed for propositional logic in earlier work.

[Aloul et al., 2006]
[Crawford et al., 1996]
Synchrony on Conditionals: Pruning Bonus

if \((x_1 > 0)\) then \(P_1\) else \(Q_1\) \(\parallel\) if \((x_2 > 0)\) then \(P_2\) else \(Q_2\)

\[
\begin{align*}
\{ x_1 = x_2 \} \\
\end{align*}
\]

\[
\begin{align*}
\{ x_1 = x_2 \land x_1 > 0 \land x_2 > 0 \} \\
\{ x_1 \neq x_2 \} \\
\end{align*}
\]

\[
\begin{align*}
\{ x_1 = x_2 \land x_1 \leq 0 \land x_2 > 0 \} \\
\{ x_1 \neq x_2 \} \\
\end{align*}
\]

\[
\begin{align*}
\{ x_1 = x_2 \land x_1 > 0 \land x_2 \leq 0 \} \\
\{ x_1 \neq x_2 \} \\
\end{align*}
\]

\[
\begin{align*}
\{ x_1 = x_2 \land x_1 \leq 0 \land x_2 \leq 0 \} \\
\{ x_1 \neq x_2 \} \\
\end{align*}
\]

\[
\begin{align*}
\{ x_1 = x_2 \land x_1 > 0 \land x_2 \leq 0 \} \\
\{ x_1 \neq x_2 \} \\
\end{align*}
\]

\[
\begin{align*}
\{ x_1 = x_2 \land x_1 \leq 0 \land x_2 \leq 0 \} \\
\{ x_1 \neq x_2 \} \\
\end{align*}
\]