

Introduction to relational verification

Gilles Barthe
MPI-SP, Germany
IMDEA Software Institute, Spain

August 1, 2019

Motivation

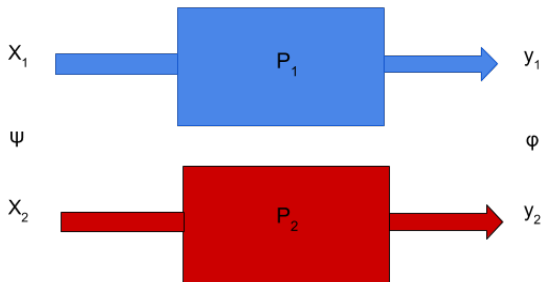
- ▶ Program verification focuses on trace properties, where
 $\text{property} = \text{set of traces}$
- ▶ Examples: safety, correctness
- ▶ Many existing techniques and scalable tools

However

- ▶ Often want to consider relational properties / 2-properties
 $\text{relational property} = \text{set of pairs of traces}$

Today: examples of relational properties and verification methods

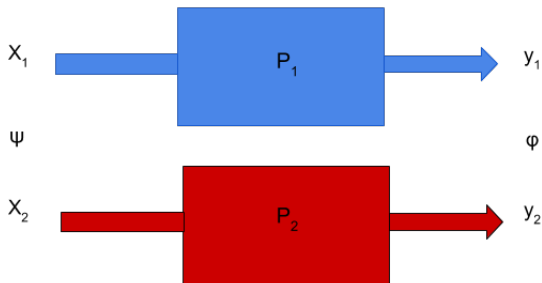
2-properties



Special case: executions starting from pair of states satisfying precondition ψ yield pair of states satisfying postcondition ϕ

$$\{\psi\}c_1 \sim c_2\{\phi\}$$

2-properties: equivalence

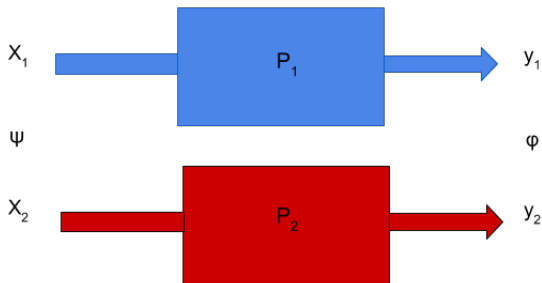


Precondition: $x_1 = x_2$

Postcondition: $y_1 = y_2$

$w_1 \leftarrow 1$; if $w_1 \leq 3$ then $y_1 \leftarrow 2 + x_1$ else $y_1 \leftarrow 0$
 $w_2 \leftarrow 1$; $y_2 \leftarrow 2 + x_2$

2-properties: equivalence wrt precondition

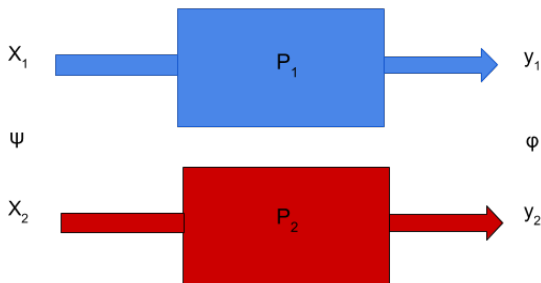


Precondition: $x_1 = x_2 = 3$

Postcondition: $y_1 = y_2$

$w_1 \leftarrow 1$; if $w_1 \leq 3$ then $y_1 \leftarrow 2 + x_1$ else $y_1 \leftarrow 0$
 $w_2 \leftarrow 1$; $y_2 \leftarrow 5$

2-properties: sensitivity / robustness



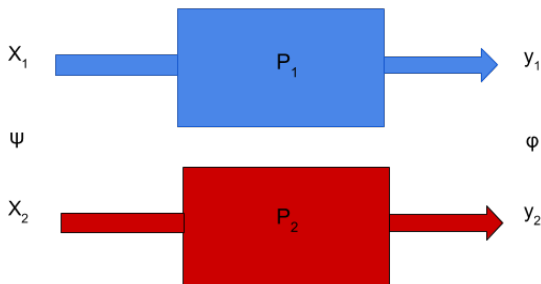
Precondition: $|x_1 - x_2| \leq k \wedge |x'_1 - x'_2| \leq k$

Postcondition: $|y_1 - y_2| \leq k \wedge |y'_1 - y'_2| \leq k$

if $x_1 \leq x'_1$ then $y_1 \leftarrow x_1; y'_1 \leftarrow x'_1$ else $y_1 \leftarrow x'_1; y'_1 \leftarrow x_1$

if $x_2 \leq x'_2$ then $y_2 \leftarrow x_2; y'_2 \leftarrow x'_2$ else $y_2 \leftarrow x'_2; y'_2 \leftarrow x_2$

2-properties: non-interference



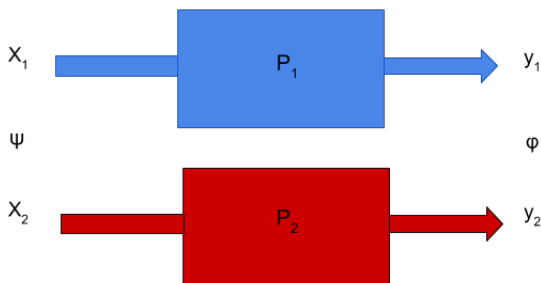
Public variables: x_1, x_2, y_1, y_2

Precondition: $x_1 = x_2$

Postcondition: $y_1 = y_2$

if $x'_1 = 0$ then $y_1 \leftarrow x'_1 \cdot x_1; y'_1 \leftarrow 0$ else $y_1 \leftarrow 0; y'_1 \leftarrow 1$
if $x'_2 = 0$ then $y_2 \leftarrow x'_2 \cdot x_2; y'_2 \leftarrow 0$ else $y_2 \leftarrow 0; y'_2 \leftarrow 1$

2-properties: program counter security



Public variables: x_1, x_2

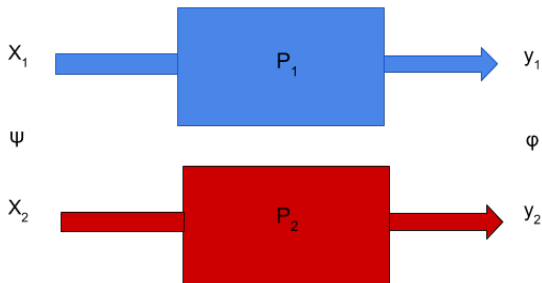
Precondition: $x_1 = x_2$

Postcondition: $O_1 = O_2$

if $x_1 \geq 0$ then $O_1 \leftarrow tt; z_1 \leftarrow x_1$ else $O_1 \leftarrow ff; z_1 \leftarrow 0$

if $x_2 \geq 0$ then $O_1 \leftarrow tt; z_1 \leftarrow x_1$ else $O_1 \leftarrow ff; z_1 \leftarrow 0$

2-properties: relative safety



Precondition: $x_1 = x_2$

Postcondition: $\text{safe}(c_1) \Rightarrow \text{safe}(c_2)$

$$y_1 \leftarrow 1/x_1 \cdot x_1'$$

if $x_1 \neq 0$ then $y_1 \leftarrow 1/x_1 \cdot x_1'$ else $y_1 \leftarrow 0$

Verifying 2-safety

- ▶ Product programs
- ▶ Relational Hoare logic

Product programs

Given programs c_1 and c_2 (with disjoint variables), pre-condition ψ and post-condition ϕ , construct a program c' such that

$$\{\psi\}c_1 \sim c_2\{\phi\} \Leftrightarrow \{\psi\}c'\{\phi\}$$

Product programs

Given programs c_1 and c_2 (with disjoint variables), pre-condition ψ and post-condition ϕ , construct a program c' such that

$$\{\psi\}c_1 \sim c_2\{\phi\} \Leftrightarrow \{\psi\}c'\{\phi\}$$

Self-composition

$$\models \{\psi\}c_1 \sim c_2\{\phi\} \text{ iff } \models \{\psi\}c_1; c_2\{\phi\}$$

- + relatively complete
- cumbersome to use: bookkeeping, complex invariants

Product programs

Given programs c_1 and c_2 (with disjoint variables), pre-condition ψ and post-condition ϕ , construct a program c' such that

$$\{\psi\}c_1 \sim c_2\{\phi\} \Leftrightarrow \{\psi\}c'\{\phi\}$$

Synchronous product

$$\frac{c_1 \times c_2 \rightarrow c \quad c'_1 \times c'_2 \rightarrow c'}{c_1; c'_1 \times c_2; c'_2 \rightarrow c; c'}$$

$$\frac{\text{while } b_1 \text{ do } c_1 \times \text{while } b_2 \text{ do } c_2 \rightarrow \text{assert}(b_1 \Leftrightarrow b_2); \text{while } b_1 \text{ do } (c; \text{assert}(b_1 \Leftrightarrow b_2))}{c_1 \times c_2 \rightarrow c \quad c'_1 \times c'_2 \rightarrow c'}$$

$$\frac{\text{if } b_1 \text{ then } c_1 \text{ then } c'_1 \times \text{if } b_2 \text{ then } c_2 \text{ then } c'_2 \rightarrow \text{assert}(b_1 = b_2); \text{if } b_1 \text{ then } c \text{ then } c'}{\text{if } b_1 \text{ then } c_1 \text{ then } c'_1 \times \text{if } b_2 \text{ then } c_2 \text{ then } c'_2 \rightarrow \text{assert}(b_1 = b_2); \text{if } b_1 \text{ then } c \text{ then } c'}$$

- + easy to use
- incomplete

Product programs

Given programs c_1 and c_2 (with disjoint variables), pre-condition ψ and post-condition ϕ , construct a program c' such that

$$\{\psi\}c_1 \sim c_2\{\phi\} \Leftrightarrow \{\psi\}c'\{\phi\}$$

General products

$$\frac{c_1 \times c_2 \rightarrow c \quad c'_1 \times c'_2 \rightarrow c'}{\text{if } b_1 \text{ then } c_1 \text{ then } c'_1 \times c_2 \rightarrow \text{if } b_1 \text{ then } c \text{ then } c'}$$

- + easy to use
- + relatively complete
- construction is non-deterministic

Relational Hoare Logic (2-sided + structural rules)

$$\frac{\psi \Rightarrow \phi[e_1/x_1, e_2/x_2]}{\{\psi\}x_1 \leftarrow e_1 \sim x_2 \leftarrow e_2\{\phi\}}$$

$$\frac{\{\psi\}c_1 \sim c_2\{\Theta\} \quad \{\Theta\}c'_1 \sim c'_2\{\phi\}}{\{\psi\}c_1; c'_1 \sim c_2; c'_2\{\phi\}}$$

$$\frac{\{\psi \wedge b_1\}c_1 \sim c_2\{\phi\} \quad \{\psi \wedge \neg b_1\}c'_1 \sim c'_2\{\phi\} \quad \psi \Longrightarrow b_1 = b_2}{\{\psi\}\text{if } b_1 \text{ then } c_1 \text{ then } c'_1 \sim \text{if } b_2 \text{ then } c_2 \text{ then } c'_2\{\phi\}}$$

$$\frac{\{\psi \wedge b_1\}c_1 \sim c_2\{\psi\} \quad \psi \Longrightarrow b_1 = b_2}{\{\psi\}\text{while } b_1 \text{ do } c_1 \sim \text{while } b_2 \text{ do } c_2\{\psi \wedge \neg b_1\}}$$

$$\frac{\{\psi\}c_1 \sim c_2\{\phi\} \quad \psi' \Longrightarrow \psi \quad \phi \Longrightarrow \phi'}{\{\psi'\}c_1 \sim c_2\{\phi'\}}$$

$$\frac{\{\psi\}c_1 \sim c_2\{\phi \wedge b\} \quad \{\psi\}c_1 \sim c_2\{\phi \wedge \neg b\}}{\{\psi\}c_1 \sim c_2\{\phi\}}$$

$$\frac{\{\psi\}c_1 \sim c_0\{\phi\} \quad \{\psi'\}c_0 \sim c_2\{\phi'\} \quad c_0 \Downarrow}{\{\psi \circ \psi'\}c_1 \sim c_2\{\phi \circ \phi'\}}$$

+ easy to use

- related inputs must have same control flow

Relational Hoare Logic (1-sided + gen. while rules)

1-sided rule

$$\frac{\{\psi \wedge b_1\}c_1 \sim c_2\{\phi\} \quad \{\psi \wedge \neg b_1\}c'_1 \sim c_2\{\phi\}}{\{\psi\}\text{if } b_1 \text{ then } c_1 \text{ then } c'_1 \sim c_2\{\phi\}}$$
$$\frac{\{\psi \wedge b_1\}c_1 \sim \text{skip}\{\psi\}}{\{\psi\}\text{while } b_1 \text{ do } c_1 \sim \text{skip}\{\psi \wedge \neg b_1\}}$$

General while rule

$$\frac{\{\psi \wedge b_1 \wedge b_2\}c_1^{k_1} \sim c_2^{k_2}\{\psi\} \quad \text{many side-conditions (inc. termination)}}{\{\psi\}\text{while } b_1 \text{ do } c_1 \sim \text{while } b_2 \text{ do } c_2\{\psi \wedge \neg b_1\}}$$

- + programs may have different control-flow
- not syntax-directed

Soundness

- ▶ RHL: if $\vdash \{\psi\}c_1 \sim c_2\{\phi\}$ then for every (m_1, m_2) ,

$$(m_1, m_2) \models \psi \implies (\llbracket c_1 \rrbracket_{m_1}, \llbracket c_2 \rrbracket_{m_2}) \models \phi$$

- ▶ Product program: if $c_1 \times c_2 \rightarrow c$ then for every (m_1, m_2) , either $c, m_1 \uplus m_2 \Downarrow \text{assert_failure}$ or

$$\llbracket c \rrbracket_{m_1 \uplus m_2} = \llbracket c_1 \rrbracket_{m_1} \uplus \llbracket c_2 \rrbracket_{m_2}$$

Full RHL and full product programs are relatively complete
for deterministic computations!

Equivalence

Synchronous product programs and core RHL are equivalent. In fact, we can define product-producing RHL (aka \times RHL):

$$\vdash \{\psi\}c_1 \sim c_2\{\phi\} \rightsquigarrow c$$

We have: $\vdash \{\psi\}c_1 \sim c_2\{\phi\} \rightsquigarrow c$ iff

- ▶ $c_1 \times c_2 \rightarrow c$
- ▶ for every (m_1, m_2) ,

$$(m_1, m_2) \models \psi \implies c, m_1 \uplus m_2 \not\Downarrow \text{assert_failure}$$

- ▶ $\{\psi\}c\{\phi\}$

Characterizing program counter security

Let c_1 and c_2 be renamings of c_0 and c be the synchronized product program of c_0 . I.e.

$$c_1 \times c_2 \rightarrow c$$

The following are equivalent:

- ▶ c_0 is program counter secure
- ▶ $\vdash \{x_1 = x_2\} c_1 \sim c_2 \{true\}$ in core RHL
- ▶ if $m \models x_1 = x_2$, then $c, m \not\Downarrow \text{assert_failure}$

2-properties: more examples

Truthfulness (aka strategyproofness)

Precondition: $x_2^{\text{Alice}} = v \wedge x_1^{\text{Bob}} = x_2^{\text{Bob}}$

Postcondition: $\text{payoff}_1 \leq \text{payoff}_2$

Algorithmic stability

Precondition: $x_2 = x_1 \setminus \{z_1\} \cup \{z_2\}$

Postcondition: $y_1 \sim y_2$

Algorithmic fairness

Sensitive attributes: w_1, w_2 vs non-sensitive attributes: x_1, x_2

Precondition: $x_1 = x_2$

Postcondition: $y_1 = y_2$

Counterfactual explanations

Precondition: $\Delta(x_1, x_2) \leq \epsilon$

Postcondition: $y_1 \neq y_2$

3-properties, 4-properties, etc

Transitivity

$$\{x_1 = u \wedge x'_1 = v \wedge x_2 = v \wedge x'_2 = w \wedge x_3 = u \wedge x'_3 = w\}$$

$$c_1 \sim c_2 \sim c_3$$

$$\{y_1 = \text{true} \wedge y_2 = \text{true} \rightarrow y_3 = \text{true}\}$$

Relative sensitivity

$$\{x_1 = x_3 = a \wedge x_2 = x_4 = b\}$$

$$c_1 \sim c_2 \sim c_3 \sim c_4$$

$$\{|y_1 - y_3| \leq |y_2 - y_4|\}$$

General framework: hyperproperties

hyperproperty = set of sets of traces

Do we need relational verification?

- ▶ No: Prove functional correctness, and then establish relational property
- ▶ Yes: above requires exact verification, invariants often much harder than relational invariants