Relational Verification

16th KeY Symposium
July 2016

Mattias Ulbrich

Title refinement

More precisely:

Effective/Efficient Verification of Relational Properties

Title refinement

More precisely:

Effective/Efficient Verification of
Relational Properties

Preliminaries

- Domain D
  (set of input-/output-values, set of states)

Preliminaries

- Domain D
  (set of input-/output-values, set of states)

- Evaluation $E = D \times D$
  (pair of input and output value)

Preliminaries

- Domain D
  (set of input-/output-values, set of states)
- Evaluation $E = D \times D$
  (pair of input and output value)
- Program $P \subseteq E$

Preliminaries

- Domain D
  (set of input-/output-values, set of states)
- Evaluation $E = D \times D$
  (pair of input and output value)
- Program $P \subseteq E$
- Deterministic Program $P : D \to D, x \mapsto P(x)$
  $P = \{(x, P(x)) \mid x \in D\} \subseteq E$

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of “good” evaluations.

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of "good" evaluations.

Program P satisfies F iff $P \subseteq F$

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of “good” evaluations.

Program P satisfies F iff $P \subseteq F$

Ex.: for $D = \mathbb{Z}$.

$$F = \{(i, o) \mid o \geq 0\}$$

postcondition $\mathit{result} \geq 0$

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of "good" evaluations.

Program P satisfies F iff $P \subseteq F$

Ex.: for $D = \mathbb{Z}$.

$$F = \{(i, o) \mid o \geq 0\}$$

postcondition
$\mathit{result} \geq 0$

Relational Property

$$R \subseteq E \times E$$

the set of "good" evaluation pairs.

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of "good" evaluations.

Program P satisfies F iff $P \subseteq F$

Ex.: for $D = \mathbb{Z}$.

$$F = \{(i, o) \mid o \geq 0\}$$

postcondition *result* ≥ 0

Relational Property

$$R \subseteq E \times E$$

the set of "good" evaluation pairs.

Program P satisfies R iff $P \times P \subseteq R$

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of "good" evaluations.

Program P satisfies F iff $P \subseteq F$

Ex.: for $D = \mathbb{Z}$.

$$F = \{(i, o) \mid o \geq 0\}$$

postcondition $\mathit{result} \geq 0$

Relational Property

$$R \subseteq E \times E$$

the set of "good" evaluation pairs.

Program P satisfies R iff $P \times P \subseteq R$

$$R = \big\{((i_1, o_1), (i_2, o_2)) \mid i_1 = i_2 \Rightarrow o_1 = o_2\big\}$$

P satisfies R iff it is *deterministic*.

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of "good" evaluations.

Program P satisfies F iff $P \subseteq F$

Ex.: for $D = \mathbb{Z}$.

$$F = \{(i, o) \mid o \geq 0\}$$

postcondition $\mathit{result} \geq 0$

Relational Property

$$R \subseteq E \times E$$

the set of "good" evaluation pairs.

Program P satisfies R iff $P \times P \subseteq R$

$$R = \big\{((i_1, o_1), (i_2, o_2)) \bigm| i_1 = i_2 \Rightarrow o_1 = o_2\big\}$$

P satisfies R iff it is *deterministic*.

k-Safety Property

$$R_k \subseteq E^k$$

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of "good" evaluations.

Program P satisfies F iff $P \subseteq F$

Ex.: for $D = \mathbb{Z}$.

$$F = \{(i, o) \mid o \geq 0\}$$

postcondition $\mathit{result} \geq 0$

Relational Property

$$R \subseteq E \times E$$

the set of "good" evaluation pairs.

Program P satisfies R iff $P \times P \subseteq R$

$$R = \{((i_1, o_1), (i_2, o_2)) \mid i_1 = i_2 \Rightarrow o_1 = o_2\}$$

P satisfies R iff it is *deterministic*.

k-Safety Property

$$R_k \subseteq E^k$$

Program P satisfies R_k iff $P^k \subseteq R_k$

Properties and Hyperproperties

Functional Property

$$F \subseteq E$$

the set of "good" evaluations.

Program P satisfies F iff $P \subseteq F$

Ex.: for $D = \mathbb{Z}$.

$F = \{(i, o) \mid o \geq 0\}$

postcondition $result \geq 0$

Relational Property

$$R \subseteq E \times E$$

the set of "good" evaluation pairs.

Program P satisfies R iff $P \times P \subseteq R$

$$R = \{((i_1, o_1), (i_2, o_2)) \mid i_1 = i_2 \Rightarrow o_1 = o_2\}$$

P satisfies R iff it is *deterministic*.

k-Safety Property

$$R_k \subseteq E^k$$

Program P satisfies R_k iff $P^k \subseteq R_k$

Ex.: for $D = \mathbb{Z}$
$Hom_+ \in E^3$

P satisfies Hom_+ iff $P(x+y) = P(x) + P(y)$

Relational Properties in Dynamic Logic

- Let P_1, P_2 be two copies of P

Relational Properties in Dynamic Logic

- Let P_1, P_2 be two copies of P
- that operate on x_1 and x_2

Relational Properties in Dynamic Logic

- Let P_1, P_2 be two copies of P
- that operate on x_1 and x_2
- Proof obligation:
  $[P_1]\,[P_2]\big(\ (old(x_1), x_1, old(x_2), x_2) \in R\ \big)$

Relational Properties in Dynamic Logic

- Let P_1, P_2 be two copies of P
- that operate on x_1 and x_2
- Proof obligation:
  $[P_1][P_2](\ (old(x_1), x_1, old(x_2), x_2) \in R\ )$
- Often:
  $x_1 \sim_{in} x_2 \to [P_1][P_2] x_1 \sim_{out} x_2$

Instances

- **Non-interference (information flow)**

$$\mathit{low}_1 = \mathit{low}_2 \;\rightarrow\; [P_1][P_2]\; \mathit{low}_1 = \mathit{low}_2$$

Instances

- **Non-interference (information flow)**

$$low_1 = low_2 \;\rightarrow\; [P_1][P_2]\; low_1 = low_2$$

- **Program Equivalence**

$$x_1 = x_2 \;\rightarrow\; [P_1][Q_2]\; r_1 = r_2$$

Instances

- **Non-interference (information flow)**

$$low_1 = low_2 \;\rightarrow\; [P_1][P_2]\; low_1 = low_2$$

- **Program Equivalence**

$$x_1 = x_2 \;\rightarrow\; [P_1][Q_2]\; r_1 = r_2$$

- **Refinement**

$$in_{Abs} \sim in_{Concr} \;\rightarrow\; [C]\langle A\rangle\; res_{Abs} \approx res_{Concr}$$

Instances

- **Non-interference (information flow)**

$$low_1 = low_2 \;\rightarrow\; [P_1][P_2]\; low_1 = low_2$$

- **Program Equivalence**

$$x_1 = x_2 \;\rightarrow\; [P_1][Q_2]\; r_1 = r_2$$

- **Refinement**

$$in_{Abs} \sim in_{Concr} \;\rightarrow\; [C]\langle A\rangle\; res_{Abs} \approx res_{Concr}$$

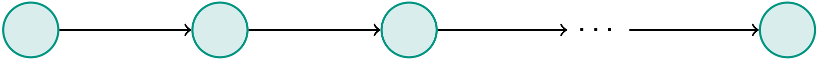
- **Relational Algorithmic Properties**, e.g., voting schemes

$$election_1 \sim eletion_2 \rightarrow [P_1][P_2]\, winner_1 \approx winner_2$$

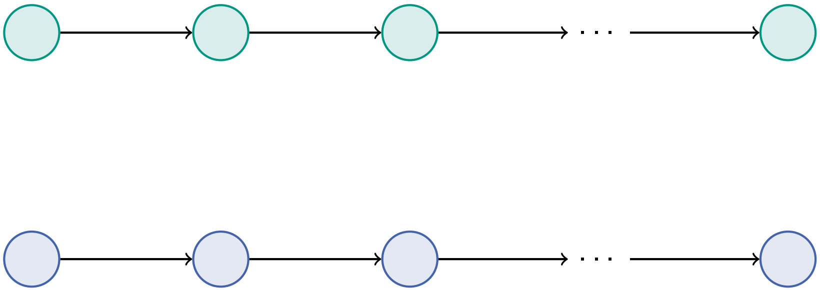
Effective/Efficient Verification of Relational Properties

Effective/Efficient Verification of Relational Properties

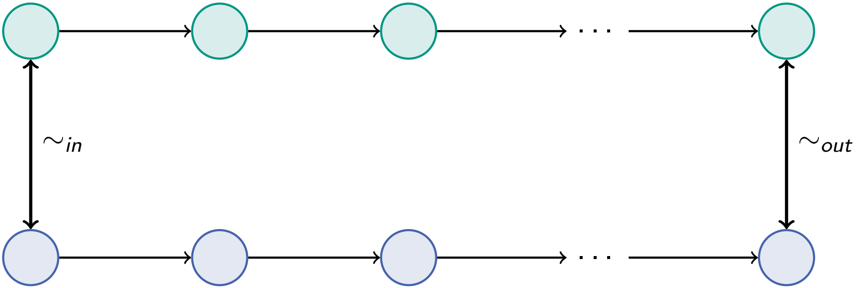
Synchronised Traces



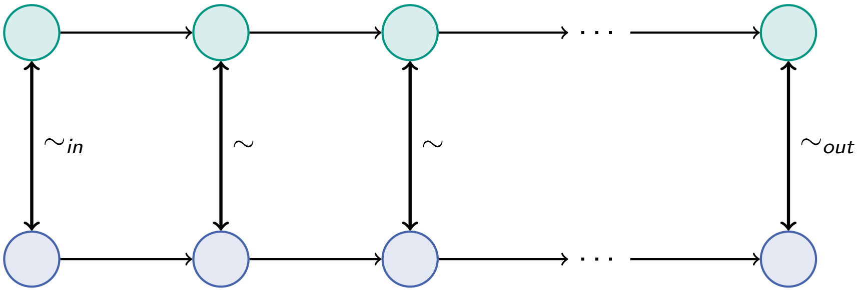
Synchronised Traces



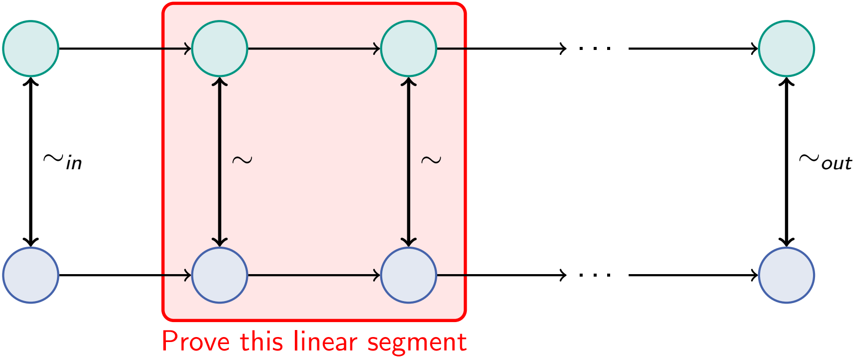
Synchronised Traces



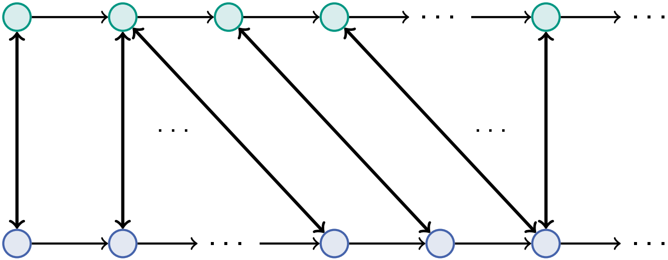
Synchronised Traces



Synchronised Traces



Losely Synchronised Traces



Why is relational verification often simpler?

Consider only single-loop programs with a single variable.

Why is relational verification often simpler?

Consider only single-loop programs with a single variable.

Claim

Proving an equality using individual loop abstraction
requires the strongest loop invariant.

Why is relational verification often simpler?

Consider only single-loop programs with a single variable.

Claim

Proving an equality using individual loop abstraction requires the strongest loop invariant.

Justification:

- Strongest loop abstraction is a functional relation.
- Any invariant weaker than the strongest has one input-state x_1, x_2 such that two post-states satisfy it.
- But outputs must be equal – equality is bound to fail for either x_1 or x_2.

Why is relational verification often simpler?

Consider only single-loop programs with a single variable.

Claim

Proving an equality using individual loop abstraction requires the strongest loop invariant.

Justification:

- Strongest loop abstraction is a functional relation.
- Any invariant weaker than the strongest has one input-state x_1, x_2 such that two post-states satisfy it.
- But outputs must be equal – equality is bound to fail for either x_1 or x_2.

Strongest functional invariants hard to specify/infer.
$\Rightarrow$ Relational regression verification is promising!

Contributions so far

- Refinement from algorithms to implementations [Ulbrich 11]
- Non-interference calculus in KeY [LOPSTR 13]
- Regression verification of C source code [ASE 14]
- Regression verification on PLC code [ICFEM 15]
- Verifying relational props of voting schemes [COMSOC 16]

Contributions so far

- Refinement from algorithms to implementations [Ulbrich 11]
- Non-interference calculus in KeY [LOPSTR 13]
- Regression verification of C source code [ASE 14]
- Regression verification on PLC code [ICFEM 15]
- Verifying relational props of voting schemes [COMSOC 16]

Similar, yet not the same

Similar techniques are used.
To be effective/efficient, technique must match application.

Contributions so far

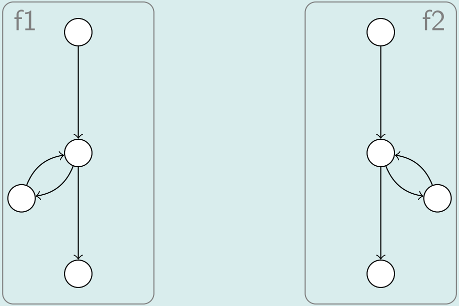
- Refinement from algorithms to implementations [Ulbrich 11]
- Non-interference calculus in KeY [LOPSTR 13]
- Regression verification of C source code [ASE 14]
- Regression verification on PLC code [ICFEM 15]
- Verifying relational props of voting schemes [COMSOC 16]
- Regression verification for LLVM bitcode [VSTTE 16]

Similar, yet not the same

Similar techniques are used.
To be effective/efficient, technique must match application.

Relational Verification

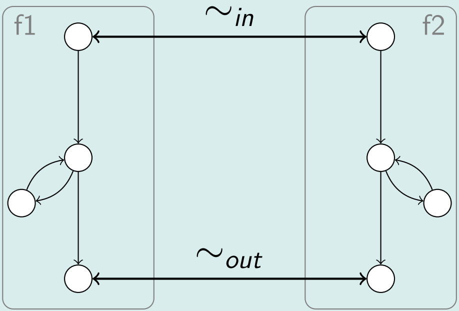
Loop synchronisation



- **To show:** Related input gives related output

Relational Verification

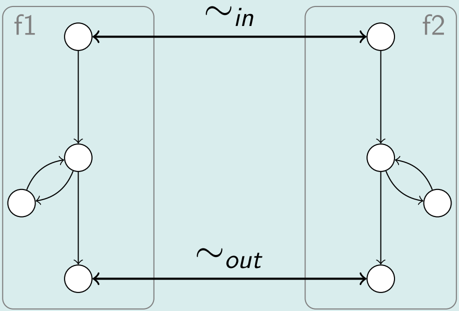
Loop synchronisation



- **To show:** Related input gives related output

Relational Verification

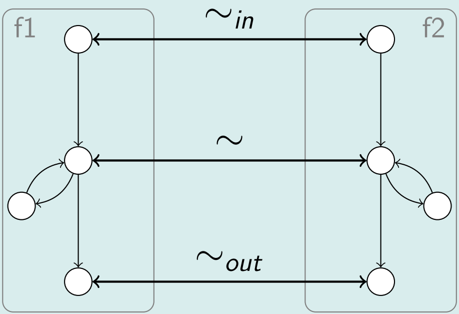
Loop synchronisation



- **To show:** Related input gives related output
- Loops are **synchronised**

Relational Verification

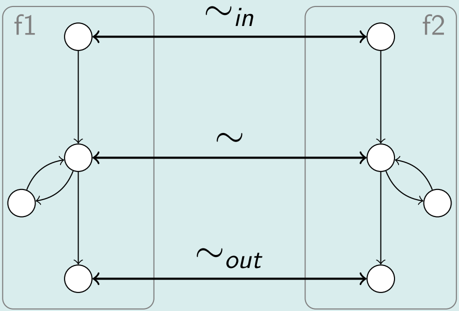
Loop synchronisation



- **To show:** Related input gives related output
- Loops are **synchronised**

Relational Verification

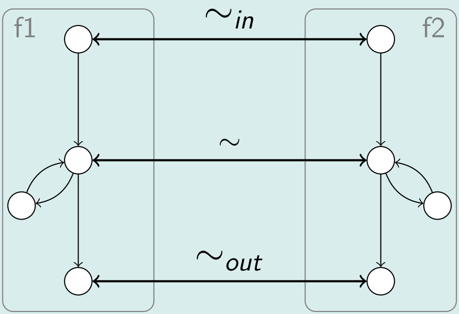
Loop synchronisation



- **To show:** Related input gives related output
- Loops are **synchronised**
- ... at least loosely synchronised

Relational Verification

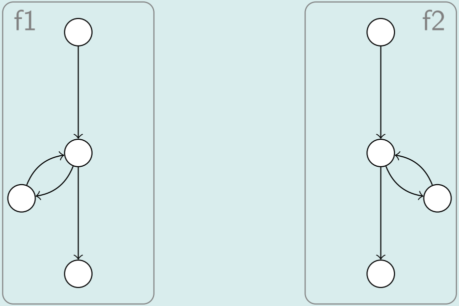
Loop synchronisation



- **To show:** Related input gives related output
- Loops are **synchronised**
- ... at least loosely synchronised

Relational Verification

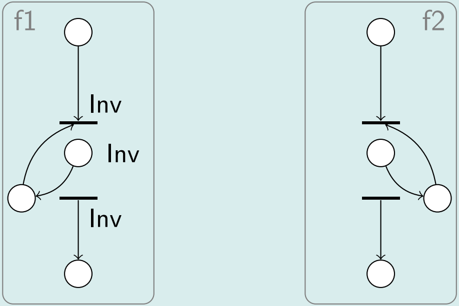
Loop synchronisation



- **To show:** Related input gives related output
- Loops are **synchronised**
- ... at least loosely synchronised
- Abstract loops by **invariants**

Relational Verification

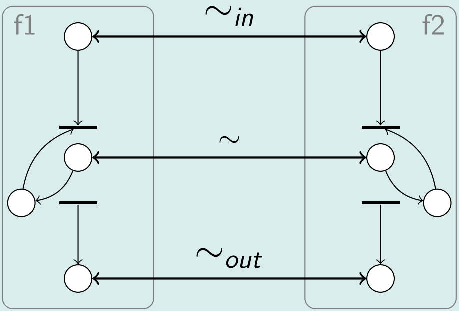
Loop synchronisation



- **To show:** Related input gives related output
- Loops are **synchronised**
- ... at least loosely synchronised
- Abstract loops by **invariants**

Relational Verification

Loop synchronisation



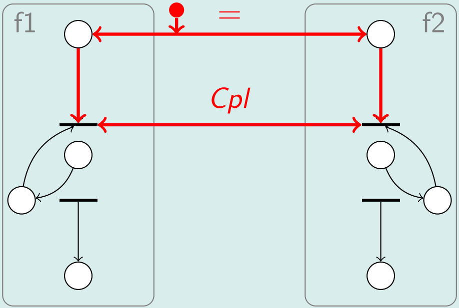
- **To show:** Related input gives related output
- Loops are **synchronised**
- ... at least loosely synchronised
- Abstract loops by **invariants**

$\Rightarrow$ Use *Cpl* as **loop invariant** for **both** programs.

($\rightarrow$*coupling invariant*)

Relational Verification

Loop synchronisation



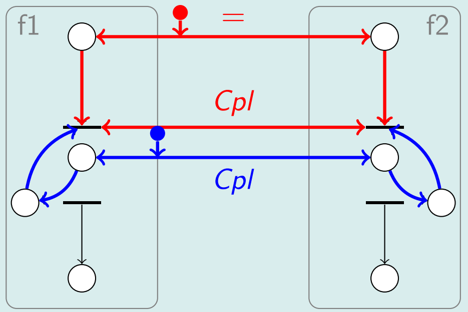
- **To show:** Related input gives related output
- Loops are **synchronised**
- … at least loosely synchronised
- Abstract loops by **invariants**

$\Rightarrow$ Use *Cpl* as **loop invariant** for **both** programs.

($\rightarrow$*coupling invariant*)

Relational Verification

Loop synchronisation



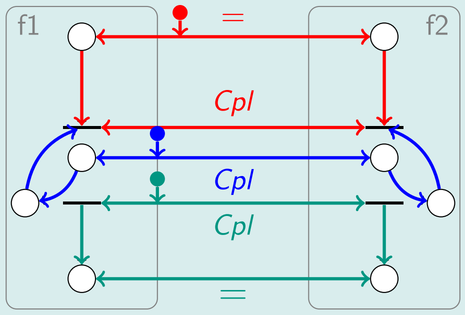
- **To show:** Related input gives related output
- Loops are **synchronised**
- ... at least loosely synchronised
- Abstract loops by **invariants**

⇒ Use *Cpl* as **loop invariant** for **both** programs.

(→*coupling invariant*)

Relational Verification

Loop synchronisation



- **To show:** Related input gives related output
- Loops are **synchronised**
- ... at least loosely synchronised
- Abstract loops by **invariants**

$\Rightarrow$ Use *Cpl* as **loop invariant** for **both** programs.

($\rightarrow$*coupling invariant*)