Some results and open problems in relational verification

Dave Naumann

Joint work with Ramana Nagasamudram and Anindya Banerjee

Cornell PLDG September 2025

This work was partially supported by NSF award CNS 2426414.

∀∀ relational property for imperative programs

$\boxed{c :\ p \rightsquigarrow q}$ partial correctness, commonly written $\{p\}\, c\, \{q\}$

$\boxed{c \mid d : \mathcal{R} \approx\!> \mathcal{S}}$ ∀∀-correctness, sometimes written $\{\mathcal{R}\}\, c \sim d\, \{\mathcal{S}\}$ and called 2-safety

$$\forall \quad \begin{tikzcd} \sigma \arrow[r, mapsto, "c"] \arrow[d, "\mathcal{R}"'] & \tau \\ \sigma' \arrow[r, mapsto, "d"] & \tau' \end{tikzcd} \quad \Longrightarrow \quad \begin{tikzcd} \tau \arrow[d, "\mathcal{S}"] \\ \tau' \end{tikzcd}$$

$$\forall \sigma\ \sigma'\ \tau\ \tau'.\ \ \sigma \mathcal{R} \sigma' \land \sigma [\![\, c \,]\!] \tau \land \sigma' [\![\, d \,]\!] \tau' \Rightarrow \tau \mathcal{S} \tau'$$

∀∀ relational property for imperative programs

$\boxed{c \mid d : \mathcal{R} \approx\!\!> \mathcal{S}}$

$$\forall \quad \begin{tikzcd} \sigma \arrow[r, mapsto, "c"] \arrow[d, "\mathcal{R}"'] & \tau \\ \sigma' \arrow[r, mapsto, "d"] & \tau' \end{tikzcd} \quad \Longrightarrow \quad \begin{tikzcd} \tau \arrow[d, "\mathcal{S}"] \\ \tau' \end{tikzcd}$$

$c \mid c : \overleftarrow{low} = \overrightarrow{low} \approx\!\!> \overleftarrow{low} = \overrightarrow{low}$ (c noninterferent)

$c \mid c : \overleftarrow{low} = \overrightarrow{low} \wedge (Fri \Rightarrow \overleftarrow{exp} = \overrightarrow{exp}) \approx\!\!> \overleftarrow{low} = \overrightarrow{low}$ (declassify)

$c \mid c : \overleftarrow{in} \leq \overrightarrow{in} \approx\!\!> \overleftarrow{out} \leq \overrightarrow{out}$ (monotonic)

$c; c \mid c : \overleftarrow{x} = \overrightarrow{x} \approx\!\!> \overleftarrow{x} = \overrightarrow{x}$ (idempotent)

$c \mid d : \overleftarrow{x} = \overrightarrow{x} \approx\!\!> \overleftarrow{x} = \overrightarrow{x}$ (d extensionally equivalent to c)

$c \mid d : \overleftarrow{x} = \overrightarrow{x} \approx\!\!> \overleftarrow{x} \leq \overrightarrow{x}$ (d majorizes c)

Verification example

Example: $c_0 \mid c_0 : \; 0 \leq \overleftarrow{x} \wedge \overleftarrow{x} = \overrightarrow{x} \;\approx\!\!> \; \overleftarrow{z} = \overrightarrow{z}$

$$c_0 \quad \widehat{=} \quad y := x; z := 1; \mathsf{while}\ y \neq 0\ \mathsf{do}\ z := z * y; y := y - 1\ \mathsf{od}$$

Verification example

Example: $c_0 \mid c_0 : \; 0 \leq \overleftarrow{x} \;\wedge\; \overleftarrow{x} = \overrightarrow{x} \;\approx\!> \; \overleftarrow{z} = \overrightarrow{z}$

$$c_0 \quad \widehat{=} \quad y := x; z := 1; \mathsf{while}\ y \neq 0\ \mathsf{do}\ z := z * y; y := y - 1\ \mathsf{od}$$

Use renamed copy to reduce to partial correctness:
$c_0; c_0' : x = x' \wedge x \geq 0 \leadsto z = z'$ (invariant includes $x! = z * y!$).

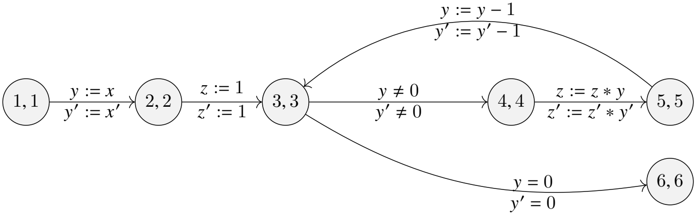
Verification example

Example: $c_0 \,|\, c_0 : \; 0 \le \overleftarrow{x} \;\wedge\; \overleftarrow{x} = \overrightarrow{x} \;\approx> \; \overleftarrow{z} = \overrightarrow{z}$

$$c_0 \quad \widehat{=} \quad y := x; z := 1; \mathsf{while}\ y \neq 0\ \mathsf{do}\ z := z * y; y := y - 1\ \mathsf{od}$$

Use renamed copy to reduce to partial correctness:
$c_0; c_0' : x = x' \wedge x \ge 0 \rightsquigarrow z = z'$ (invariant includes $x! = z * y!$).

Better: lockstep aligned product (invariant: $\overleftarrow{y} = \overrightarrow{y} \wedge \overleftarrow{z} = \overrightarrow{z}$).



Outline

Goal of this work: theory for foundational & usable auto-active verification tools. For general programs; maximizing automation.

Outline of this talk:

- deductive & automata-based ∀∀ reasoning (sequential programs, standard interpretation)
- alignment completeness
- ∀∃ automata, deduction, completeness
- logics & completeness beyond inductive assertion method
- (if time) reducing ∀∀ to ∀∃

Lockstep alignment of different loops

```
procedure fact (x: int) returns (z: int)
{ var y: int; y := 0; z := 1;
  while (y < x) { y := y+1; z := z*y; } }

procedure fact' (x: int) returns (z: int)
{ var y: int; y := 1; z := 1;
  while (y ≤ x) { z := z*y; y := y+1; } }

procedure fact_eq (x, x': int) returns (z, z': int)
  requires x == x' ∧ x ≥ 0;    ensures z == z';
{ var y, y': int;
  y := 0; y' := 1; z := 1; z' := 1;
  while (y < x)
    invariant y' == y + 1 ∧ 0 ≤ y ∧ y ≤ x;
    invariant z == z' ∧ z > 0;
    invariant y < x ⟺ y' ≤ x; // adequacy
    { y := y+1; z := z * y; z' := z' * y'; y' := y'+1; }
}
```

Relational Hoare logic [Francez'83, Benton'04, Yang'07, Barthe...]

Lockstep alignment:

$$\frac{c \mid c' : \mathcal{P} \approx\!> \mathcal{R} \qquad d \mid d' : \mathcal{R} \approx\!> \mathcal{Q}}{c; d \mid c'; d' : \mathcal{P} \approx\!> \mathcal{Q}} \qquad x := e \mid x' := e' : \mathcal{R}^{x|x'}_{e|e'} \approx\!> \mathcal{R}$$

$$\frac{\mathcal{P} \Rightarrow \overleftarrow{e} = \overrightarrow{e'} \qquad c \mid c' : \mathcal{P} \wedge \overleftarrow{e} \wedge \overrightarrow{e'} \approx\!> \mathcal{P}}{\textsf{while } e \textsf{ do } c \textsf{ od} \mid \textsf{while } e' \textsf{ do } c' \textsf{ od} : \mathcal{P} \approx\!> \mathcal{P} \wedge \neg\overleftarrow{e} \wedge \neg\overrightarrow{e'}}$$

Relational Hoare logic [Francez'83, Benton'04, Yang'07, Barthe...]

Lockstep alignment:

$$\frac{c \mid c' : \mathcal{P} \approx\!> \mathcal{R} \qquad d \mid d' : \mathcal{R} \approx\!> \mathcal{Q}}{c; d \mid c'; d' : \mathcal{P} \approx\!> \mathcal{Q}} \qquad x := e \mid x' := e' : \underbrace{\mathcal{R}^{x|x'}_{e|e'}}_{\text{subst.}} \approx\!> \mathcal{R}$$

$$\frac{\mathcal{P} \Rightarrow \overleftarrow{e} = \overrightarrow{e'} \qquad c \mid c' : \mathcal{P} \wedge \overleftarrow{e} \wedge \overrightarrow{e'} \approx\!> \mathcal{P}}{\mathsf{while}\ e\ \mathsf{do}\ c\ \mathsf{od} \mid \mathsf{while}\ e'\ \mathsf{do}\ c'\ \mathsf{od} : \mathcal{P} \approx\!> \mathcal{P} \wedge \neg\overleftarrow{e} \wedge \neg\overrightarrow{e'}}$$

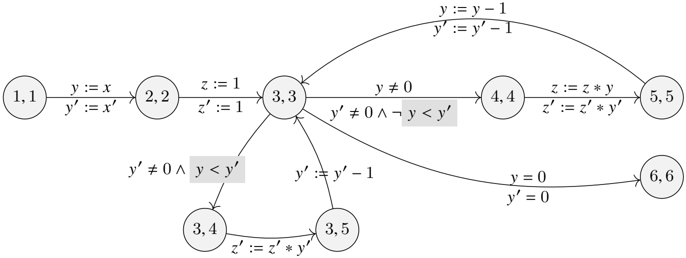
Sequential alignment:

$$\frac{c \mid \mathsf{skip} : \mathcal{P} \approx\!> \mathcal{R} \qquad \mathsf{skip} \mid c' : \mathcal{R} \approx\!> \mathcal{Q}}{c \mid c' : \mathcal{P} \approx\!> \mathcal{Q}}$$

Data dependent alignment

Example $c_0 \mid c_0 : 0 \leq \overleftarrow{x} \leq \overrightarrow{x} \approx\!> \overleftarrow{z} \leq \overrightarrow{z}$

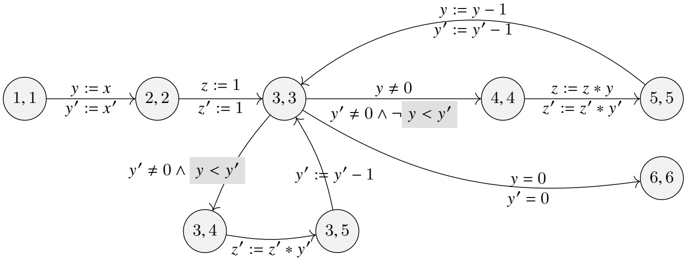
$c_0 \quad \widehat{=} \quad y := x; z := 1;$ while $y \neq 0$ do $z := z * y; y := y - 1$ od



Proof uses $an(3,3)\ \widehat{=}\ 0 \leq \overleftarrow{y} \leq \overrightarrow{y} \wedge 0 \leq \overleftarrow{z} \leq \overrightarrow{z}$ invariant at $(3, 3)$

Data dependent alignment

Example $c_0 \mid c_0 : 0 \leq \overleftarrow{x} \leq \overrightarrow{x} \approx\!> \overleftarrow{z} \leq \overrightarrow{z}$

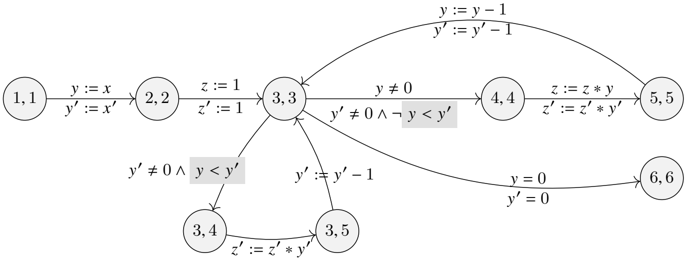


Proof uses $an(3,3) \mathrel{\widehat{=}} 0 \leq \overleftarrow{y} \leq \overrightarrow{y} \wedge 0 \leq \overleftarrow{z} \leq \overrightarrow{z}$ invariant at $(3,3)$

The automaton is *adequate*: covers all pairs of runs.

Data dependent alignment

Example $c_0 \,|\, c_0 : 0 \le \overleftarrow{x} \le \overrightarrow{x} \approx\!> \overleftarrow{z} \le \overrightarrow{z}$

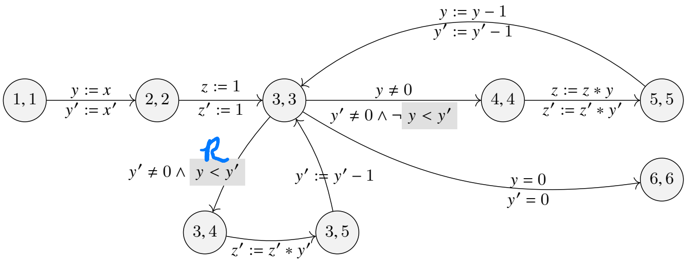


Proof uses $an(3,3) \mathrel{\widehat{=}} 0 \le \overleftarrow{y} \le \overrightarrow{y} \land 0 \le \overleftarrow{z} \le \overrightarrow{z}$ invariant at $(3,3)$

The automaton is *adequate*: covers all pairs of runs.

Alignment given by three state relations: when to do *L*eft-only, *R*ight-only, or *J*oint transitions.

Conditionally aligned loop rule



$$
\dfrac{
\begin{array}{c}
c \mid \mathsf{skip} : \mathcal{Q} \wedge \mathcal{L} \wedge \overleftarrow{e} \mathrel{\approx\!>} \mathcal{Q} \qquad \mathsf{skip} \mid c' : \mathcal{Q} \wedge \mathcal{R} \wedge \overrightarrow{e'} \mathrel{\approx\!>} \mathcal{Q} \\
c \mid c' : \mathcal{Q} \wedge \overleftarrow{e} \wedge \overrightarrow{e'} \wedge \neg\mathcal{L} \wedge \neg\mathcal{R} \mathrel{\approx\!>} \mathcal{Q} \\
\mathcal{Q} \Rightarrow (\overleftarrow{e} = \overrightarrow{e'}) \vee (\mathcal{L} \wedge \overleftarrow{e}) \vee (\mathcal{R} \wedge \overrightarrow{e'}) \quad \text{adequacy}
\end{array}
}{
\mathsf{while}\ e\ \mathsf{do}\ c\ \mathsf{od} \mid \mathsf{while}\ e'\ \mathsf{do}\ c'\ \mathsf{od} : \mathcal{Q} \mathrel{\approx\!>} \mathcal{Q} \wedge \neg\overleftarrow{e} \wedge \neg\overrightarrow{e'}
}
$$

For example: $\mathcal{L} := \mathit{false}$, $\mathcal{R} := \overleftarrow{y} < \overrightarrow{y}$.

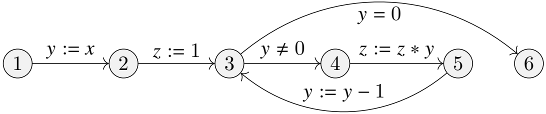
Cook Completeness

Theorem: the sequential alignment rule plus "unary Hoare logic" is complete (relative to assertion logic).

That is, $\models c \mid c' : \mathcal{P} \approx\!> \mathcal{Q}$ implies $\vdash c \mid c' : \mathcal{P} \approx\!> \mathcal{Q}$.

Problem: poor criterion, doesn't motivate the alignment rules.

Floyd completeness of Hoare logic

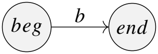


n	$an(n)$
1	$0 \leq x$
3	$x! = z * y!$
6	$x! = z$

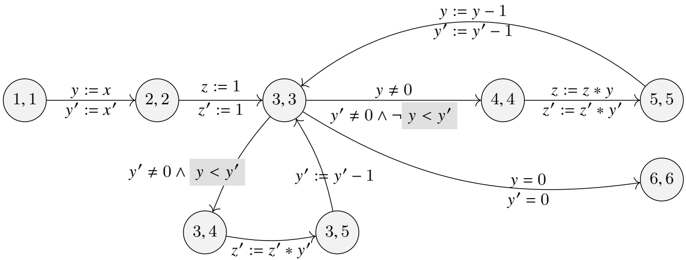
annotation for $P \rightsquigarrow Q$ assigns formulas to points in control-flow graph, cutting loops, with P initial and Q final

verification conditions: validity along paths

Thm[NN21]: given a valid annotation, an, for $c : P \rightsquigarrow Q$, there is a HL proof using only judgments

$b : an(beg) \rightsquigarrow an(end)$ for subprograms  of c.

Alignment completeness for certain automata



Theorem: for alignment products like the above, conditional loop + lockstep + one-side rules are alignment complete: Automata proof gives rise to deductive proof using “the same” assertions.

The associated judgments look like
$z := 1 \mid z := 1 : an(2,2) \approx\!> an(3,3)$.

Alignment completeness (∀∀)

Theorem: Given an adequate L, R, J-alignment automaton for $aut(c)$ and $aut(c')$, with a valid annotation an proving $\mathcal{P} \approx\!> \mathcal{Q}$, then there is a proof of $c \mid c' : \mathcal{P} \approx\!> \mathcal{Q}$ using "the same" assertions, by the RHL rules plus…

Alignment completeness (∀∀)

Theorem: Given an adequate L, R, J-alignment automaton for $aut(c)$ and $aut(c')$, with a valid annotation an proving $\mathcal{P} \approx\!> \mathcal{Q}$, then there is a proof of $c \,|\, c' : \mathcal{P} \approx\!> \mathcal{Q}$ using "the same" assertions, by the RHL rules plus…

Proof idea: Rewrite c and c' to an automaton-like normal form using explicit pc variable. The verification conditions imply this is invariant: $\land_{i,j}\,(\overleftarrow{pc} = i \land \overrightarrow{pc} = j \Rightarrow an(i,j))$.

control point label

$y :=^{1} x; z :=^{2} 1; \mathsf{do}^{3}\ y \neq 0 \rightarrow z :=^{4} z * y; y :=^{5} y - 1\ \mathsf{od}$

$$
\begin{array}{ll}
pc := 1; & \mathsf{do}\ pc = 1 \rightarrow y := x; pc := 2 \\
& [\!]\ pc = 2 \rightarrow x := 1; pc := 3 \\
& [\!]\ pc = 3 \land y \neq 0 \rightarrow pc := 4 \\
& [\!]\ pc = 3 \land y = 0 \rightarrow pc := 6 \qquad (*\ term.\ *) \\
& [\!]\ pc = 4 \rightarrow z := z * y; pc := 5 \\
& [\!]\ pc = 5 \rightarrow y := y - 1; pc := 3 \quad \mathsf{od}
\end{array}
$$

Additional rules for alignment completeness

Need disjunction, ghost elimination, and rewriting.

$$\frac{c \mid c' : \mathcal{R} \approx\!> \mathcal{S} \qquad [\![c]\!] = [\![d]\!] \qquad [\![c']\!] = [\![d']\!]}{d \mid d' : \mathcal{R} \approx\!> \mathcal{S}}\ \textsc{Lemma}$$

$$\frac{c \mid c' : \mathcal{R} \approx\!> \mathcal{S} \qquad \text{KAT,AX} \vdash c = d \qquad \text{KAT,AX} \vdash c' = d'}{d \mid d' : \mathcal{R} \approx\!> \mathcal{S}}\ \textsc{rRewrite}$$

Additional rules for alignment completeness

Need disjunction, ghost elimination, and rewriting.

$$\frac{c \mid c' : \mathcal{R} \approx\!> \mathcal{S} \qquad [\![c]\!] = [\![d]\!] \qquad [\![c']\!] = [\![d']\!]}{d \mid d' : \mathcal{R} \approx\!> \mathcal{S}}\ \textsc{Lemma}$$

$$\frac{c \mid c' : \mathcal{R} \approx\!> \mathcal{S} \qquad \text{KAT,AX} \vdash c = d \qquad \text{KAT,AX} \vdash c' = d'}{d \mid d' : \mathcal{R} \approx\!> \mathcal{S}}\ \textsc{rRewrite}$$

For any c there is $\hat{c}$ in automaton normal form such that KAT,AX $\vdash addPC(c) = \hat{c}$.

∀∃ properties for nondeterminism

$$\boxed{c \mid d : \mathcal{R} \overset{\exists}{\leadsto} \mathcal{S}} \qquad \forall \quad \begin{tikzcd} \sigma \arrow[r, mapsto, "c"] \arrow[d, "\mathcal{R}"'] & \tau \\ \sigma' & \end{tikzcd} \qquad \exists \quad \begin{tikzcd} & \tau \arrow[d, "\mathcal{S}"] \\ \sigma' \arrow[r, mapsto, "d"] & \tau' \end{tikzcd}$$

$$\forall \sigma\ \sigma'\ \tau.\ \ \sigma \mathcal{R} \sigma' \wedge \sigma [\![\, c \,]\!] \tau \Rightarrow \exists \tau'.\ \ \sigma' [\![\, d \,]\!] \tau' \wedge \tau \mathcal{S} \tau'$$

∀∃ properties for nondeterminism

$\boxed{c \mid d : \mathcal{R} \overset{\exists}{\rightsquigarrow} \mathcal{S}}$

$$\forall \quad \begin{array}{ccc} \sigma & \overset{c}{\longmapsto} & \tau \\ \mathcal{R}\big\downarrow & & \\ \sigma' & & \end{array} \qquad \exists \quad \begin{array}{ccc} & & \tau \\ & & \big\downarrow\mathcal{S} \\ \sigma' & \overset{d}{\longmapsto} & \tau' \end{array}$$

$c \mid c : \mathbb{A}low \overset{\exists}{\rightsquigarrow} \mathbb{A}low$ (possibilistic noninterference)

$\mathbb{A}x$ means $\overleftarrow{x} = \overrightarrow{x}$

$c \mid d : \mathbb{A}vars \overset{\exists}{\rightsquigarrow} \mathbb{A}vars$ (c refines d)

$\mathsf{skip} \mid c : \overrightarrow{p} \overset{\exists}{\rightsquigarrow} \overrightarrow{q}$ (forward underapprox, may term.)

$$\frac{b \mid c : \mathcal{P} \overset{\exists}{\rightsquigarrow} \mathcal{Q} \qquad c \mid d : \mathcal{R} \overset{\exists}{\rightsquigarrow} \mathcal{S}}{b \mid d : (\mathcal{P};\mathcal{R}) \overset{\exists}{\rightsquigarrow} (\mathcal{Q};\mathcal{S})}$$ (transitivity, vertical composition)

Note: $(\mathbb{A}x;\mathbb{A}x) \Leftrightarrow \mathbb{A}x$

Filtered alignment automata

Recall: automaton on state pairs, with state relations for *L*eft-only, *R*ight-only, or *J*oint transitions. Add: state relation K to block certain transitions.

Example: $\mathsf{hav}\ x \mid \mathsf{hav}\ y : \mathit{true} \overset{\exists}{\approx>} \overleftarrow{x} = \overrightarrow{y}$

Filtered alignment automata

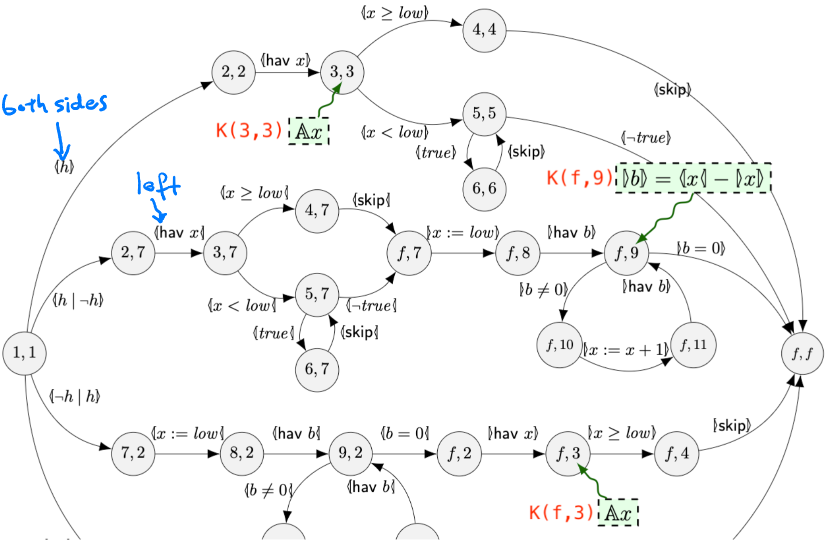
Recall: automaton on state pairs, with state relations for *L*eft-only, *R*ight-only, or *J*oint transitions. Add: state relation K to block certain transitions.

Example: $\mathsf{hav}\ x \,|\, \mathsf{hav}\ y : \mathit{true} \overset{\exists}{\approx\!>} \overleftarrow{x} = \overrightarrow{y}$

Example [Unno et al]: possibilistic NI $\quad \mathbb{A}low \overset{\exists}{\approx\!>} \mathbb{A}x$

$\mathsf{if}\ h \neq 0 \qquad$ (∗ branch on high∗)
$\mathsf{then}\ \{\mathsf{hav}\ x;\ \mathsf{if}\ x < low\ \mathsf{then\ while}\ \mathit{true}\ \mathsf{do\ skip};\ \}$
$\mathsf{else}\ \{x := low;\ \mathsf{hav}\ b;\ \mathsf{while}\ b \neq 0\ \mathsf{do}\ x := x + 1;\ \mathsf{hav}\ b;\}$

A filtered alignment automaton for [Unno et al]



Completeness of filtered automata

Theorem (soundness): Given (L, R, J, K)-automaton

- with annotation *an* that is inductive, for pre $\mathcal{P}$ post $\mathcal{Q}$
- and K satisfies local adequacy conditions w.r.t. *an*

then $\mathcal{P} \overset{\exists}{\approx\!\!>} \mathcal{Q}$ holds for the underlying programs.

Theorem (completeness): if $\mathcal{P} \overset{\exists}{\approx\!\!>} \mathcal{Q}$ holds then there is such an automaton.

Completeness of filtered automata

Theorem (soundness): Given (L, R, J, K)-automaton

- with annotation *an* that is inductive, for pre $\mathcal{P}$ post $\mathcal{Q}$
- and K satisfies local adequacy conditions w.r.t. *an*

then $\mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ holds for the underlying programs.

Theorem (completeness): if $\mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ holds then there is such an automaton.

Inductive assertion method plus "relative liveness" checks.

A logic of forward simulation

$$\mathsf{skip} \mid \mathsf{hav}\ x : (\exists |x.\, \mathcal{P}) \overset{\exists}{\rightsquigarrow} \mathcal{P} \qquad \mathsf{hav}\ x \mid \mathsf{skip} : (\forall x|.\, \mathcal{P}) \overset{\exists}{\rightsquigarrow} \mathcal{P}$$

on right

on left

$$\frac{\begin{array}{c} c \mid \mathsf{skip} : \mathcal{Q} \wedge \overleftarrow{e} \wedge \mathcal{L} \overset{\exists}{\rightsquigarrow} \mathcal{Q} \\ \mathsf{skip} \mid c' : \mathcal{Q} \wedge \overrightarrow{e'} \wedge \mathcal{R} \wedge \boxed{n = E} \overset{\exists}{\rightsquigarrow} \mathcal{Q} \wedge \boxed{0 \leq E < n} \text{ for all } n \in \mathbb{Z} \\ c \mid c' : \mathcal{Q} \wedge \overleftarrow{e} \wedge \overrightarrow{e'} \wedge \neg\mathcal{L} \wedge \neg\mathcal{R} \overset{\exists}{\rightsquigarrow} \mathcal{Q} \\ \mathcal{Q} \Rightarrow (\overleftarrow{e} = \overrightarrow{e'}) \vee (\mathcal{L} \wedge \overleftarrow{e}) \vee (\mathcal{R} \wedge \overrightarrow{e'}) \end{array}}{\mathsf{while}\ e\ \mathsf{do}\ c\ \mathsf{od} \mid \mathsf{while}\ e'\ \mathsf{do}\ c'\ \mathsf{od} : \mathcal{Q} \overset{\exists}{\rightsquigarrow} \mathcal{Q} \wedge \neg\overleftarrow{e} \wedge \neg\overrightarrow{e'}}$$

Consequence, assign, lockstep loop, etc – same as for ∀∀.

Alignment completeness (∀∃)

Theorem: Given an adequate L, R, J, K-alignment automaton with a valid annotation proving $\mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$, then there is a proof using “the same” assertions, by the rules above plus disjunction, rewrite, and ghost elimination.

Proof idea: as for ∀∀, relying on ∀∃ adequacy.

Alignment completeness (∀∃)

Theorem: Given an adequate L, R, J, K-alignment automaton with a valid annotation proving $\mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$, then there is a proof using "the same" assertions, by the rules above plus disjunction, rewrite, and ghost elimination.

Proof idea: as for ∀∀, relying on ∀∃ adequacy.

Problem: our proofs of alignment completeness don't imply a practical technique.

Beyond IAM and alignment (and back to ∀∀)

Far beyond: arbitrary trace relations

Slightly beyond: rule of conjunction; frame rule

Hypotheses:

- relational procedure specs $\quad f(x) : \mathcal{P}(x) \approx\!> \mathcal{Q}(x) \vdash c : \mathcal{R} \approx\!> \mathcal{S}$
- commutativity $\quad c; d \mid d; c : \mathbb{A}x \approx\!> \mathbb{A}x$,

$$c \mid c : \mathbb{A}x \approx\!> \mathbb{A}x, \quad d \mid d : \mathbb{A}x \approx\!> \mathbb{A}x$$

$$\models \quad c; d; d \mid d; d; c : \mathbb{A}x \approx\!> \mathbb{A}x$$

Beyond IAM and alignment (and back to ∀∀)

Far beyond: arbitrary trace relations

Slightly beyond: rule of conjunction; frame rule

Hypotheses:

- relational procedure specs $f(x) : \mathcal{P}(x) \approx\!> \mathcal{Q}(x) \vdash c : \mathcal{R} \approx\!> \mathcal{S}$
- commutativity
$$\begin{array}{rl} & \boxed{c;d \mid d;c : \mathbb{A}x \approx\!> \mathbb{A}x}, \\ & c \mid c : \mathbb{A}x \approx\!> \mathbb{A}x, \quad d \mid d : \mathbb{A}x \approx\!> \mathbb{A}x \\ \models & c;d;d \mid d;d;c : \mathbb{A}x \approx\!> \mathbb{A}x \end{array}$$
- idempotence
$$\begin{array}{rl} & \boxed{c \mid c : \overleftarrow{x} = v \approx\!> \overrightarrow{x} = v \Rightarrow \overleftarrow{x} = v}, \\ & c \mid c : \mathbb{A}x \approx\!> \mathbb{A}x \\ \models & c \mid c;c : \mathbb{A}x \approx\!> \mathbb{A}x \end{array}$$
- hoisting an idempotent out of a loop

[D'Osualdo et al'22]

Rules for D'Osualdo examples

D'Osualdo et al propose an k-safety logic with varying k (and a semantic rule for rewriting); use 3 for the examples.

For commutativity:

$$\frac{b \mid c : \mathcal{P} \approx\!> \mathcal{Q} \qquad c \mid d : \mathcal{R} \approx\!> \mathcal{S} \qquad \boxed{\mathsf{skip} \mid c : \mathcal{P} \overset{\exists}{\approx\!>} \mathit{true}}}{b \mid d : (\mathcal{P};\mathcal{R}) \approx\!> (\mathcal{Q};\mathcal{S})}$$

For idempotence:

$$\frac{c \mid c' : \mathcal{P} \approx\!> \mathsf{wp}(d \mid \mathsf{skip})(\mathcal{Q})}{c;d \mid c' : \mathcal{P} \approx\!> \mathcal{Q}} \qquad \frac{c \mid c' : \mathcal{R} \approx\!> \mathcal{S} \qquad \mathsf{skip} \mid c' : \overrightarrow{p} \approx\!> \overrightarrow{q}}{c \mid c' : \mathcal{R} \wedge \overrightarrow{p} \approx\!> \mathcal{S} \wedge \overrightarrow{q}}$$

Rules for D'Osualdo examples

D'Osualdo et al propose an k-safety logic with varying k (and a semantic rule for rewriting); use 3 for the examples.

For commutativity:

$$\frac{b \mid c : \mathcal{P} \approx\!> \mathcal{Q} \qquad c \mid d : \mathcal{R} \approx\!> \mathcal{S} \qquad \boxed{\mathsf{skip} \mid c : \mathcal{P} \overset{\exists}{\approx\!>} \mathit{true}}}{b \mid d : (\mathcal{P}; \mathcal{R}) \approx\!> (\mathcal{Q}; \mathcal{S})}$$

For idempotence:

$$\frac{c \mid c' : \mathcal{P} \approx\!> \mathsf{wp}(d \mid \mathsf{skip})(\mathcal{Q})}{c; d \mid c' : \mathcal{P} \approx\!> \mathcal{Q}} \qquad \frac{c \mid c' : \mathcal{R} \approx\!> \mathcal{S} \qquad \mathsf{skip} \mid c' : \overrightarrow{p} \approx\!> \overrightarrow{q}}{c \mid c' : \mathcal{R} \land \overrightarrow{p} \approx\!> \mathcal{S} \land \overrightarrow{q}}$$

$$\frac{\mathsf{skip} \mid c' : \overrightarrow{p} \approx\!> \mathcal{Q}}{\mathsf{skip} \mid c' : \overrightarrow{p} \approx\!> \overrightarrow{\mathsf{diag}(\mathcal{Q})}}$$

So what?

Entailment completeness

$$\boxed{\text{If } hyps \models c \mid c' : \mathcal{P} \approx\!> \mathcal{Q} \text{ then } hyps \vdash c \mid c' : \mathcal{P} \approx\!> \mathcal{Q}.}$$

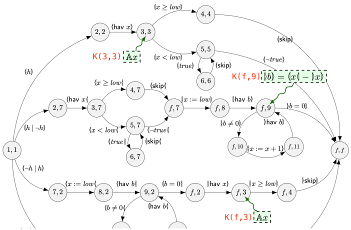
Q: What rules (and judgments) suffice for entailment complete ∀∀ logic? ∀∀ together with ∀∃?

Q: How prove that, say RHL+, isn't entailment complete?

Precursors:

- trivial with implication operator (Reynolds' Spec Logic; shallow emb.)
- KAT, Propositional Dynamic Logic
- Propositional Hoare Logic [Kozen,Tiuryn'01]

Filter automata in program form



Specialize filter to right-side havoc.

```
if (high ≠ 0 ∧ high' ≠ 0) {
   havoc x;
   assert (∃ v:int • x == v); // added by chk
   havoc x'; assume x == x';
   if (x ≥ low ∨ x' ≥ low') { /*skip*/ }
   else { while (true) { /*skip*/ } }
}
```

Filter adequacy transformation

Theorem: if $chk(prod) : \mathcal{P} \approx\!> \mathcal{Q}$ then $c \,|\, c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ for underlying c, c' of $prod$. (WhyRel implementation, Rocq formalization)

Auto-active solver hacking…

```
else if (high ≠ 0 ∧ high' == 0) {
   havoc x;
   if (x ≥ low) { /*skip*/ }
   else { while (true) { /*skip*/ } }
    x' ≔ low;
    assert inst(x - x');
    assert (∃ v:int • {inst(v)} v == x - x'); // added by chk
    havoc b'; assume b' == x - x';
    while (b' ≠ 0)
       invariant x ≥ x' ∧ b' == x - x'; /* variant b' */
    {  bsnap' ≔ b'; // added by chk
       x' ≔ x' + 1;
       assert inst(x - x');
       assert (∃ v:int • {inst(v)} v == x - x'); // added by chk
       havoc b'; assume b' == x - x';
       assert b' < bsnap'; // added by chk
    }
 }
```

Summary

- Syntax-oriented logic can express alignment-based relational verification (inductive assertion method), and guide design of auto-active tools.
- Open Q: minimizing rewriting for proof of alignment completeness. (Compare the impact of wlp, versus Gödel encoding, for Cook completeness proofs.)
- Related Q: deriving a syntactic alignment product from alignment conditions inferred for unstructured transition system.
- Entailment of correctness judgments motivates additional rules like vertical composition, wlp assertions…
- Qpen Q: Is there an entailment complete system? for ∀∀ together with ∀∃? Must it go beyond 2-properties? Robust proof for interpreted logic?

Credits

Cook completeness of RHLs [Francez'83, Barthe et al'04, ...]

Relational logic with framing, procedures, abstraction [Banerjee, Nagasamudram, Nikouei, Naumann TOPLAS'22, TACAS'23]

Alignment complete relational Hoare logics for some and all. [Nagasamudram, Banerjee, Naumann; to appear in LMCS; see v5 for D'Osualdo examples `https://arxiv.org/abs/2307.10045v5`.]

Proving Hypersafety Compositionally. [D'Osualdo, Farzan, Dreyer OOPSLA'22; cf. Bao et al POPL'25.]

Forall-exists relational verification by filtering to forall-forall. [Nagasamudram, Banerjee, Naumann; `https://arxiv.org/abs/2509.04777`.]

Example conditional alignment

```
procedure hiccupSum (n: int) returns (r: int)
{ var h: bool; var i: int;
  i, r := 0, 0;
  havoc h;
  while (i < n) {
    if (!h) { r := r+i; i := i+1; }
    havoc h;
  }
}
procedure hiccupSum_eq (n, n': int) returns (r, r': int)
  requires n == n';
  ensures r == r';
{
  var h, h': bool; var i, i': int;
  i, r := 0, 0;   i', r' := 0, 0;
  havoc h; havoc h';

  while (i < n ∨ i' < n') // left/right alignment conditions h/h'
    invariant i == i';
    invariant r == r';
    {
      if (h ∧ i < n) { // left body only
        if (!h) { r := r+i; i := i+1; } havoc h;
      } else if (h' ∧ i' < n') { // right body only
        if (!h') { r' := r'+i'; i' := i'+1; } havoc h';
      } else if (i < n ∧ i' < n' ∧ !h ∧ !h') { // both
        if (!h) { r := r+i; i := i+1; } havoc h;
        if (!h') { r' := r'+i'; i' := i'+1; } havoc h';
      } else {
        assert false; // adequacy condition
      }
```