Peter Müller

Joint work with Thibault Dardinier and Anqi Li

HYPER HOARE LOGIC

ETH zürich

Program Properties

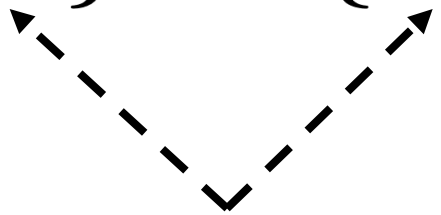
Correctness	Absence of bugs	$\forall$
Reachability	Presence of bugs	$\exists$
Determinism	Hash functions	$\forall\forall$
Transitivity	Comparators	$\forall\forall\forall$
Non-interference	Secure information flow	$\forall\forall$
Generalized non-interference	Secure information flow	$\forall\forall\exists$
Existence of a minimum	Optimization algorithms	$\exists\forall$

Our goal:

Develop a program logic that can (dis-)prove arbitrary program hyperproperties

Hyper-Triples

$$\{\,\mathbf{P}\,\}\ C\ \{\,\mathbf{Q}\,\}$$



Properties over sets of reachable states

$$\vDash \{\,\mathbf{P}\,\}\ C\ \{\,\mathbf{Q}\,\} \ \Leftrightarrow\ \forall S \cdot \mathbf{P}(S) \Rightarrow \mathbf{Q}(\mathsf{sem}(C, S))$$

$$\mathsf{sem}(C, S) \ =\ \{\sigma' \mid \exists \sigma \in S \cdot \langle C, \sigma \rangle \rightarrow \sigma'\}$$

Examples

- Correctness

$$\{\, \lambda S.\, \forall \sigma \in S \cdot \sigma(\mathsf{x}) > 0 \land \sigma(\mathsf{y}) > 0 \,\}\ \mathsf{r} := \mathsf{x} * \mathsf{y}\ \{\, \lambda S'.\, \forall \sigma' \in S' \cdot \sigma'(\mathsf{r}) > 0 \,\}$$

- Reachability

$$\{\, \lambda S.\, \exists \sigma \in S \cdot \mathsf{true} \,\}\ \mathsf{x} := 0 \,\square\, \mathsf{x} := 1\ \{\, \lambda S'.\, \exists \sigma' \in S' \cdot \sigma'(\mathsf{x}) = 0 \,\}$$

- Determinism and non-interference ($\forall\forall$)

$$\{\, \lambda S.\, \forall \sigma_0, \sigma_1 \in S \cdot \sigma_0(\mathsf{x}) = \sigma_1(\mathsf{x}) \,\}\ \mathsf{r} := \mathsf{hash}(\mathsf{x})\ \{\, \lambda S'.\, \forall \sigma'_0, \sigma'_1 \in S' \cdot \sigma'_0(\mathsf{r}) = \sigma'_1(\mathsf{r}) \,\}$$

- Quantification over executions becomes explicit in assertions

- Hyper-triples can express over-approximate properties (like Hoare triples) and under-approximate properties (like triples in Incorrectness logic)

Examples: Tracking Executions

- Attempt to express monotonicity ($\forall\forall$)

$$\{\, \lambda S.\, \forall \sigma_0, \sigma_1 \in S \cdot \sigma_0(\mathsf{x}) \leq \sigma_1(\mathsf{x}) \,\}\ \texttt{r := foo(x)}\ \{\, \lambda S'.\, \forall \sigma'_0, \sigma'_1 \in S' \cdot \sigma'_0(\mathsf{r}) \leq \sigma'_1(\mathsf{r}) \,\}$$

- Use logical variables to track different executions
  - Logical variables cannot be changed by program executions: $\sigma(\mathsf{T}) = \sigma'(\mathsf{T})$

- Monotonicity

$$\{\, \lambda S.\, \forall \sigma_0, \sigma_1 \in S \cdot \sigma_0(\mathsf{T}) = 0 \land \sigma_1(\mathsf{T}) = 1 \Rightarrow \sigma_0(\mathsf{x}) \leq \sigma_1(\mathsf{x}) \,\}$$

```
r := foo(x)
```

$$\{\, \lambda S'.\, \forall \sigma'_0, \sigma'_1 \in S' \cdot \sigma'_0(\mathsf{T}) = 0 \land \sigma'_1(\mathsf{T}) = 1 \Rightarrow \sigma'_0(\mathsf{r}) \leq \sigma'_1(\mathsf{r}) \,\}$$

Core Rules

Commands

$$C ::= \ \textbf{skip} \mid \mathsf{x} := \mathsf{e} \mid \textbf{havoc}\ \mathsf{x} \mid \textbf{assume}\ \mathsf{b} \mid C_0; C_1 \mid C_0 \,[]\, C_1 \mid C^*$$

- We use the usual encodings

$$\textbf{if}\ (\mathsf{b})\ \{\, C_0 \,\}\ \textbf{else}\ \{\, C_1 \,\} \equiv \ (\textbf{assume}\ \mathsf{b}; C_0) \,[]\, (\textbf{assume}\ \neg\mathsf{b}; C_1)$$

$$\textbf{while}\ (\mathsf{b})\ \{\, C \,\} \equiv \ (\textbf{assume}\ \mathsf{b}; C)^*; \textbf{assume}\ \neg\mathsf{b}$$

Basic Rules

$$\frac{}{\vdash \{\, \lambda S.\mathbf{P}(\{\sigma[\mathsf{x} \mapsto \mathsf{e}(\sigma)] \mid \sigma \in S\}) \,\}\ \mathsf{x}\ :=\ \mathsf{e}\ \{\, \mathbf{P} \,\}}\ (Assign)$$

$$\frac{}{\vdash \{\, \lambda S.\mathbf{P}(\{\sigma[\mathsf{x} \mapsto v] \mid \sigma \in S\}) \,\}\ \textbf{havoc}\ \mathsf{x}\ \{\, \mathbf{P} \,\}}\ (Havoc)$$

$$\frac{}{\vdash \{\, \lambda S.\,\mathbf{P}(\{\sigma \in S \mid \mathsf{b}(\sigma)\}) \,\}\ \textbf{assume}\ \mathsf{b}\ \{\, \mathbf{P} \,\}}\ (Assume)$$

$$\frac{\vdash \{\, \mathbf{P} \,\}\ C_0\ \{\, \mathbf{R} \,\} \qquad \vdash \{\, \mathbf{R} \,\}\ C_1\ \{\, \mathbf{Q} \,\}}{\vdash \{\, \mathbf{P} \,\}\ C_0; C_1\ \{\, \mathbf{Q} \,\}}\ (Seq)$$

Control Flow

- Post-states of a non-deterministic choice is union of post-states of the branches

$$\frac{\vdash \{\,\mathbf{P}\,\}\, C_0\, \{\,\mathbf{Q}_0\,\} \quad \vdash \{\,\mathbf{P}\,\}\, C_1\, \{\,\mathbf{Q}_1\,\}}{\vdash \{\,\mathbf{P}\,\}\, C_0 \,[]\, C_1\, \{\,\mathbf{Q}_0 \otimes \mathbf{Q}_1\,\}}\,(Choice)$$

$$\mathbf{Q}_0 \otimes \mathbf{Q}_1 \;\equiv\; \lambda S.\exists S_0, S_1 \cdot S = S_0 \cup S_1 \wedge \mathbf{Q}_0(S_0) \wedge \mathbf{Q}_1(S_1)$$

- Iteration repeats this choice a finite or infinite number of times

$$\frac{\vdash \{\,\mathbf{I}_n\,\}\, C\, \{\,\mathbf{I}_{n+1}\,\}}{\vdash \{\,\mathbf{I}_0\,\}\, C^*\, \{\,\bigotimes_{n\in\mathbb{N}} \mathbf{I}_n\,\}}\,(Iter)$$

$$\bigotimes_{n\in\mathbb{N}} \mathbf{I}_n \;\equiv\; \lambda S.\exists \overline{S_i} \cdot (S = \bigcup_{n\in\mathbb{N}} S_n) \;\wedge\; (\forall n \in \mathbb{N} \cdot \mathbf{I}_n(S_n))$$

Command-Independent Rules

- Rule of consequence

$$\frac{\mathbf{P} \models \mathbf{P}' \quad \mathbf{Q}' \models \mathbf{Q} \quad \vdash \{\mathbf{P}'\}\, C\, \{\mathbf{Q}'\}}{\vdash \{\mathbf{P}\}\, C\, \{\mathbf{Q}\}}\,(Cons)$$

- Exist-rule is necessary for completeness

$$\frac{\forall \mathsf{x} \cdot (\vdash \{\mathbf{P}\}\, C\, \{\mathbf{Q}\})}{\vdash \{\exists \mathsf{x} \cdot \mathbf{P}\}\, C\, \{\exists \mathsf{x} \cdot \mathbf{Q}\}}\,(Exist)$$

Soundness, Completeness, Expressiveness

- The core rules are sound and complete

- A program hyperproperty relates the pre- and post-states of terminating executions (a set of pairs of states)

- Every program hyperproperty can be expressed as a hyper-triple

- The negation of a hyper-triple can be expressed as a hyper-triple

Syntactic Rules

Syntactic Assertions

- Syntactic hyper-assertions interact with sets of states only through quantification
  - This limitation is not a relevant restriction in practice

- Assertions

$$\mathsf{e} ::= \mathsf{c} \mid \mathsf{y} \mid \sigma(\mathsf{x}) \mid \mathsf{e} \circ \mathsf{e} \mid f(\mathsf{e})$$

$$A ::= \mathsf{b} \mid A \wedge A \mid A \vee A \mid \forall \mathsf{y} \cdot A \mid \exists \mathsf{y} \cdot A \mid \forall \langle \sigma \rangle \cdot A \mid \exists \langle \sigma \rangle \cdot A$$

  - Negation is defined recursively

Syntactic Proof Rules

- The assignment rule performs a syntactic substitution for each state look-up

$$\frac{}{\vdash \{\, \mathbf{P}[\sigma(\mathsf{x}) \mapsto \mathsf{e}(\sigma)] \,\}\ \mathsf{x}\ :=\ \mathsf{e}\ \{\, \mathbf{P} \,\}}\ (Assign)$$

- The substitution in the havoc rule introduces a quantified value *v* for each occurrence of σ(x)

$$\frac{}{\vdash \{\, \forall \langle\sigma\rangle \cdot \forall v \cdot v > 0 \,\}\ \textbf{havoc}\ \mathsf{x}\ \{\, \forall \langle\sigma\rangle \cdot \sigma(\mathsf{x}) > 0 \,\}}$$

$$\frac{}{\vdash \{\, \exists \langle\sigma\rangle \cdot \exists v \cdot v > 0 \,\}\ \textbf{havoc}\ \mathsf{x}\ \{\, \exists \langle\sigma\rangle \cdot \sigma(\mathsf{x}) > 0 \,\}}$$

Syntactic Proof Rules

- The syntactic transformation in the assume rule also depends on the quantifier

$$\frac{}{\vdash \{\, \forall\langle\sigma\rangle \cdot \sigma(\mathsf{x}) = 7 \Rightarrow \sigma(\mathsf{x}) > 0 \,\}\ \textbf{assume}\ \mathsf{x} = 7\ \{\, \forall\langle\sigma\rangle \cdot \sigma(\mathsf{x}) > 0 \,\}}$$

$$\frac{}{\vdash \{\, \exists\langle\sigma\rangle \cdot \sigma(\mathsf{x}) = 7 \wedge \sigma(\mathsf{x}) > 0 \,\}\ \textbf{assume}\ \mathsf{x} = 7\ \{\, \exists\langle\sigma\rangle \cdot \sigma(\mathsf{x}) > 0 \,\}}$$

- The proof rule for sequential composition, the rule of consequence, and the exist-rule remain unchanged

Rules for Synchronized Loops

- For potentially non-terminating loops

$$\frac{\mathbf{I} \models \mathtt{low(b)} \qquad \vdash \{\, \mathbf{I} \wedge \Box \mathtt{b} \,\}\ C\ \{\, \mathbf{I} \,\}}{\vdash \{\, \mathbf{I} \,\}\ \textbf{while}\ (\mathtt{b})\ \{\, C \,\}\ \{\, (\mathbf{I} \vee \mathtt{emp}) \wedge \Box \neg \mathtt{b} \,\}}$$

$$\begin{aligned} \mathtt{low(b)} &\equiv \forall \langle \sigma \rangle, \langle \sigma' \rangle \cdot \mathtt{b}(\sigma) = \mathtt{b}(\sigma') \\ \Box \mathtt{e} &\equiv \forall \langle \sigma \rangle \cdot \mathtt{e}(\sigma) \\ \mathtt{emp} &\equiv \forall \langle \sigma \rangle \cdot \mathtt{false} \end{aligned}$$

- For terminating loops

$$\frac{\mathbf{I} \models \mathtt{low(b)} \qquad \vdash_{\Downarrow} \{\, \mathbf{I} \wedge \Box(\mathtt{b} \wedge \mathtt{d} = \mathtt{D}) \,\}\ C\ \{\, \mathbf{I} \wedge \Box(\mathtt{d} < \mathtt{D}) \,\}}{\vdash_{\Downarrow} \{\, \mathbf{I} \,\}\ \textbf{while}\ (\mathtt{b})\ \{\, C \,\}\ \{\, \mathbf{I} \wedge \Box \neg \mathtt{b} \,\}}$$

d is the ranking function
< is well-founded
D is a fresh logical variable

Automation

General Approach

- Use an intermediate verification language (IVL) and off-the-shelf verifier such as Boogie, Why3, or Viper to encode programs, assertions, and proof rules

- Model all executions of the input program by a single execution of the IVL program

- Represent states as maps from variables to values

- Track sets of states in IVL variables

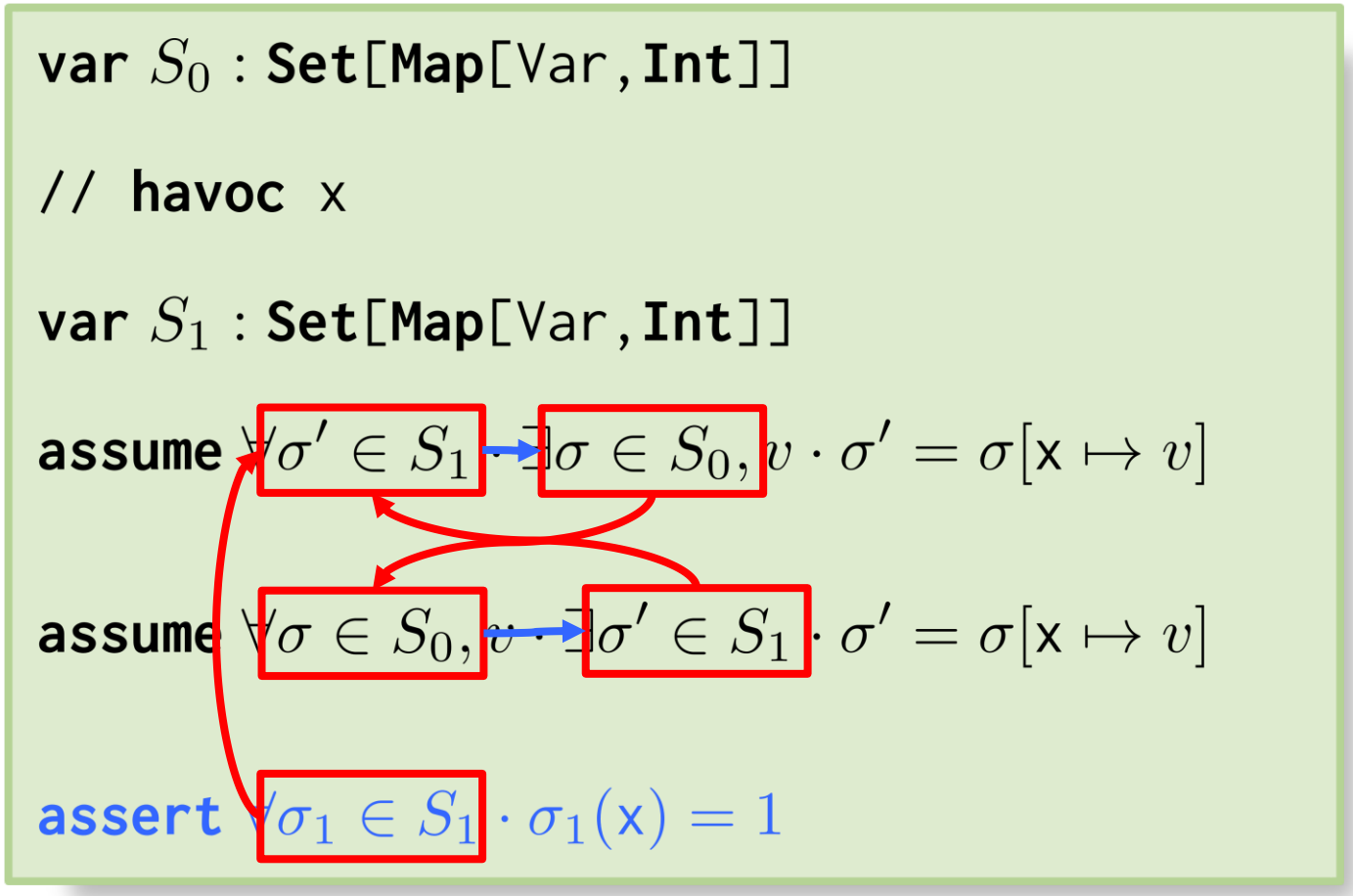
```
havoc x
```

```
var S_0 : Set[Map[Var,Int]]

// havoc x

var S_1 : Set[Map[Var,Int]]
assume ∀σ_1 ∈ S_1 · ∃σ_0 ∈ S_0, v · σ_1 = σ_0[x ↦ v]
assume ∀σ_0 ∈ S_0, v · ∃σ_1 ∈ S_1 · σ_1 = σ_0[x ↦ v]
```

Problem: Matching Loops



Tracking Bounds on Reachable States

- We track separately a lower bound $S^{\forall}$ and an upper bound $S^{\exists}$ on the set of states

$$S^{\forall} \subseteq S \subseteq S^{\exists}$$

- We do not assume in general that they are equal

- This encoding eliminates matching loops

var $S_0^{\forall}$: **Set**[**Map**[Var, **Int**]]

var $S_0^{\exists}$: **Set**[**Map**[Var, **Int**]]

// **havoc** x

var $S_1^{\forall}$: **Set**[**Map**[Var, **Int**]]

var $S_1^{\exists}$: **Set**[**Map**[Var, **Int**]]

assume $\forall \sigma' \in S_1^{\forall} \cdot \exists \sigma \in S_0^{\forall}, v \cdot \sigma' = \sigma[\mathsf{x} \mapsto v]$

assume $\forall \sigma \in S_0^{\exists}, v \cdot \exists \sigma' \in S_1^{\exists} \cdot \sigma' = \sigma[\mathsf{x} \mapsto v]$

assert $\forall \sigma_1 \in S_1^{\forall} \cdot \sigma_1(\mathsf{x}) = 1$

Encoding of Assertions

requires $\forall\langle\sigma_0\rangle \cdot \exists\langle\sigma_1\rangle \cdot \sigma_0(\mathsf{x}) < \sigma_1(\mathsf{x})$

`y := x + 1`

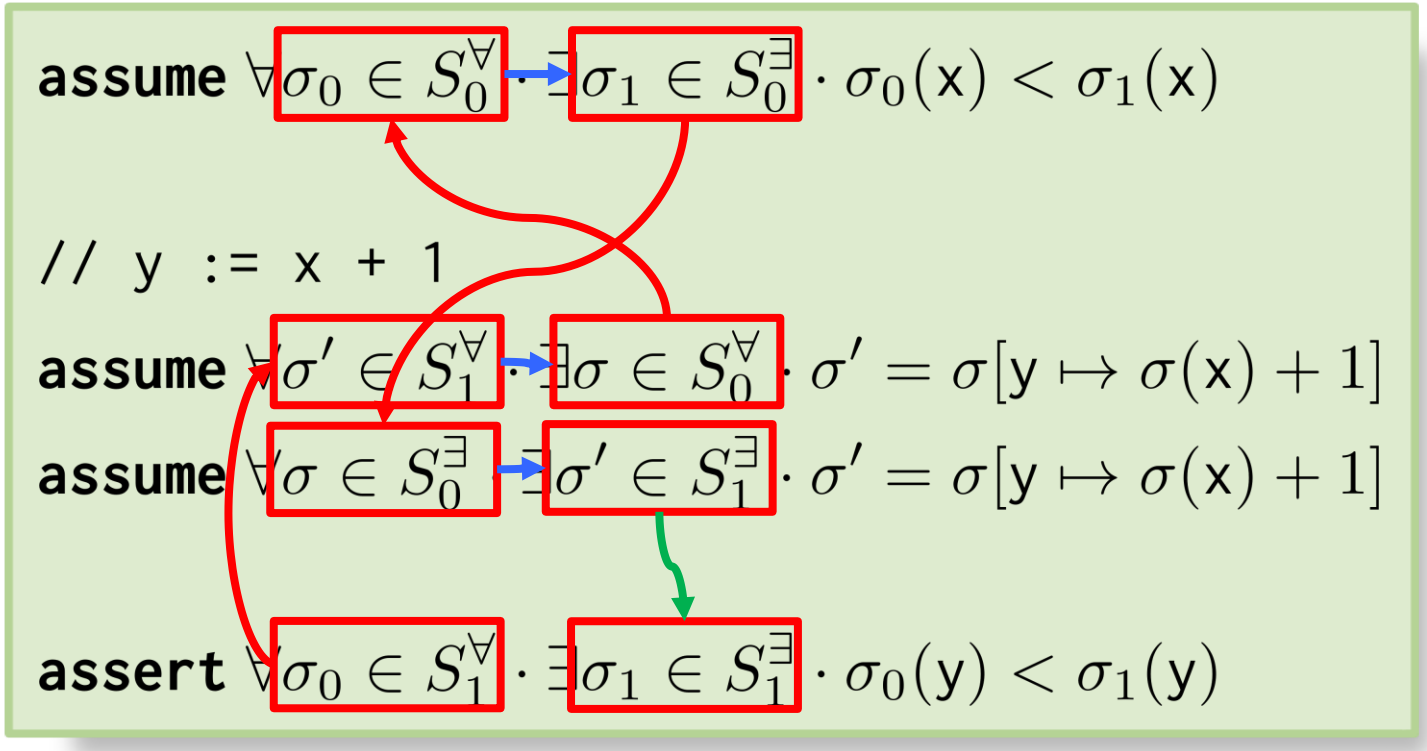
ensures $\forall\langle\sigma_0\rangle \cdot \exists\langle\sigma_1\rangle \cdot \sigma_0(\mathsf{y}) < \sigma_1(\mathsf{y})$

assume $\forall\sigma_0 \in S_0^{\forall} \cdot \exists\sigma_1 \in S_0^{\exists} \cdot \sigma_0(\mathsf{x}) < \sigma_1(\mathsf{x})$

`y := x + 1`

assert $\forall\sigma_0 \in S_1^{\forall} \cdot \exists\sigma_1 \in S_1^{\exists} \cdot \sigma_0(\mathsf{y}) < \sigma_1(\mathsf{y})$

Example


assume $\forall\sigma_0 \in S_0^\forall \cdot \exists\sigma_1 \in S_0^\exists \cdot \sigma_0(\mathsf{x}) < \sigma_1(\mathsf{x})$
// y := x + 1
assume $\forall\sigma' \in S_1^\forall \cdot \exists\sigma \in S_0^\forall \cdot \sigma' = \sigma[\mathsf{y} \mapsto \sigma(\mathsf{x}) + 1]$
assume $\forall\sigma \in S_0^\exists \cdot \exists\sigma' \in S_1^\exists \cdot \sigma' = \sigma[\mathsf{y} \mapsto \sigma(\mathsf{x}) + 1]$
assert $\forall\sigma_0 \in S_1^\forall \cdot \exists\sigma_1 \in S_1^\exists \cdot \sigma_0(\mathsf{y}) < \sigma_1(\mathsf{y})$

Program Properties

Correctness	Absence of bugs	$\forall$
Reachability	Presence of bugs	$\exists$
Determinism	Hash functions	$\forall\forall$
Transitivity	Comparators	$\forall\forall\forall$
Non-interference	Secure information flow	$\forall\forall$
Generalized non-interference	Secure information flow	$\forall\forall\exists$
Existence of a minimum	Optimization algorithms	$\exists\forall$

Hyper-Triples

$$\{\,\mathbf{P}\,\}\ C\ \{\,\mathbf{Q}\,\}$$

Properties over sets of reachable states

$$\vDash \{\,\mathbf{P}\,\}\ C\ \{\,\mathbf{Q}\,\} \ \Leftrightarrow\ \forall S \cdot \mathbf{P}(S) \Rightarrow \mathbf{Q}(\mathsf{sem}(C,S))$$

$$\mathsf{sem}(C,S) \ \Leftrightarrow\ \{\sigma' \mid \exists \sigma \in S \cdot \langle C,\sigma\rangle \to \sigma'\}$$

Rules for Synchronized Loops

- For potentially non-terminating loops

$$\frac{\mathbf{I} \models \mathtt{low(b)} \qquad \vdash \{\,\mathbf{I} \wedge \Box \mathtt{b}\,\}\ C\ \{\,\mathbf{I}\,\}}{\vdash \{\,\mathbf{I}\,\}\ \mathbf{while}\ (\mathtt{b})\ \{C\}\ \{\,(\mathbf{I} \vee \mathtt{emp}) \wedge \Box\neg \mathtt{b}\,\}}$$

$$\mathtt{low(b)} \equiv \forall\langle\sigma\rangle,\langle\sigma'\rangle \cdot \mathtt{b}(\sigma) = \mathtt{b}(\sigma')$$
$$\mathtt{emp} \equiv \forall\langle\sigma\rangle \cdot \mathtt{false}$$

- For terminating loops

$$\frac{\mathbf{I} \models \mathtt{low(b)} \qquad \vdash_{\Downarrow} \{\,\mathbf{I} \wedge \Box(\mathtt{b} \wedge \mathtt{d} = \mathtt{D})\,\}\ C\ \{\,\mathbf{I} \wedge \Box(\mathtt{d} < \mathtt{D})\,\}}{\vdash_{\Downarrow} \{\,\mathbf{I}\,\}\ \mathbf{while}\ (\mathtt{b})\ \{C\}\ \{\,\mathbf{I} \wedge \Box\neg \mathtt{b}\,\}}$$

d is the ranking function
< is well-founded
D is a fresh logical variable

Over- and Under-Approximating Reachable States

- We track separately a lower bound $S^\forall$ and an upper bound $S^\exists$ on the set of states
$$S^\forall \subseteq S \subseteq S^\exists$$
- We do not assume in general that they are equal
- This encoding eliminates matching loops

```
var S0∀ : Set[Map[Var,Int]]
var S0∃ : Set[Map[Var,Int]]

// havoc x

var S1∀ : Set[Map[Var,Int]]
var S1∃ : Set[Map[Var,Int]]

assume ∀σ' ∈ S1∀ ⇒ ∃σ ∈ S0∀, v · σ' = σ[x ↦ v]
assume ∀σ ∈ S0∃, v · ∃σ' ∈ S1∃ · σ' = σ[x ↦ v]

assert ∀σ1 ∈ S1∀ · σ1(x) = 1
```