Contents

State Dependency of Formula Evaluation

Closed FOL formula is either valid or not wrt model $\mathcal{M}$

Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ to be static part of snapshot, ie state

Let $\mathtt{x}$ be program (local) variable or attribute

Execution of program $\mathtt{p}$ may change state, ie value of $\mathtt{x}$

State Dependency of Formula Evaluation

Closed FOL formula is either valid or not wrt model $\mathcal{M}$

Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ to be static part of snapshot, ie state

Let $\mathtt{x}$ be program (local) variable or attribute

Execution of program $\mathtt{p}$ may change state, ie value of $\mathtt{x}$

Example

Executing $\mathtt{x} = 3;$ results in $\mathcal{M}$ such that $\mathcal{M} \models \mathrm{x} \doteq 3$

Executing $\mathtt{x} = 4;$ results in $\mathcal{M}$ such that $\mathcal{M} \not\models \mathrm{x} \doteq 3$

State Dependency of Formula Evaluation

Closed FOL formula is either valid or not wrt **model** $\mathcal{M}$

Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ to be static part of **snapshot**, ie **state**

Let $\mathtt{x}$ be program (local) variable or attribute

Execution of program $\mathtt{p}$ may change state, ie value of $\mathtt{x}$

Example

Executing $\mathtt{x} = 3;$ results in $\mathcal{M}$ such that $\mathcal{M} \models \mathtt{x} \doteq 3$

Executing $\mathtt{x} = 4;$ results in $\mathcal{M}$ such that $\mathcal{M} \not\models \mathtt{x} \doteq 3$

Need a logic to capture state before/after program execution

Rigid versus Flexible Symbols

Signature of program logic defined as in FOL, but:

In addition there are program variables, attributes, etc.

Rigid versus Flexible

Rigid versus Flexible Symbols

Signature of program logic defined as in FOL, **but**:

In addition there are program variables, attributes, etc.

Rigid versus Flexible

- **Rigid** symbols, same interpretation in **all** execution states

  Needed, for example, to hold initial value of program variable

  Logical variables and **built-in functions/predicates** are rigid

Rigid versus Flexible Symbols

Signature of program logic defined as in FOL, but:

In addition there are program variables, attributes, etc.

Rigid versus Flexible

- **Rigid symbols, same interpretation in all execution states**

  Needed, for example, to hold initial value of program variable

  Logical variables and built-in functions/predicates are rigid

- **Non-rigid (or flexible) symbols, interpretation depends on state**

  Needed to capture state change after program execution

  Functions modeling program variables and attributes are flexible

Signature of Dynamic Logic (Simple Version)

Given type hierarchy $\mathcal{T}_q = \{\texttt{int}, \texttt{boolean}, \top\}$

Signature $\Sigma = ($**VSym**, **PSym**, **FSym**, **PVSym**, $\alpha)$

Variable Symbols	**VSym** $= \{x_i \mid i \in I\!N\}$
Rigid Predicate Symbols	**PSym**$_r = \{>, >=, \ldots,\}$
Rigid Function Symbols	**FSym**$_r = \{+, -, *, \mathtt{0}, \mathtt{1}, \mathtt{TRUE}, \mathtt{FALSE}\}$
Non-rigid Function Symbols	**FSym**$_{nr} = \{i, j, k, \ldots, p, q, r, \ldots\}$

Signature of Dynamic Logic (Simple Version)

Given type hierarchy $\mathcal{T}_q = \{\texttt{int}, \texttt{boolean}, \top\}$

Signature $\Sigma =$ (**VSym**, **PSym**, **FSym**, **PVSym**, α)

Variable Symbols	**VSym** $= \{x_i \mid i \in \mathbb{N}\}$
Rigid Predicate Symbols	**PSym**$_r = \{>, >=, \ldots, \}$
Rigid Function Symbols	**FSym**$_r = \{+, -, *, \texttt{0}, \texttt{1}, \texttt{TRUE}, \texttt{FALSE}\}$
Non-rigid Function Symbols	**FSym**$_{nr} = \{i, j, k, \ldots, p, q, r, \ldots\}$

Typing function α **for all symbols:**

- $\alpha(\texttt{j}) \in \{\texttt{int}, \texttt{boolean}\}$ **for all** $\texttt{j} \in$ **FSym**$_{nr}$

  When $b : \rightarrow \texttt{boolean}$, **write** $\texttt{boolean b}$, **etc.;, use as program variable**

- **Standard typing for rigid function/predicate symbols**

  For example, $\texttt{TRUE} : \rightarrow \texttt{boolean}$, $\quad > : \texttt{int}, \texttt{int}$

Terms

First-order terms may contain rigid and non-rigid symbols

Different syntactic categories: $\mathsf{FSym}_r \cap \mathsf{FSym}_{nr} = \emptyset$

Program variables are non-rigid (=flexible) constants

Emphasize distinction to variables VSym: call them logical variables

A term containing at least one flexible symbol is flexible, otherwise rigid

Terms

First-order terms may contain rigid and non-rigid symbols

Different syntactic categories: FSym$_r$ $\cap$ FSym$_{nr}$ $= \emptyset$

Program variables are non-rigid (=flexible) constants

Emphasize distinction to variables VSym: call them logical variables

A term containing at least one flexible symbol is flexible, otherwise rigid

Examples

VSym $= \{x : \mathtt{int}, b : \mathtt{boolean}\}$
FSym$_{nr}$ $= \{\mathtt{int}\ \mathtt{j}, \mathtt{boolean}\ \mathtt{p}\}$

Well-formed terms: $\mathtt{j}+x$, $\mathtt{j}$, b
Ill-formed terms: $\mathtt{j}+b$, $\mathtt{j}+\mathtt{p}$

Atomic Programs

Atomic Programs Π_0

- **Assignments** $\mathtt{j} = t$ **with:**
  $\mathtt{z}\ \mathtt{j} \in$ **FSym**$_{nr}$**,** $\quad t$ **term of type** $\mathtt{z}$ **without logical variables**

Atomic Programs

Atomic Programs Π_0

- **Assignments** $\mathtt{j} = t$ **with:**
  $\mathtt{z}\ \mathtt{j} \in$ **FSym**$_{nr}$**,** $\quad t$ **term of type** $\mathtt{z}$ **without logical variables**

Examples

VSym $= \{x : \mathtt{int}, b : \mathtt{boolean}\}$
FSym$_{nr}$ $= \{\mathtt{int}\ \mathtt{j}, \mathtt{boolean}\ \mathtt{p}\}$

Well-formed atomic programs: $\mathtt{j} = \mathtt{j} + 1$**,** $\mathtt{j} = 0$**,** $\mathtt{p} = \mathtt{FALSE}$

Ill-formed atomic programs: $\mathtt{j} = \mathtt{j} + x$**,** $x = 1$**,** $\mathtt{j} \doteq \mathtt{j}$**,** $\mathtt{p} = 0$

Dynamic Logic (Simple Version) Programs

Programs Π

- If π is an atomic program, then $\pi;$ is a program
- If p and q are programs, then pq is a program
- If b is a variable-free term of type `boolean`, p and q programs, then

  if (b) {p} else {q};

  is a program
- If b is a variable-free term of type `boolean`, p a program, then

  while (b) {p};

  is a program

Programs contain no logical variables

Dynamic Logic Syntax Example

Given signature

PSym$_r = \{<\}$

FSym$_r = \{0, +, -\}$

FSym$_{nr} = \{\mathtt{int}\ \mathtt{i}, \mathtt{int}\ \mathtt{r}, \mathtt{int}\ \mathtt{n}\}$

An admissible DL program $\mathtt{p}$**:**

```
i=0;
r=0;
while (i<n) {
  i=i+1;
  r=r+i;
};
r=r+r-n;
```

What does $\mathtt{p}$ **compute?**

Dynamic Logic (Simple Version) Formulas

Dynamic Logic Formulas (DL Formulas)

- **Each FOL formula is a DL formula**

  DL formulas closed under FOL operators and connectives

- **If p is a program and ϕ a DL formula then**

  $\langle \mathrm{p} \rangle \phi$ **is a DL formula**

  $[\mathrm{p}]\,\phi$ **is a DL-Formula**

Program variables are constants: never bound in quantifiers

Programs contain no logical variables

The operators $\langle\ \rangle$ and $[\ ]$ can be arbitrarily nested

Dynamic Logic Syntax Example

Check for syntactic well-formedness and derive the signature

$\forall y. ((\langle \mathtt{x} = 1; \rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{x} = 1 * 1; \rangle \mathtt{x} \doteq y))$ **Syntax ?**

Dynamic Logic Syntax Example

Check for syntactic well-formedness and derive the signature

$\forall y. ((\langle \mathtt{x = 1;} \rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{x = 1 * 1;} \rangle \mathtt{x} \doteq y))$ **ok** **(**$y : \mathtt{int}$**)**

Dynamic Logic Syntax Example

Check for syntactic well-formedness and derive the signature

$\forall y. ((\langle \mathtt{x} = \mathtt{1};\rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{x} = \mathtt{1} * \mathtt{1};\rangle \mathtt{x} \doteq y))$ **ok (**$y : \mathtt{int}$**)**

$\exists \mathtt{x}. ([\mathtt{x} = \mathtt{1};] (\mathtt{x} \doteq 1))$ **Syntax ?**

Dynamic Logic Syntax Example

Check for syntactic well-formedness and derive the signature

$\forall y. ((\langle \mathtt{x} = \mathtt{1};\rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{x} = \mathtt{1} * \mathtt{1};\rangle \mathtt{x} \doteq y))$ **ok (y : $\mathtt{int}$)**

$\exists \mathtt{x}. ([\mathtt{x} = \mathtt{1};] (\mathtt{x} \doteq 1))$ **bad**

- x **cannot be logical variable, because it occurs in program**
- x **cannot be program variable, because it is quantified**

Dynamic Logic Syntax Example

Check for syntactic well-formedness and derive the signature

$\forall y. ((\langle \mathtt{x} = 1; \rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{x} = 1 * 1; \rangle \mathtt{x} \doteq y))$ **ok** **(**$y : \mathtt{int}$**)**

$\exists \mathtt{x}. ([\mathtt{x} = 1;] (\mathtt{x} \doteq 1))$ **bad**

- x **cannot be logical variable, because it occurs in program**
- x **cannot be program variable, because it is quantified**

$\langle \mathtt{x} = 1; \rangle ([\mathbf{while}\ (\mathbf{true})\ \mathbf{\{\}}]\ \mathbf{false})$ **Syntax ?**

Dynamic Logic Syntax Example

Check for syntactic well-formedness and derive the signature

$\forall y. (((\langle \mathtt{x} = 1; \rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{x} = 1 * 1; \rangle \mathtt{x} \doteq y))$ **ok** **(**$y : \mathtt{int}$**)**

$\exists \mathtt{x}. ([\mathtt{x} = 1;] (\mathtt{x} \doteq 1))$ **bad**

- x **cannot be logical variable, because it occurs in program**
- x **cannot be program variable, because it is quantified**

$\langle \mathtt{x} = 1; \rangle ([\textbf{while}\ (\textbf{true})\ \textbf{\{\}}]\ \textbf{false})$ **ok** **(**$\mathtt{int\ x}$**)**

- **Program formulas can appear nested**

More Examples of DL Formulas

1. $\mathtt{x} \doteq i$ **&** $\mathtt{y} \doteq j$ **->** $\langle \mathtt{z} = \mathtt{x}; \mathtt{x} = \mathtt{y}; \mathtt{y} = \mathtt{x}; \rangle \mathtt{x} \doteq j$ **&** $\mathtt{y} \doteq i$

2. $\mathtt{x} \doteq 3$ **|** $\mathtt{y} \doteq -2$ **->** $\langle \mathtt{y} = \mathtt{x} * \mathtt{x} - \mathtt{x} + 6; \rangle \mathtt{y} \doteq 0$

3. $\langle$**if** 0 **<=** $\mathtt{a}$ **then {} else {**$\mathtt{a} = -\mathtt{a};$ **}**$\rangle 0$ **<=** $\mathtt{a}$

4. $\langle$**while** $(\mathtt{c}$ **<=** $\mathtt{n} - 1)$ **{**$\mathtt{p} = \mathtt{p} + \mathtt{m}; \mathtt{c} = \mathtt{c} + 1;$ **}**$\rangle$ $\mathtt{p} \doteq \mathtt{m} * \mathtt{m}$

Dynamic Logic Semantics: States

First-order model can be considered as (execution) state

Interpretation of non-rigid symbols can vary from state to state

(eg, program variables)

Interpretation of rigid symbols is the same in all states

(eg, built-in functions and predicates)

State = First-order model:

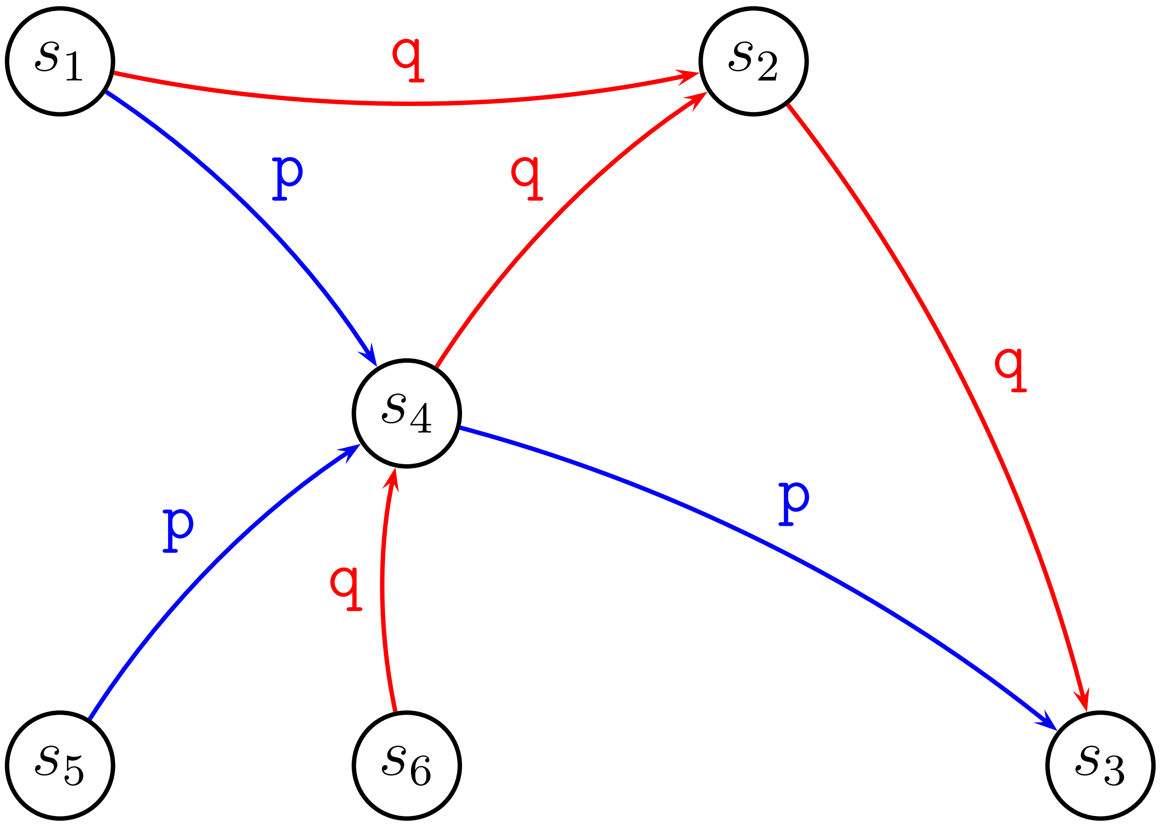
$\mathcal{M} = s = (\mathcal{D}, \delta, \mathcal{I})$ over FSym $=$ FSym$_r \cup$ FSym$_{nr}$

Set of all states s is S

Dynamic Logic Semantics: Kripke Structure

Kripke structure $K = (S, \rho)$

State (model) $s = (\mathcal{D}, \delta, \mathcal{I}) \in S$ **and** $\rho : \Pi \to (S \to S)$ $\quad \rho(\mathtt{p}),\ \rho(\mathtt{q})$

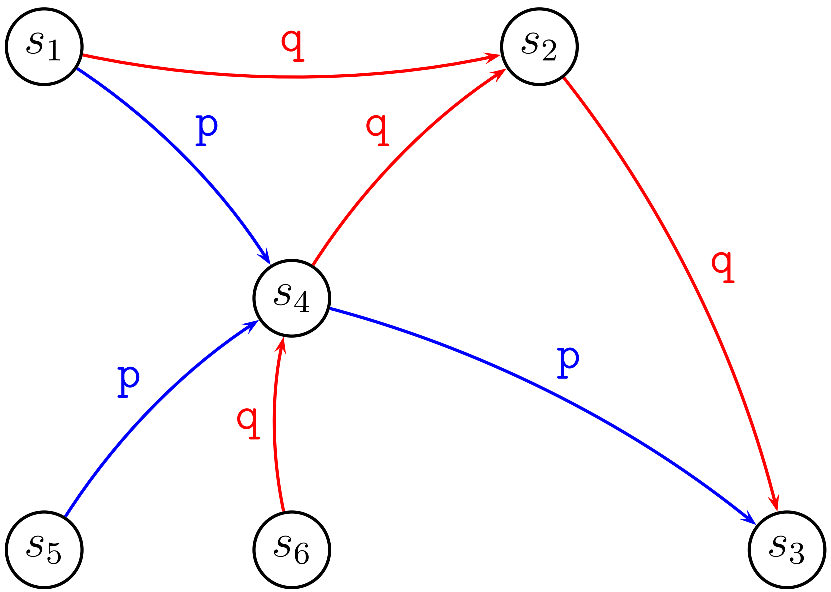


Each state is first-order model $s = (\mathcal{D}, \delta, \mathcal{I})$ **over same domain** $\mathcal{D}$

Dynamic Logic Semantics: Program Formulas

- $s, \beta \models \langle \mathtt{p} \rangle \phi$ **iff** $\rho(\mathtt{p})(s), \beta \models \phi$ **and** $\rho(\mathtt{p})(s)$ **defined**

  $\mathtt{p}$ ***terminates and*** ϕ ***is true in the final state after execution***



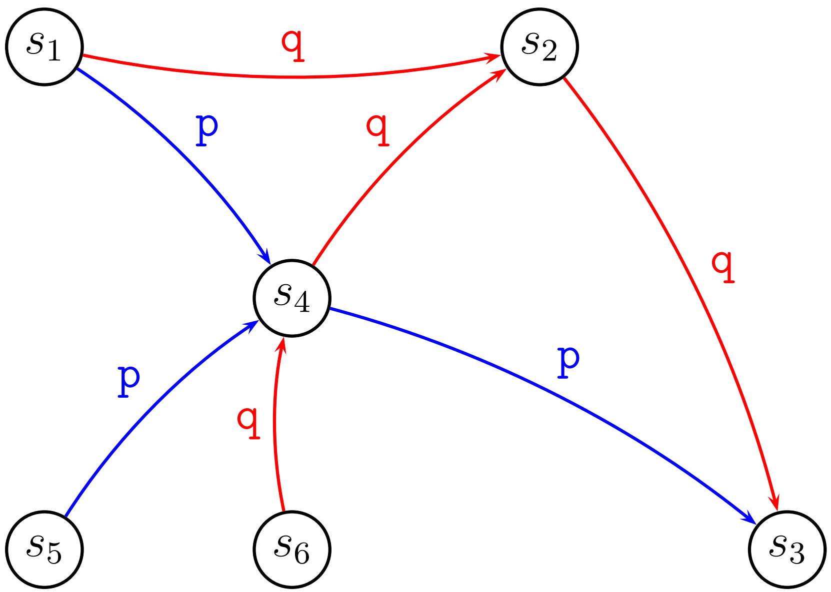
Dynamic Logic Semantics: Program Formulas

- $s, \beta \models \langle \mathtt{p} \rangle \phi$ **iff** $\rho(\mathtt{p})(s), \beta \models \phi$ **and $\rho(\mathtt{p})(s)$ defined**

  $\mathtt{p}$ ***terminates and ϕ is true in the final state after execution***

- $s, \beta \models [\mathtt{p}]\, \phi$ **iff** $\rho(\mathtt{p})(s), \beta \models \phi$ **whenever $\rho(\mathtt{p})(s)$ defined**

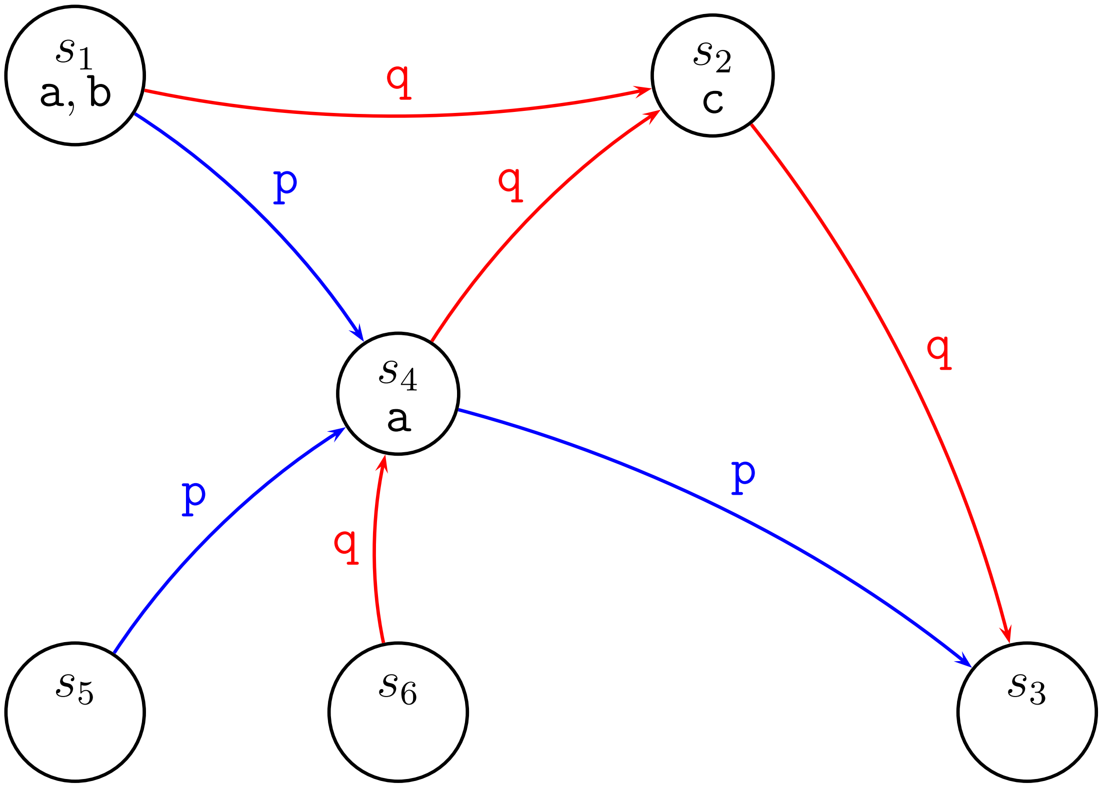
  If $\mathtt{p}$ ***terminates then ϕ is true in the final state after execution***



Dynamic Logic Semantics Example

Boolean program variables

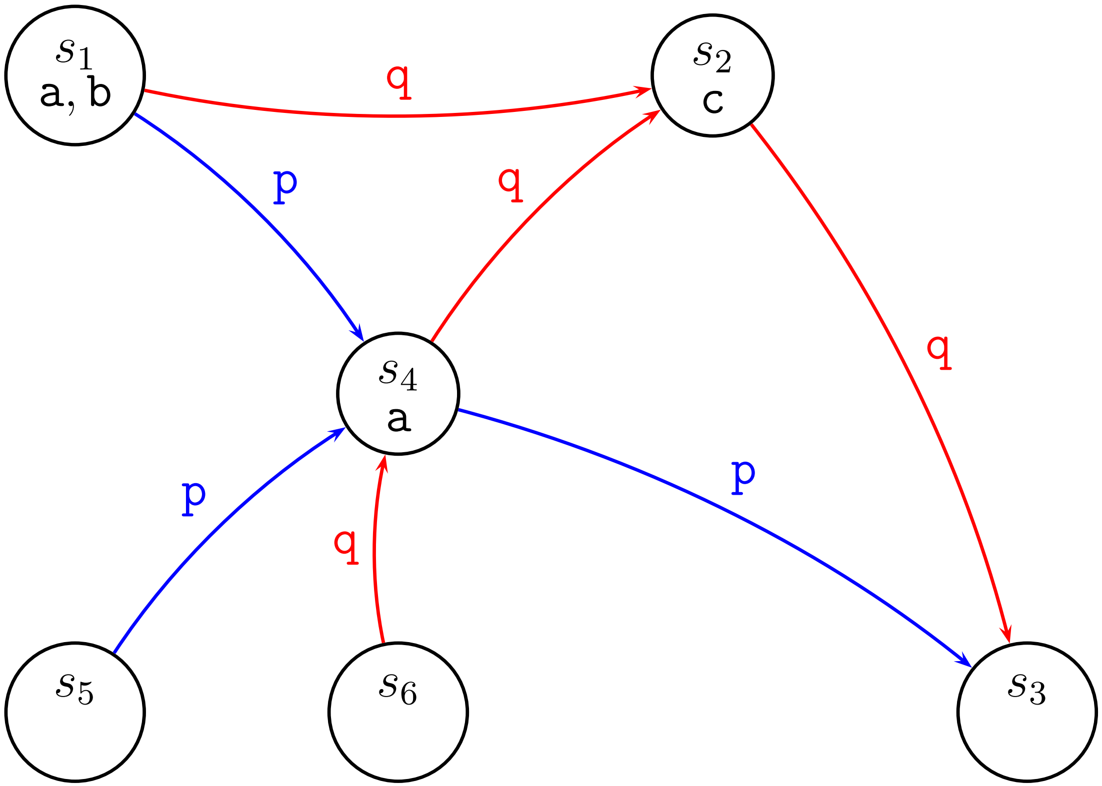
$\textbf{FSym}_{nr} = \{\texttt{boolean a}, \texttt{boolean b}, \texttt{boolean c}, \ldots\}$



Dynamic Logic Semantics Example

Boolean program variables

FSym$_{nr} = \{$`boolean a`$,$ `boolean b`$,$ `boolean c`$, \ldots\}$

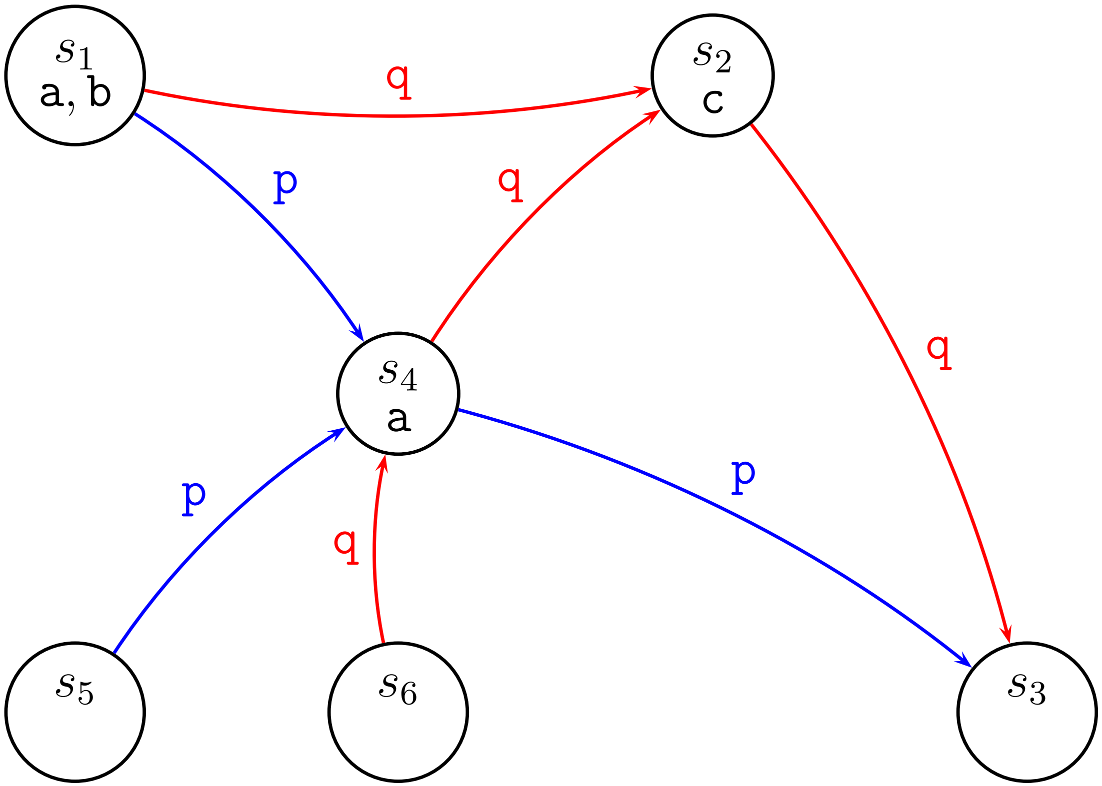


$s_1 \models \langle \mathtt{p} \rangle \mathtt{a} \doteq \mathtt{TRUE}$ **?**

Dynamic Logic Semantics Example

Boolean program variables

FSym$_{nr} = \{$`boolean a`, `boolean b`, `boolean c`$, \ldots\}$

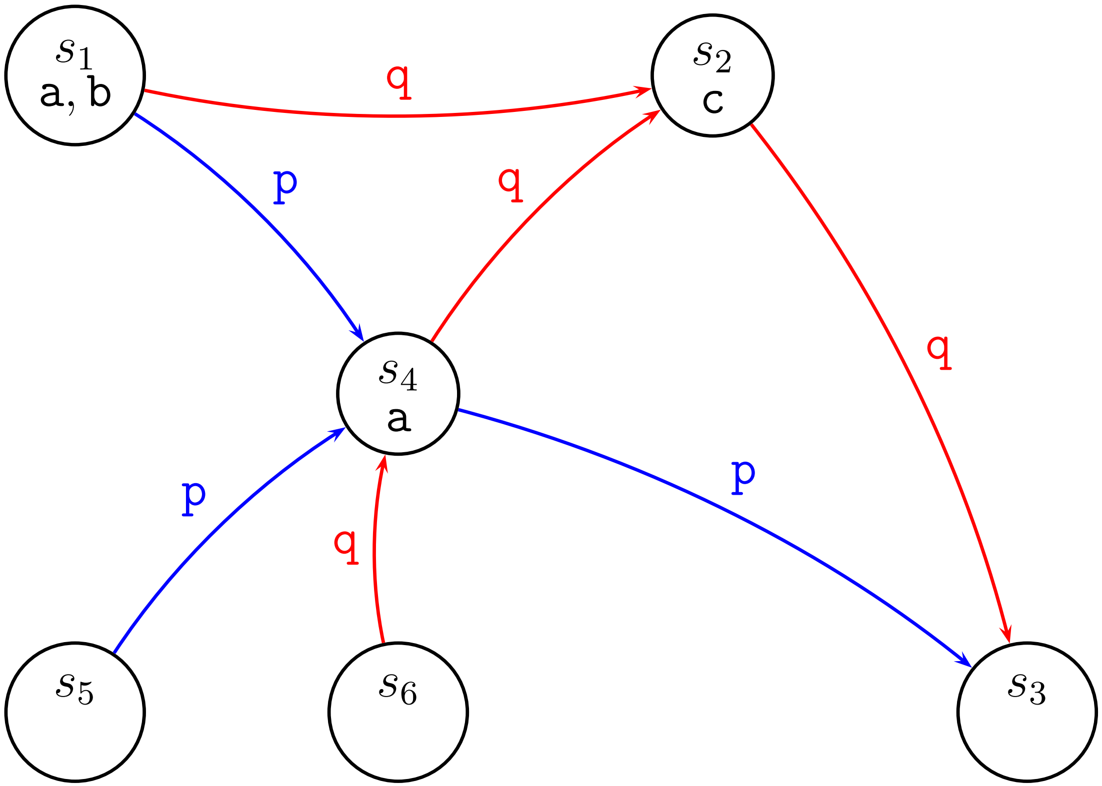


$s_1 \models \langle \mathtt{p} \rangle \mathtt{a} \doteq \mathtt{TRUE}$ **(ok),**

Dynamic Logic Semantics Example

Boolean program variables

FSym$_{nr} = \{\texttt{boolean a}, \texttt{boolean b}, \texttt{boolean c}, \ldots\}$

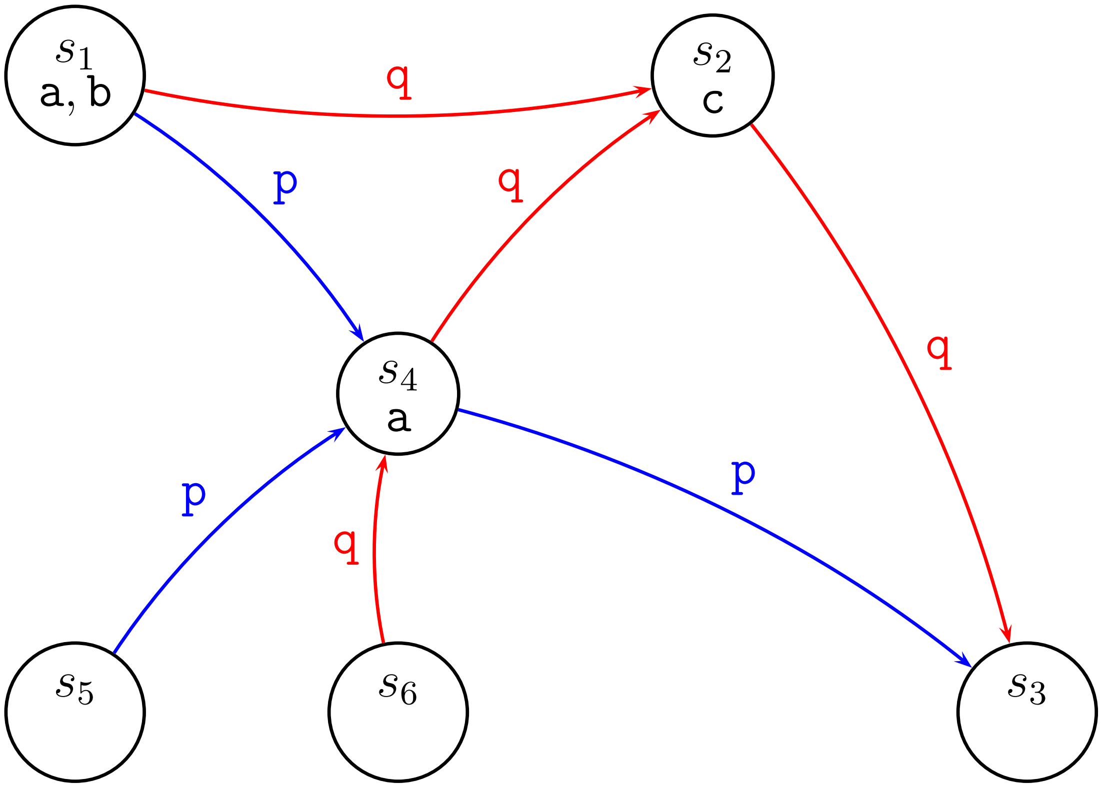


$s_1 \models \langle \texttt{p} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(ok),** $\qquad s_1 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **?**

Dynamic Logic Semantics Example

Boolean program variables

FSym$_{nr} = \{\texttt{boolean a}, \texttt{boolean b}, \texttt{boolean c}, \ldots\}$

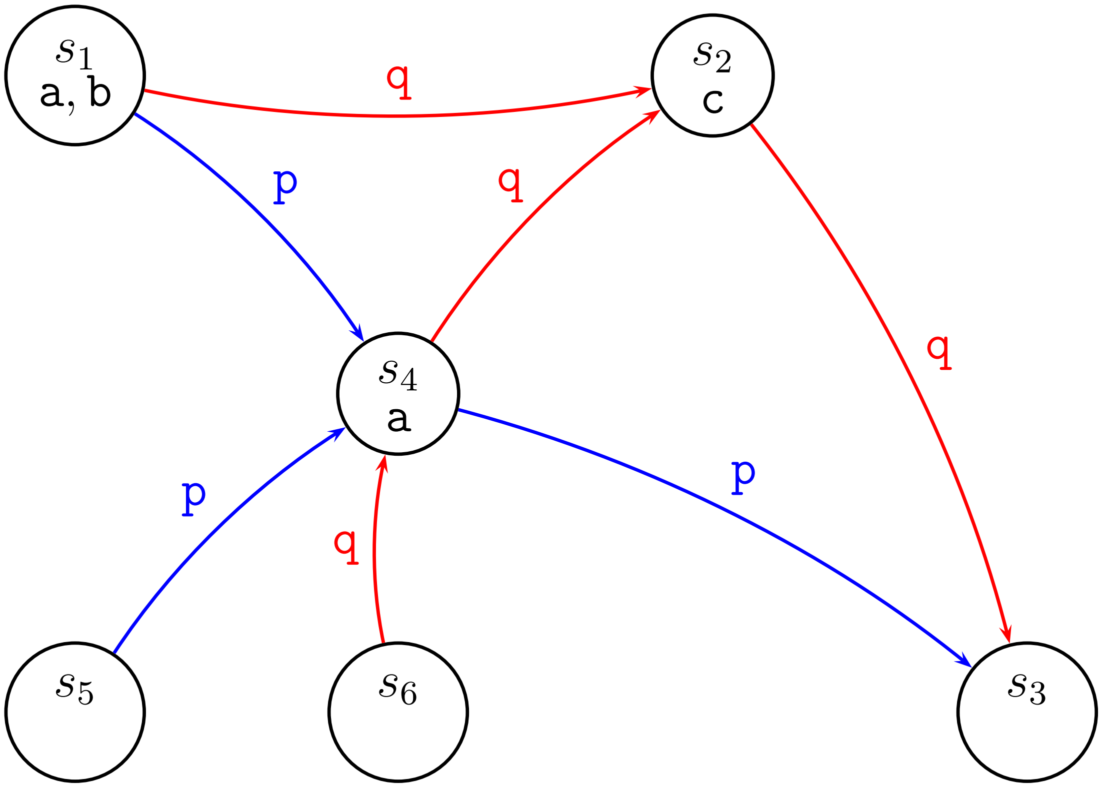


$s_1 \models \langle \texttt{p} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(ok),** $s_1 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—)**

Dynamic Logic Semantics Example

Boolean program variables

FSym$_{nr} = \{\texttt{boolean a}, \texttt{boolean b}, \texttt{boolean c}, \ldots\}$



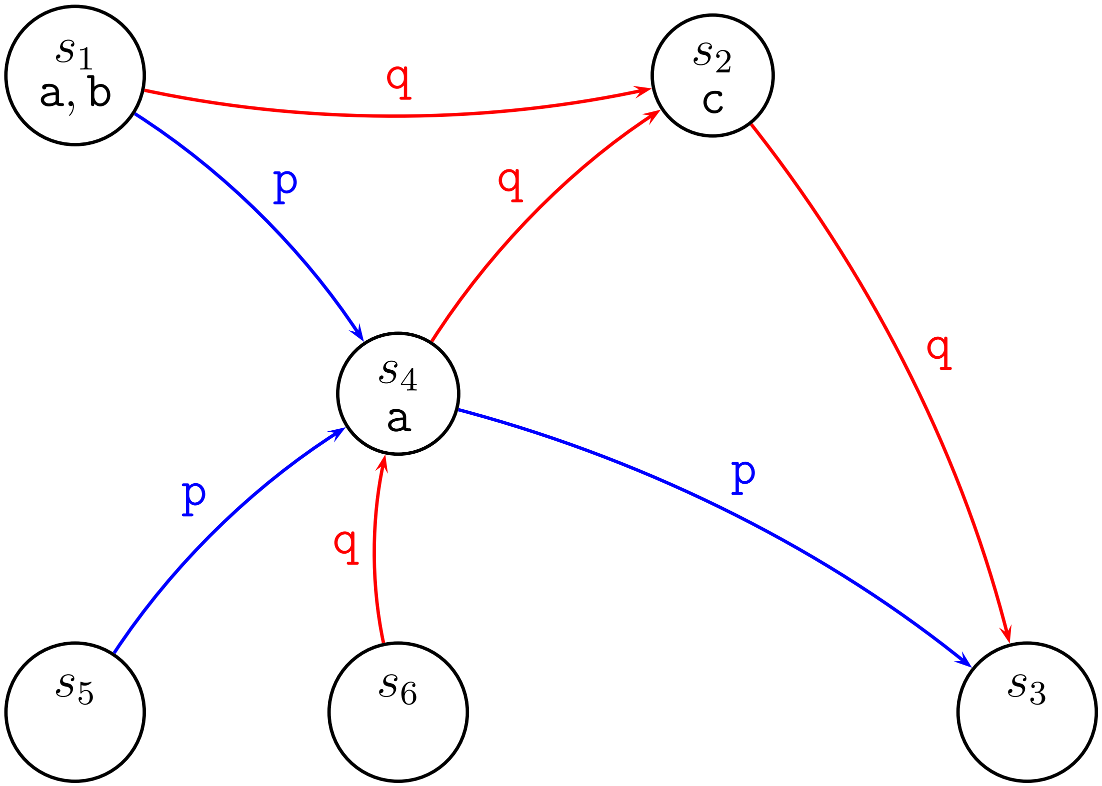
$s_1 \models \langle \texttt{p} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(ok),** $s_1 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—)**

$s_5 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **?**

Dynamic Logic Semantics Example

Boolean program variables

FSym$_{nr} = \{\texttt{boolean a}, \texttt{boolean b}, \texttt{boolean c}, \ldots\}$



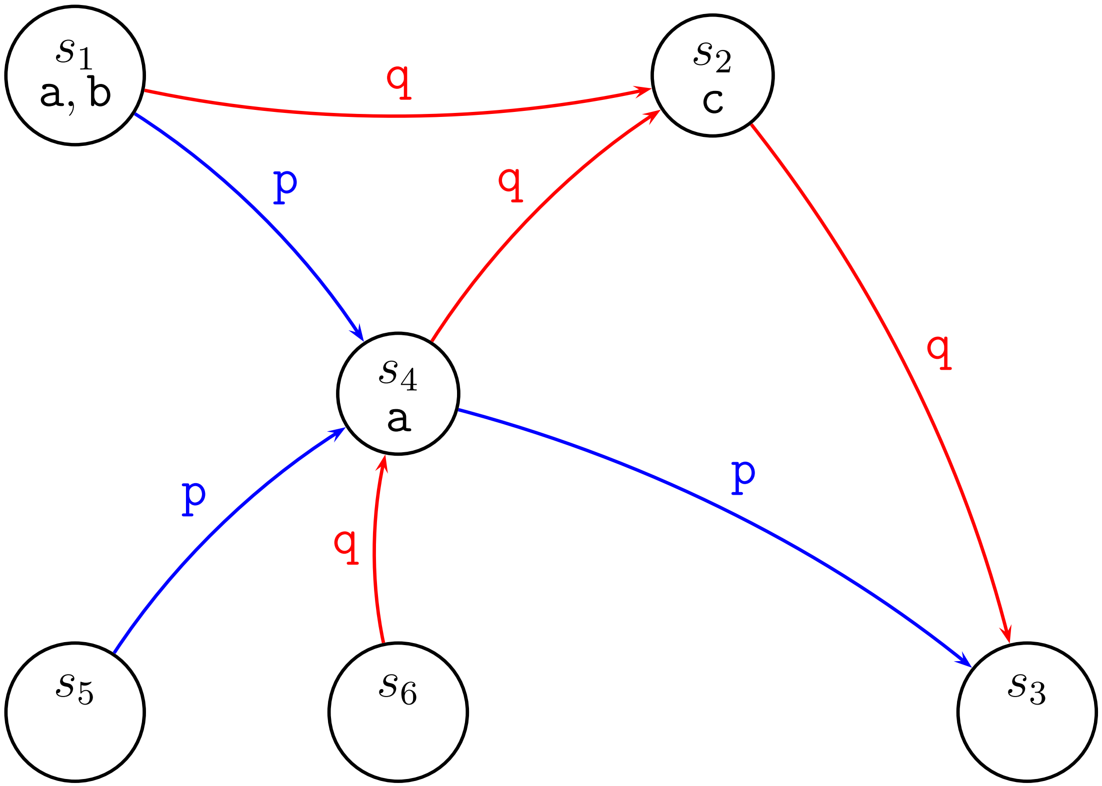
$s_1 \models \langle \texttt{p} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(ok),** $s_1 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—)**

$s_5 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—),**

Dynamic Logic Semantics Example

Boolean program variables

$\textbf{FSym}_{nr} = \{\texttt{boolean a}, \texttt{boolean b}, \texttt{boolean c}, \ldots\}$



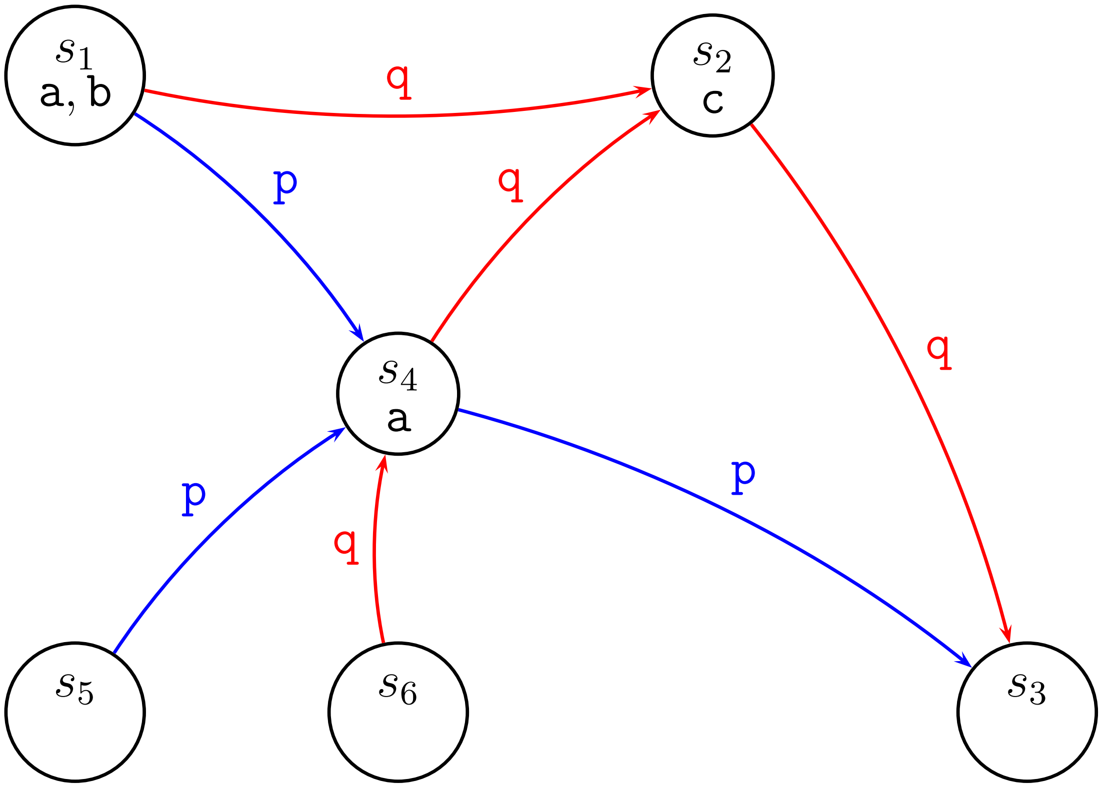
$s_1 \models \langle \texttt{p} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(ok),** $s_1 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—)**

$s_5 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—),** $s_5 \models [\texttt{q}]\, \texttt{a} \doteq \texttt{TRUE}$**?**

Dynamic Logic Semantics Example

Boolean program variables

$\textbf{FSym}_{nr} = \{\texttt{boolean a}, \texttt{boolean b}, \texttt{boolean c}, \ldots\}$



$s_1 \models \langle \texttt{p} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(ok),** $s_1 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—)**

$s_5 \models \langle \texttt{q} \rangle \texttt{a} \doteq \texttt{TRUE}$ **(—),** $s_5 \models [\texttt{q}]\, \texttt{a} \doteq \texttt{TRUE}$ **(ok)**

Program Correctness

- $s, \beta \models \langle \mathtt{p} \rangle \phi$

  $\mathtt{p}$ **totally correct (with respect to ϕ) in s, β**

Program Correctness

- $s, \beta \models \langle \mathtt{p} \rangle \phi$

  $\mathtt{p}$ **totally correct (with respect to ϕ) in s, β**

- $s, \beta \models [\mathtt{p}]\, \phi$

  $\mathtt{p}$ **partially correct (with respect to ϕ) in s, β**

Program Correctness

- $s, \beta \models \langle \mathtt{p} \rangle \phi$

  $\mathtt{p}$ **totally correct (with respect to ϕ) in s, β**

- $s, \beta \models [\mathtt{p}]\,\phi$

  $\mathtt{p}$ **partially correct (with respect to ϕ) in s, β**

- **Duality** $\langle \mathtt{p} \rangle \phi$ **iff** **!** $[\mathtt{p}]$ **!**ϕ

  Exercise: justify this with semantic definitions

Program Correctness

- $s, \beta \models \langle \mathtt{p} \rangle \phi$

  $\mathtt{p}$ **totally correct (with respect to ϕ) in s, β**

- $s, \beta \models [\mathtt{p}]\, \phi$

  $\mathtt{p}$ **partially correct (with respect to ϕ) in s, β**

- **Duality** $\langle \mathtt{p} \rangle \phi$ **iff** $!\, [\mathtt{p}]\, !\phi$

  Exercise: justify this with semantic definitions

- **Implication** **if** $\langle \mathtt{p} \rangle \phi$ **then** $[\mathtt{p}]\, \phi$

  Total correctness implies partial correctness
  (holds only for deterministic programs)

Semantics of Sequents

Let $\Gamma = \{\phi_1, \ldots, \phi_n\} \subseteq$ **For and** $\Delta = \{\psi_1, \ldots, \psi_m\} \subseteq$ **For**

Recall: $s \models (\Gamma$ **==>** $\Delta)$ **iff** $s \models (\phi_1$**&**$\cdots$**&**$\phi_n)$ **->** $(\psi_1$**|**$\cdots$**|**$\psi_m)$

Semantics of DL sequents should be defined identically with semantics of FOL sequents (assume Γ, Δ are sets of closed DL formulas):

Γ **==>** Δ **is valid** **iff** $s \models (\Gamma$ **==>** $\Delta)$ **in all states** s

Semantics of Sequents

Let $\Gamma = \{\phi_1, \ldots, \phi_n\} \subseteq$ For and $\Delta = \{\psi_1, \ldots, \psi_m\} \subseteq$ For

Recall: $s \models (\Gamma$ **==>** $\Delta)$ **iff** $s \models (\phi_1$**&**$\cdots$**&**$\phi_n)$ **->** $(\psi_1$**|**$\cdots$**|**$\psi_m)$

Semantics of DL sequents should be defined identically with semantics of FOL sequents (assume Γ, Δ are sets of closed DL formulas):

Γ **==>** Δ **is valid** **iff** $s \models (\Gamma$ **==>** $\Delta)$ **in all states** s

Consequence for program variables

In valid formulas they represent any possible value of their type

Initial States

How to restrict validity to set of initial states $S_0 \subseteq S$?

1. **Design closed FOL formula Init with**

   $s \models \mathbf{Init}$ **iff** $s \in S_0$

2. **Use sequent** $\Gamma, \mathbf{Init}$ **==>** Δ

Later: simple method for specifying initial value of program variables

Dynamic Logic Semantics: States, Updates

- **States $s = (\mathcal{D}, \delta, \mathcal{I})$ all have**
  - **the same domain $\mathcal{D}$ (all objects present from start)**
  - **the same typing function δ (dynamic type never changes)**

  May assume $\rho(\mathtt{p})$ works on interpretations $\mathcal{I}$

  Define $\mathcal{I}, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (\mathcal{D}, \delta, \mathcal{I})$

- **Program variables $\mathtt{j}$ as flexible constants in s with value $\mathcal{I}(\mathtt{j})$**

Dynamic Logic Semantics: States, Updates

- States $s = (\mathcal{D}, \delta, \mathcal{I})$ all have
  - the same domain $\mathcal{D}$ (all objects present from start)
  - the same typing function δ (dynamic type never changes)

  May assume $\rho(\mathtt{p})$ works on interpretations $\mathcal{I}$

  Define $\mathcal{I}, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (\mathcal{D}, \delta, \mathcal{I})$
- Program variables $\mathtt{j}$ as flexible constants in s with value $\mathcal{I}(\mathtt{j})$

Modified state update of $\mathcal{I}$ at $\mathtt{j}$ of type $\mathtt{z}$ with $d \in \mathcal{D}^{\mathtt{z}}$

$$\mathcal{I}_{\mathtt{j}}^{d}(\mathtt{x}) = \begin{cases} \mathcal{I}(\mathtt{x}) & \mathtt{x} \neq \mathtt{j} \\ d & \mathtt{x} = \mathtt{j} \end{cases}$$

Cf. modified variable assignment

Operational Semantics of Programs

State transformation ρ defines semantics of programs

Same ρ for all programs, so not part of s

- $\rho(\mathtt{x\ =\ t;})(\mathcal{I}) = \mathcal{I}_{\mathtt{x}}^{val_{\mathcal{I},\beta}(t)}$ **(can ignore β)**

Operational Semantics of Programs

State transformation ρ defines semantics of programs

Same ρ for all programs, so not part of s

- $\rho(\texttt{x = t;})(\mathcal{I}) = \mathcal{I}_{\texttt{x}}^{val_{\mathcal{I},\beta}(t)}$ **(can ignore β)**
- $\rho(\texttt{if } (b) \texttt{ \{p\} else \{q\};})(\mathcal{I}) = \begin{cases} \rho(\texttt{p})(\mathcal{I}) & \mathcal{I} \models b \doteq \texttt{TRUE} \\ \rho(\texttt{q})(\mathcal{I}) & \textbf{otherwise} \end{cases}$

Operational Semantics of Programs

State transformation ρ defines semantics of programs

Same ρ for all programs, so not part of s

- $\rho(\texttt{x = t;})(\mathcal{I}) = \mathcal{I}_{\texttt{x}}^{val_{\mathcal{I},\beta}(t)}$ **(can ignore β)**
- $\rho(\texttt{if } (b) \; \texttt{\{p\} else \{q\};})(\mathcal{I}) = \begin{cases} \rho(\texttt{p})(\mathcal{I}) & \mathcal{I} \models b \doteq \texttt{TRUE} \\ \rho(\texttt{q})(\mathcal{I}) & \textbf{otherwise} \end{cases}$
- $\rho(\texttt{pq})(\mathcal{I}) = \rho(\texttt{q})(\rho(\texttt{p})(\mathcal{I}))$, **if** $\rho(\texttt{p})(\mathcal{I})$ **defined, undefined otherwise**

Operational Semantics of Programs

State transformation ρ defines semantics of programs

Same ρ for all programs, so not part of s

- $\rho(\mathtt{x} \;=\; \mathtt{t;})(\mathcal{I}) = \mathcal{I}_{\mathtt{x}}^{val_{\mathcal{I},\beta}(t)}$ **(can ignore β)**
- $\rho(\mathtt{if}\ (b)\ \mathtt{\{p\}\ else\ \{q\};})(\mathcal{I}) = \begin{cases} \rho(\mathtt{p})(\mathcal{I}) & \mathcal{I} \models b \doteq \mathtt{TRUE} \\ \rho(\mathtt{q})(\mathcal{I}) & \textbf{otherwise} \end{cases}$
- $\rho(\mathtt{pq})(\mathcal{I}) = \rho(\mathtt{q})(\rho(\mathtt{p})(\mathcal{I}))$**, if** $\rho(\mathtt{p})(\mathcal{I})$ **defined, undefined otherwise**
- $\rho(\mathtt{while}\ (b)\ \mathtt{\{p\};})(\mathcal{I}) = \mathcal{I}'$ **iff there are** $\mathcal{I} = \mathcal{I}_0, \ldots, \mathcal{I}_n = \mathcal{I}'$ **such that**

Operational Semantics of Programs

State transformation ρ defines semantics of programs

Same ρ for all programs, so not part of s

- $\rho(\mathtt{x\ =\ t;})(\mathcal{I}) = \mathcal{I}_{\mathtt{x}}^{val_{\mathcal{I},\beta}(t)}$ **(can ignore β)**
- $\rho(\mathtt{if}\ (b)\ \mathtt{\{p\}\ else\ \{q\};})(\mathcal{I}) = \begin{cases} \rho(\mathtt{p})(\mathcal{I}) & \mathcal{I} \models b \doteq \mathtt{TRUE} \\ \rho(\mathtt{q})(\mathcal{I}) & \textbf{otherwise} \end{cases}$
- $\rho(\mathtt{pq})(\mathcal{I}) = \rho(\mathtt{q})(\rho(\mathtt{p})(\mathcal{I}))$**, if $\rho(\mathtt{p})(\mathcal{I})$ defined,** **undefined otherwise**
- $\rho(\mathtt{while}\ (b)\ \mathtt{\{p\};})(\mathcal{I}) = \mathcal{I}'$ **iff there are $\mathcal{I} = \mathcal{I}_0, \ldots, \mathcal{I}_n = \mathcal{I}'$ such that**
  - $\mathcal{I}_j, \beta \models b \doteq \mathtt{TRUE}$ **for $0 \leq j < n$**

Operational Semantics of Programs

State transformation ρ defines semantics of programs

Same ρ for all programs, so not part of s

- $\rho(\mathtt{x\ =\ t;})(\mathcal{I}) = \mathcal{I}_{\mathtt{x}}^{val_{\mathcal{I},\beta}(t)}$ **(can ignore β)**
- $\rho(\mathtt{if}\ (b)\ \mathtt{\{p\}\ else\ \{q\};})(\mathcal{I}) = \begin{cases} \rho(\mathtt{p})(\mathcal{I}) & \mathcal{I} \models b \doteq \mathtt{TRUE} \\ \rho(\mathtt{q})(\mathcal{I}) & \textbf{otherwise} \end{cases}$
- $\rho(\mathtt{pq})(\mathcal{I}) = \rho(\mathtt{q})(\rho(\mathtt{p})(\mathcal{I}))$**, if** $\rho(\mathtt{p})(\mathcal{I})$ **defined, undefined otherwise**
- $\rho(\mathtt{while}\ (b)\ \mathtt{\{p\};})(\mathcal{I}) = \mathcal{I}'$ **iff there are** $\mathcal{I} = \mathcal{I}_0, \ldots, \mathcal{I}_n = \mathcal{I}'$ **such that**
  - $\mathcal{I}_j, \beta \models b \doteq \mathtt{TRUE}$ **for** $0 \leq j < n$
  - $\rho(\mathtt{p})(\mathcal{I}_j) = \mathcal{I}_{j+1}$ **for** $0 \leq j < n$

Operational Semantics of Programs

State transformation ρ defines semantics of programs

Same ρ for all programs, so not part of s

- $\rho(\texttt{x = t;})(\mathcal{I}) = \mathcal{I}_{\texttt{x}}^{val_{\mathcal{I},\beta}(t)}$ **(can ignore β)**
- $\rho(\texttt{if } (b) \ \texttt{\{p\} else \{q\};})(\mathcal{I}) = \begin{cases} \rho(\texttt{p})(\mathcal{I}) & \mathcal{I} \models b \doteq \texttt{TRUE} \\ \rho(\texttt{q})(\mathcal{I}) & \textbf{otherwise} \end{cases}$
- $\rho(\texttt{pq})(\mathcal{I}) = \rho(\texttt{q})(\rho(\texttt{p})(\mathcal{I}))$, **if** $\rho(\texttt{p})(\mathcal{I})$ **defined, undefined otherwise**
- $\rho(\texttt{while } (b) \ \texttt{\{p\};})(\mathcal{I}) = \mathcal{I}'$ **iff there are** $\mathcal{I} = \mathcal{I}_0, \ldots, \mathcal{I}_n = \mathcal{I}'$ **such that**
  - $\mathcal{I}_j, \beta \models b \doteq \texttt{TRUE}$ **for** $0 \leq j < n$
  - $\rho(\texttt{p})(\mathcal{I}_j) = \mathcal{I}_{j+1}$ **for** $0 \leq j < n$
  - $\mathcal{I}_n, \beta \models b \doteq \texttt{FALSE}$ **undefined otherwise**

Proof by Symbolic Program Execution

Need to have rules for program formulas: but which?

What corresponds to top-level connective in sequential program?

Proof by Symbolic Program Execution

Need to have rules for program formulas: but which?

What corresponds to top-level connective in sequential program?

Idea: follow natural program control flow

Proof by Symbolic Program Execution

Need to have rules for program formulas: but which?

What corresponds to top-level connective in sequential program?

Idea: follow natural program control flow

Sound and complete rule for conclusions with main formulas:

$$\langle \xi \mathtt{q} \rangle \phi, \qquad [\xi \mathtt{q}]\, \phi$$

ξ **one single admissible program statement,** $\mathtt{q}$ **remaining program**

Proof by Symbolic Program Execution

Need to have rules for program formulas: but which?

What corresponds to top-level connective in sequential program?

Idea: follow natural program control flow

Sound and complete rule for conclusions with main formulas:

$$\langle \xi \mathtt{q} \rangle \phi, \qquad [\xi \mathtt{q}]\, \phi$$

ξ one single admissible program statement, $\mathtt{q}$ remaining program

Rules execute symbolically the first active statement

Proof corresponds to symbolic program execution

Dynamic Logic Calculus

$$\text{CONCATENATE}\ \frac{\Gamma \quad \texttt{==>} \quad \langle \mathtt{p} \rangle \left(\langle \mathtt{q} \rangle \phi \right), \Delta}{\Gamma \quad \texttt{==>} \quad \langle \mathtt{pq} \rangle \phi, \Delta}$$

Dynamic Logic Calculus

$$\textbf{CONCATENATE}\ \frac{\Gamma \ \ \textbf{==>}\ \ \langle \mathtt{p}\rangle\,(\langle \mathtt{q}\rangle\phi)\,,\Delta}{\Gamma \ \ \textbf{==>}\ \ \langle \mathtt{pq}\rangle\phi,\Delta}$$

$$\textbf{IF}\ \frac{\Gamma, b \doteq \mathtt{TRUE}\ \ \textbf{==>}\ \ \langle \mathtt{p}\rangle\phi,\Delta \qquad\qquad \Gamma, b \doteq \mathtt{FALSE}\ \ \textbf{==>}\ \ \langle \mathtt{q}\rangle\phi,\Delta}{\Gamma \ \ \textbf{==>}\ \ \langle \mathtt{if}\ (b)\ \textbf{\{}\mathtt{p}\textbf{\}}\ \mathtt{else}\ \textbf{\{}\mathtt{q}\textbf{\}};\rangle\phi,\Delta}$$

Dynamic Logic Calculus

$$\textbf{CONCATENATE}\ \frac{\Gamma \ \texttt{==>}\ \langle \texttt{p}\rangle\,(\langle \texttt{q}\rangle\phi)\,,\Delta}{\Gamma \ \texttt{==>}\ \langle \texttt{pq}\rangle\phi,\Delta}$$

$$\textbf{IF}\ \frac{\Gamma, b \doteq \texttt{TRUE}\ \texttt{==>}\ \langle \texttt{p}\rangle\phi,\Delta \qquad \Gamma, b \doteq \texttt{FALSE}\ \texttt{==>}\ \langle \texttt{q}\rangle\phi,\Delta}{\Gamma \ \texttt{==>}\ \langle \texttt{if}\ (b)\ \{\texttt{p}\}\ \texttt{else}\ \{\texttt{q}\};\rangle\phi,\Delta}$$

$$\textbf{ASSIGN}\ \frac{\{\texttt{x}/\texttt{x}_{old}\}\Gamma,\ \texttt{x} \doteq \{\texttt{x}/\texttt{x}_{old}\}t\ \texttt{==>}\ \phi,\ \{\texttt{x}/\texttt{x}_{old}\}\Delta}{\Gamma \ \texttt{==>}\ \langle x = t;\rangle\phi,\Delta}$$

$\texttt{x}_{old}$ **new program variable that “rescues” old value of** $\texttt{x}$

Dynamic Logic Calculus

$$\text{CONCATENATE}\ \frac{\Gamma \ \texttt{==>}\ \langle \texttt{p}\rangle\,(\langle \texttt{q}\rangle\phi)\,,\Delta}{\Gamma \ \texttt{==>}\ \langle \texttt{pq}\rangle\phi,\Delta}$$

$$\text{IF}\ \frac{\Gamma, b \doteq \texttt{TRUE}\ \texttt{==>}\ \langle \texttt{p}\rangle\phi,\Delta \qquad \Gamma, b \doteq \texttt{FALSE}\ \texttt{==>}\ \langle \texttt{q}\rangle\phi,\Delta}{\Gamma \ \texttt{==>}\ \langle \texttt{if}\ (b)\ \texttt{\{p\}}\ \texttt{else}\ \texttt{\{q\}};\rangle\phi,\Delta}$$

$$\text{ASSIGN}\ \frac{\{\texttt{x}/\texttt{x}_{old}\}\Gamma,\ \texttt{x} \doteq \{\texttt{x}/\texttt{x}_{old}\}t\ \texttt{==>}\ \phi,\ \{\texttt{x}/\texttt{x}_{old}\}\Delta}{\Gamma \ \texttt{==>}\ \langle x = t;\rangle\phi,\Delta}$$

$\texttt{x}_{old}$ new program variable that “rescues” old value of $\texttt{x}$

$$\text{UNWIND}\ \frac{\Gamma, b \doteq \texttt{FALSE}\ \texttt{==>}\ \phi,\Delta \qquad \Gamma, b \doteq \texttt{TRUE}\ \texttt{==>}\ \langle \texttt{p}\rangle\langle \texttt{while}\ (b)\ \texttt{\{p\}};\rangle\phi,\Delta}{\Gamma \ \texttt{==>}\ \langle \texttt{while}\ (b)\ \texttt{\{p\}};\rangle\phi,\Delta}$$

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \mathtt{p}\ \{\phi\}$$

If $\mathtt{p}$ is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ **->** $[\mathtt{p}]\,\phi$

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \mathtt{p}\ \{\phi\}$$

If $\mathtt{p}$ is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ **->** $[\mathtt{p}]\,\phi$

Valid formulas

$$[\mathtt{x = 1;}]\,(\mathtt{x} \doteq 1)$$

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \mathtt{p}\ \{\phi\}$$

If $\mathtt{p}$ is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ **->** $[\mathtt{p}]\,\phi$

Valid formulas

$[\mathtt{x} = 1;]\,(\mathtt{x} \doteq 1)$ $[\mathtt{while}\ (\mathbf{true})\ \mathbf{\{}\mathtt{x} = \mathtt{x};\mathbf{\}};]$ **false**

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \mathtt{p}\ \{\phi\}$$

If $\mathtt{p}$ is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ **->** $[\mathtt{p}]\,\phi$

Valid formulas

$[\mathtt{x} = 1;]\,(\mathtt{x} \doteq 1)$ $[\mathtt{while}\ (\mathbf{true})\ \{\mathtt{x} = \mathtt{x};\};]$ **false**

Validity depends on $\mathtt{p}$, $\mathtt{q}$

$\forall y.\,((\langle \mathtt{p}\rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{q}\rangle \mathtt{x} \doteq y))$ **meaning ?**

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \mathtt{p}\ \{\phi\}$$

If $\mathtt{p}$ is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ **->** $[\mathtt{p}]\,\phi$

Valid formulas

$[\mathtt{x} = \mathtt{1};]\,(\mathtt{x} \doteq 1)$ $[\mathtt{while}\ (\mathbf{true})\ \{\mathtt{x} = \mathtt{x};\};]$ **false**

Validity depends on $\mathtt{p}$, $\mathtt{q}$

$\forall y.\,((\langle \mathtt{p} \rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{q} \rangle \mathtt{x} \doteq y))$ $\mathtt{p}$, $\mathtt{q}$ **equivalent relative to** $\mathtt{x}$

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \mathtt{p}\ \{\phi\}$$

If $\mathtt{p}$ is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ **->** $[\mathtt{p}]\,\phi$

Valid formulas

$[\mathtt{x} = 1;]\,(\mathtt{x} \doteq 1)$ $[\mathtt{while}\ (\mathbf{true})\ \{\mathtt{x} = \mathtt{x};\};]$ **false**

Validity depends on $\mathtt{p}$, $\mathtt{q}$

$\forall y.\,((\langle\mathtt{p}\rangle\mathtt{x} \doteq y)$ **<->** $(\langle\mathtt{q}\rangle\mathtt{x} \doteq y))$ $\mathtt{p}$, $\mathtt{q}$ **equivalent relative to** $\mathtt{x}$

$\exists y.(\mathtt{x} \doteq y$ **->** $\langle\mathtt{p}\rangle\mathbf{true})$ **meaning ?**

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \mathtt{p}\ \{\phi\}$$

If $\mathtt{p}$ is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ **->** $[\mathtt{p}]\,\phi$

Valid formulas

$[\mathtt{x} = 1;]\,(\mathtt{x} \doteq 1)$ $[\mathtt{while}\ (\mathbf{true})\ \{\mathtt{x} = \mathtt{x};\};]$ **false**

Validity depends on $\mathtt{p}$, $\mathtt{q}$

$\forall y.\,((\langle \mathtt{p}\rangle \mathtt{x} \doteq y)$ **<->** $(\langle \mathtt{q}\rangle \mathtt{x} \doteq y))$ $\mathtt{p}$, $\mathtt{q}$ **equivalent relative to** $\mathtt{x}$

$\exists y.(\mathtt{x} \doteq y$ **->** $\langle \mathtt{p}\rangle \mathbf{true})$ $\mathtt{p}$ **terminates for some initial value of** $\mathtt{x}$

Induction Rule

Motivation

- UNWIND-rule only works if number of loop iterations small & known
- Properties of inductive FO data structures unprovable

  (numbers, lists, trees, etc.)

Induction Rule

Motivation

- UNWIND-rule only works if number of loop iterations small & known
- Properties of inductive FO data structures unprovable (numbers, lists, trees, etc.)

Induction Rule (over natural numbers)

$$\frac{\Gamma \ \texttt{==>}\ [n/0]\,\phi, \Delta \qquad \Gamma, [n/n']\,\phi \ \texttt{==>}\ [n/n'{+}1]\,\phi, \Delta \qquad \Gamma, \forall n.\phi \ \texttt{==>}\ \Delta}{\Gamma \ \texttt{==>}\ \Delta}$$

Where n logical variable, n' constant of type `int` not occurring in Γ, Δ

Induction Rule Example

Definition of $\mathbf{even}$ (unary predicate on `int`):

- **==>** $\mathbf{even}(0)$
- **==>** $\forall x.(\mathbf{even}(x)$ **->** $\mathbf{even}(x+2))$

How to prove ==> $\mathbf{even}(2*7)$?

Induction Rule Example

Definition of even (unary predicate on `int`):

- **==>** $\textbf{even}(0)$
- **==>** $\forall x.(\textbf{even}(x)$ **->** $\textbf{even}(x+2))$

How to prove ==> $\textbf{even}(2*7)$ **?**

1. **Apply definition** 7 **times**
2. **Use induction rule with induction hypothesis** $\phi \quad = \quad \textbf{even}(2*n)$

Induction Rule Example

Definition of $\mathbf{even}$ **(unary predicate on `int`):**

- **==>** $\mathbf{even}(0)$
- **==>** $\forall x.(\mathbf{even}(x)$ **->** $\mathbf{even}(x+2))$

How to prove ==> $\mathbf{even}(2*7)$ **?**

1. **Apply definition** 7 **times**
2. **Use induction rule with induction hypothesis** $\phi = \mathbf{even}(2*n)$

$$\frac{\texttt{==>}\,\mathbf{even}(2*0) \quad \mathbf{even}(2*n')\,\texttt{==>}\,\mathbf{even}(2*(n'+1)) \quad \forall n.\mathbf{even}(2*n)\,\texttt{==>}\,\mathbf{even}(2*7)}{\texttt{==>}\,\mathbf{even}(2*7)}$$

Demo in **`dlIntro/ind.key`**

Quantifying over Program Variables

What if induction hypothesis contains program?

Cannot quantify over program variables!

How to express validity for arbitrary initial value of program variable?

Quantifying over Program Variables

What if induction hypothesis contains program?

Cannot quantify over program variables!

How to express validity for arbitrary initial value of program variable?

Not allowed: $\forall \mathtt{i}.\langle \mathtt{p}(\mathtt{i})\rangle \phi$ **(program $\neq$ logical variable)**

Quantifying over Program Variables

What if induction hypothesis contains program?

Cannot quantify over program variables!

How to express validity for arbitrary initial value of program variable?

Not allowed: $\forall \mathtt{i}.\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(program $\neq$ logical variable)**

Not intended: **==>** $\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(Validity of sequents: quantification over *all* states)**

Quantifying over Program Variables

What if induction hypothesis contains program?

Cannot quantify over program variables!

How to express validity for arbitrary initial value of program variable?

Not allowed: $\forall \mathtt{i}.\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(program $\neq$ logical variable)**

Not intended: **==>** $\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(Validity of sequents: quantification over *all* states)**

As previous: $\forall n.(n \doteq \mathtt{i}$ **->** $\langle \mathtt{p}(\mathtt{i})\rangle\phi)$

Quantifying over Program Variables

What if induction hypothesis contains program?

Cannot quantify over program variables!

How to express validity for arbitrary initial value of program variable?

Not allowed: $\forall \mathtt{i}.\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(program $\neq$ logical variable)**

Not intended: **==>** $\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(Validity of sequents: quantification over *all* states)**

As previous: $\forall n.(n \doteq \mathtt{i}$ **->** $\langle \mathtt{p}(\mathtt{i})\rangle\phi)$

Not allowed: $\forall n.\langle \mathtt{p}(n)\rangle\phi$ **(no logical variables in programs)**

Quantifying over Program Variables

What if induction hypothesis contains program?

Cannot quantify over program variables!

How to express validity for arbitrary initial value of program variable?

Not allowed: $\forall \mathtt{i}.\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(program $\neq$ logical variable)**

Not intended: **==>** $\langle \mathtt{p}(\mathtt{i})\rangle\phi$ **(Validity of sequents: quantification over *all* states)**

As previous: $\forall n.(n \doteq \mathtt{i}$ **->** $\langle \mathtt{p}(\mathtt{i})\rangle\phi)$

Not allowed: $\forall n.\langle \mathtt{p}(n)\rangle\phi$ **(no logical variables in programs)**

Solution

Use explicit construct to record state change information

Update $\forall n.(\mathbf{\{}\mathtt{i} := n\mathbf{\}}\langle \mathtt{p}(\mathtt{i})\rangle\phi)$

Explicit State Updates

Updates record computation state in which we evaluate a formula

Explicit State Updates

Updates record computation state in which we evaluate a formula

Syntax

If $\mathtt{v}$ is program variable, t, t' FOL terms, and ϕ any DL formula, then $\{\mathtt{v} := t\}\phi$ is DL formula and $\{\mathtt{v} := t\}t'$ is DL term

Explicit State Updates

Updates record computation state in which we evaluate a formula

Syntax

If $\mathtt{v}$ is program variable, t, t' FOL terms, and ϕ any DL formula, then $\{\mathtt{v} := t\}\phi$ is DL formula and $\{\mathtt{v} := t\}t'$ is DL term

Semantics

$$\mathcal{I}, \beta \models \{\mathtt{v} := t\}\phi \quad \textbf{iff} \quad \mathcal{I}_{\mathtt{v}}^{val_{\mathcal{I},\beta}(t)}, \beta \models \phi$$

Semantics identical to assignment, may depend on logical variables in t

Updates work like “lazy” assignments

Updates are not assignments: may contain logical variable

Updates are not equations: change interpretation of non-rigid terms

Computing Effect of Updates (Automatic)

Update followed by program variable | **by logical variable**

$$\{\mathtt{x} := t\}\mathtt{y} \quad \rightsquigarrow \quad \mathtt{y}$$

$$\{\mathtt{x} := t\}\mathtt{x} \quad \rightsquigarrow \quad t$$

$$\{\mathtt{x} := t\}w \quad \rightsquigarrow \quad w$$

Computing Effect of Updates (Automatic)

Update followed by program variable **by logical variable**

$$\{\mathtt{x} := t\}\mathtt{y} \quad \leadsto \quad \mathtt{y}$$

$$\{\mathtt{x} := t\}\mathtt{x} \quad \leadsto \quad t$$

$$\{\mathtt{x} := t\}w \quad \leadsto \quad w$$

Update followed by complex term

$$\{\mathtt{x} := t\}f(t_1, \ldots, t_n) \quad \leadsto \quad f(\{\mathtt{x} := t\}t_1, \ldots, \{\mathtt{x} := t\}t_n)$$

Computing Effect of Updates (Automatic)

Update followed by program variable | **by logical variable**

$$\{\mathtt{x} := t\}\mathtt{y} \quad \rightsquigarrow \quad \mathtt{y}$$

$$\{\mathtt{x} := t\}\mathtt{x} \quad \rightsquigarrow \quad t$$

$$\{\mathtt{x} := t\}w \quad \rightsquigarrow \quad w$$

Update followed by complex term

$$\{\mathtt{x} := t\}f(t_1, \ldots, t_n) \quad \rightsquigarrow \quad f(\{\mathtt{x} := t\}t_1, \ldots, \{\mathtt{x} := t\}t_n)$$

Update followed by first-order formula

$$\{\mathtt{x} := t\}(\phi \,\&\, \psi) \quad \rightsquigarrow \quad \{\mathtt{x} := t\}\phi \,\&\, \{\mathtt{x} := t\}\psi \quad \textbf{etc.}$$

$$\{\mathtt{x} := t\}(\forall y.\phi) \quad \rightsquigarrow \quad \forall y.(\{\mathtt{x} := t\}\phi) \quad \textbf{etc.}$$

Computing Effect of Updates (Automatic)

Update followed by program variable | **by logical variable**

$$\{\mathtt{x} := t\}\mathtt{y} \quad \rightsquigarrow \quad \mathtt{y}$$

$$\{\mathtt{x} := t\}\mathtt{x} \quad \rightsquigarrow \quad t$$

$$\{\mathtt{x} := t\}w \quad \rightsquigarrow \quad w$$

Update followed by complex term

$$\{\mathtt{x} := t\}f(t_1, \ldots, t_n) \quad \rightsquigarrow \quad f(\{\mathtt{x} := t\}t_1, \ldots, \{\mathtt{x} := t\}t_n)$$

Update followed by first-order formula

$$\{\mathtt{x} := t\}(\phi \,\&\, \psi) \quad \rightsquigarrow \quad \{\mathtt{x} := t\}\phi \,\&\, \{\mathtt{x} := t\}\psi \quad \textbf{etc.}$$

$$\{\mathtt{x} := t\}(\forall y.\phi) \quad \rightsquigarrow \quad \forall y.(\{\mathtt{x} := t\}\phi) \quad \textbf{etc.}$$

Update followed by program formula

$$\{\mathtt{x} := t\}(\langle \mathtt{p} \rangle \phi) \quad \rightsquigarrow \quad \{\mathtt{x} := t\}(\langle \mathtt{p} \rangle \phi)$$

unchanged!

Update computation delayed until $\mathtt{p}$ symbolically executed

Assignment Rule Using Updates

$$\textsf{ASSIGN}\ \frac{\Gamma \ \texttt{==>} \ \{\mathtt{x} := t\}\phi, \Delta}{\Gamma \ \texttt{==>} \ \langle \mathtt{x} = \mathtt{t};\rangle\phi, \Delta}$$

Avoids renaming of program variables

Works as long as t has no side effects (ok in simple DL)

But: rules dealing with programs need to account for updates

Assignment Rule Using Updates

$$\text{ASSIGN}\ \frac{\Gamma\ \texttt{==>}\ \{\mathtt{x} := t\}\phi, \Delta}{\Gamma\ \texttt{==>}\ \langle \mathtt{x} = \mathtt{t;}\rangle\phi, \Delta}$$

Avoids renaming of program variables

Works as long as t has no side effects (ok in simple DL)

But: rules dealing with programs need to account for updates

Solution: rules work on first active statement after updates and prefix, followed by postfix (remaining code)

Explicit concatenation rule not longer useful

Assignment Rule Using Updates

$$\text{ASSIGN}\ \frac{\Gamma \ \texttt{==>}\ \{\mathtt{x} := t\}\phi, \Delta}{\Gamma \ \texttt{==>}\ \langle \mathtt{x} = \mathtt{t};\rangle \phi, \Delta}$$

Avoids renaming of program variables

Works as long as t has no side effects (ok in simple DL)

But: rules dealing with programs need to account for updates

Solution: rules work on first active statement after updates and prefix, followed by postfix (remaining code)

Explicit concatenation rule not longer useful

General form of conclusion in rule for symbolic execution

$$\Gamma \ \texttt{==>}\ \mathcal{U}\langle\ \pi\ \ \xi;\ \ \omega\ \rangle \phi, \Delta$$

Prefix Active statement Postfix

Example Proof

```
\programVariables { // program variables in FSym
  int x;
}
\problem {
 \exists int y; (x = y ->  // y logical variable
    \<{while (x > 0) {x = x-1;}}\> true)
      // modal brackets written as \<, \>
}
```

Intuitive Meaning? Satisfiable? Valid?

Demo

`dlIntro/term.key`