Introduction to Dynamic Logics

Prof. P. H. Schmitt, Christian Engel, Benjamin Weiß

December 3, 2008

Reasoning by Syntactic Transformation

Establish $\Gamma \models G$ by purely syntactic transformations of Γ and G

(Logic) Calculus: a set of transformation rules $\mathcal{R}$ defining relation $\vdash \subseteq 2^{For_0^\Sigma} \times For_0^\Sigma$ such that $\Gamma \models G$ iff $\Gamma \vdash G$

$\models \subseteq \vdash$ **Completeness** $\models \supseteq \vdash$ **Soundness**

Sequent Calculus based on notion of **sequent**

$$\underbrace{\psi_1, \ldots, \psi_m}_{Antecedent} \quad \texttt{==>} \quad \underbrace{\phi_1, \ldots, \phi_n}_{Succedent}$$

has same semantics as

$$\begin{array}{ccc} (\psi_1 \& \cdots \& \psi_m) & \texttt{->} & (\phi_1 \mid \cdots \mid \phi_n) \\ \{\psi_1, \ldots, \psi_m\} & \models & \phi_1 \mid \cdots \mid \phi_n \end{array}$$

Notation for Sequents

$$\psi_1, \ldots, \psi_m \quad \texttt{==>} \quad \phi_1, \ldots, \phi_n$$

Consider antecedent/succedent as sets of formulas, may be empty

Use schematic variables Γ, Δ that match sets of formulas

$$\Gamma \quad \texttt{==>} \quad \Delta, \phi$$

Matches any sequent with an occurrence of ϕ in succedent
Call ϕ **main formula** and Γ **side formulas** of sequent

Any sequent of the form Γ, ϕ ==> Δ, ϕ is valid: **axiom**

Sequent Calculus Rules

Basic idea write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

$$\text{RULE NAME}\ \frac{\overbrace{\Gamma_1 \ {==>}\ \Delta_1 \quad \cdots \quad \Gamma_r \ {==>}\ \Delta_r}^{\text{Premisses}}}{\underbrace{\Gamma \ {==>}\ \Delta}_{\text{Conclusion}}}$$

Sequent Calculus Rules

Basic idea write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

$$\text{RULE NAME}\ \frac{\overbrace{\Gamma_1 \texttt{==>} \Delta_1 \quad \cdots \quad \Gamma_r \texttt{==>} \Delta_r}^{\text{Premisses}}}{\underbrace{\Gamma \texttt{==>} \Delta}_{\text{Conclusion}}}$$

Sound rule (essential) $\{\Gamma_1\texttt{==>}\Delta_1, \cdots, \Gamma_r\texttt{==>}\Delta_r\} \models (\Gamma\texttt{==>}\Delta)$

Sequent Calculus Rules

Basic idea write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

$$\text{RULE NAME}\ \frac{\overbrace{\Gamma_1 ==> \Delta_1 \quad \cdots \quad \Gamma_r ==> \Delta_r}^{\text{Premisses}}}{\underbrace{\Gamma ==> \Delta}_{\text{Conclusion}}}$$

Sound rule (essential) $\{\Gamma_1 ==> \Delta_1, \cdots, \Gamma_r ==> \Delta_r\} \models (\Gamma ==> \Delta)$

Complete rule (desirable) $\{\Gamma ==> \Delta\} \models (\Gamma_1 ==> \Delta_1 \& \cdots \& \Gamma_r ==> \Delta_r)$

Sequent Calculus Rules

Basic idea write syntactic transformation pattern for sequents that mimicks semantics of connectives as closely as possible

$$\text{RULE NAME}\ \frac{\overbrace{\Gamma_1 \texttt{==>} \Delta_1 \quad \cdots \quad \Gamma_r \texttt{==>} \Delta_r}^{\text{Premisses}}}{\underbrace{\Gamma \texttt{==>} \Delta}_{\text{Conclusion}}}$$

Sound rule (essential) $\{\Gamma_1\texttt{==>}\Delta_1, \cdots, \Gamma_r\texttt{==>}\Delta_r\} \models (\Gamma\texttt{==>}\Delta)$

Complete rule (desirable) $\{\Gamma\texttt{==>}\Delta\} \models (\Gamma_1\texttt{==>}\Delta_1 \& \cdots \& \Gamma_r\texttt{==>}\Delta_r)$

Admissible to have no premisses (iff conclusion is valid, eg axiom)

Sequent Calculus Proofs

Goal to prove: $\mathcal{S} = \quad \psi_1, \ldots, \psi_m$ **==>** $\phi_1, \ldots, \phi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{S}$

Sequent Calculus Proofs

Goal to prove: $\mathcal{S} = \quad \psi_1, \ldots, \psi_m$ **==>** $\phi_1, \ldots, \phi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{S}$
- instantiate $\mathcal{R}$ such that conclusion identical to $\mathcal{S}$

Sequent Calculus Proofs

Goal to prove: $\mathcal{S} = \quad \psi_1, \ldots, \psi_m$ **==>** $\phi_1, \ldots, \phi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{S}$
- instantiate $\mathcal{R}$ such that conclusion identical to $\mathcal{S}$
- recursively find proofs for resulting premisses $\mathcal{S}_1, \ldots, \mathcal{S}_r$

Sequent Calculus Proofs

Goal to prove: $\mathcal{S} = \quad \psi_1, \ldots, \psi_m$ **==>** $\phi_1, \ldots, \phi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{S}$
- instantiate $\mathcal{R}$ such that conclusion identical to $\mathcal{S}$
- recursively find proofs for resulting premisses $\mathcal{S}_1, \ldots, \mathcal{S}_r$
- tree structure with goal as root

Sequent Calculus Proofs

Goal to prove: $\mathcal{S} = \quad \psi_1, \ldots, \psi_m$ ==> $\phi_1, \ldots, \phi_n$

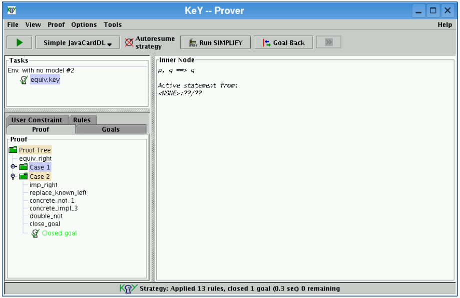
- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{S}$
- instantiate $\mathcal{R}$ such that conclusion identical to $\mathcal{S}$
- recursively find proofs for resulting premisses $\mathcal{S}_1$, ..., $\mathcal{S}_r$
- tree structure with goal as root
- **close** proof branch when rule without premise encountered

Sequent Calculus Proofs

Goal to prove: $\mathcal{S} = \quad \psi_1, \ldots, \psi_m$ **==>** $\phi_1, \ldots, \phi_n$

- find rule $\mathcal{R}$ whose conclusion matches $\mathcal{S}$
- instantiate $\mathcal{R}$ such that conclusion identical to $\mathcal{S}$
- recursively find proofs for resulting premisses $\mathcal{S}_1, \ldots, \mathcal{S}_r$
- tree structure with goal as root
- **close** proof branch when rule without premise encountered

Goal-directed proof search
In KeY tool proof displayed as JAVA Swing tree



Rules of Propositional Sequent Calculus

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \texttt{ ==> } A, \Delta}{\Gamma, \texttt{!}A \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } \Delta}{\Gamma \texttt{ ==> } \texttt{!}A, \Delta}$

Rules of Propositional Sequent Calculus

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \texttt{ ==> } A, \Delta}{\Gamma, \texttt{!}A \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } \Delta}{\Gamma \texttt{ ==> } \texttt{!}A, \Delta}$
and	$\dfrac{\Gamma, A, B \texttt{ ==> } \Delta}{\Gamma, A \& B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } A, \Delta \quad \Gamma \texttt{ ==> } B, \Delta}{\Gamma \texttt{ ==> } A \& B, \Delta}$

Rules of Propositional Sequent Calculus

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \;\texttt{==>}\; A, \Delta}{\Gamma, \texttt{!}A \;\texttt{==>}\; \Delta}$	$\dfrac{\Gamma, A \;\texttt{==>}\; \Delta}{\Gamma \;\texttt{==>}\; \texttt{!}A, \Delta}$
and	$\dfrac{\Gamma, A, B \;\texttt{==>}\; \Delta}{\Gamma, A \texttt{\&} B \;\texttt{==>}\; \Delta}$	$\dfrac{\Gamma \;\texttt{==>}\; A, \Delta \quad \Gamma \;\texttt{==>}\; B, \Delta}{\Gamma \;\texttt{==>}\; A \texttt{\&} B, \Delta}$
or	$\dfrac{\Gamma, A \;\texttt{==>}\; \Delta \quad \Gamma, B \;\texttt{==>}\; \Delta}{\Gamma, A \texttt{\|} B \;\texttt{==>}\; \Delta}$	$\dfrac{\Gamma \;\texttt{==>}\; A, B, \Delta}{\Gamma \;\texttt{==>}\; A \texttt{\|} B, \Delta}$

Rules of Propositional Sequent Calculus

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \texttt{ ==> } A, \Delta}{\Gamma, \texttt{!}A \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } \Delta}{\Gamma \texttt{ ==> } \texttt{!}A, \Delta}$
and	$\dfrac{\Gamma, A, B \texttt{ ==> } \Delta}{\Gamma, A\&B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } A, \Delta \quad \Gamma \texttt{ ==> } B, \Delta}{\Gamma \texttt{ ==> } A\&B, \Delta}$
or	$\dfrac{\Gamma, A \texttt{ ==> } \Delta \quad \Gamma, B \texttt{ ==> } \Delta}{\Gamma, A \mid B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } A, B, \Delta}{\Gamma \texttt{ ==> } A \mid B, \Delta}$
imp	$\dfrac{\Gamma \texttt{ ==> } A, \Delta \quad \Gamma, B \texttt{ ==> } \Delta}{\Gamma, A\texttt{->}B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } B, \Delta}{\Gamma \texttt{ ==> } A\texttt{->}B, \Delta}$

Rules of Propositional Sequent Calculus

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \texttt{ ==> } A, \Delta}{\Gamma, \texttt{!}A \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } \Delta}{\Gamma \texttt{ ==> } \texttt{!}A, \Delta}$
and	$\dfrac{\Gamma, A, B \texttt{ ==> } \Delta}{\Gamma, A\&B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } A, \Delta \quad \Gamma \texttt{ ==> } B, \Delta}{\Gamma \texttt{ ==> } A\&B, \Delta}$
or	$\dfrac{\Gamma, A \texttt{ ==> } \Delta \quad \Gamma, B \texttt{ ==> } \Delta}{\Gamma, A\,\vert\,B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } A, B, \Delta}{\Gamma \texttt{ ==> } A\,\vert\,B, \Delta}$
imp	$\dfrac{\Gamma \texttt{ ==> } A, \Delta \quad \Gamma, B \texttt{ ==> } \Delta}{\Gamma, A\texttt{->}B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } B, \Delta}{\Gamma \texttt{ ==> } A\texttt{->}B, \Delta}$

$$\text{AXIOM} \quad \overline{\Gamma, A \texttt{ ==> } A, \Delta} \qquad \text{TRUE} \quad \overline{\Gamma \texttt{ ==> } \texttt{true}, \Delta}$$

$$\text{FALSE} \quad \overline{\Gamma, \texttt{false} \texttt{ ==> } \Delta}$$

Justification of Rules

Compute rules by applying semantics definition of connectives

Justification of Rules

Compute rules by applying semantics definition of connectives

$$\text{OR\_RIGHT}\ \frac{\Gamma \texttt{ ==> } A, B, \Delta}{\Gamma \texttt{ ==> } A \mid B, \Delta}$$

$$\text{AND\_RIGHT}\ \frac{\Gamma \texttt{ ==> } A, \Delta \quad \Gamma \texttt{ ==> } B, \Delta}{\Gamma \texttt{ ==> } A \& B, \Delta}$$

Justification of Rules

Compute rules by applying semantics definition of connectives

$$\text{OR\_RIGHT}\ \frac{\Gamma\ \texttt{==>}\ A, B, \Delta}{\Gamma\ \texttt{==>}\ A \,|\, B, \Delta}$$

$$\text{AND\_RIGHT}\ \frac{\Gamma\ \texttt{==>}\ A, \Delta \quad \Gamma\ \texttt{==>}\ B, \Delta}{\Gamma\ \texttt{==>}\ A \& B, \Delta}$$

Follows directly from semantics of sequents

$$\Gamma\texttt{->}(A \& B) \,|\, \Delta$$

iff

$$\Gamma\texttt{->}A \,|\, \Delta \quad \textbf{and} \quad \Gamma\texttt{->}B \,|\, \Delta$$

A Simple Proof

$$\Gamma \texttt{ ==> } (A \& (A \texttt{->} B)) \texttt{->} B, \Delta$$

A Simple Proof

$$\frac{\Gamma, (A \& (A \text{->} B)) \text{ ==> } B, \Delta}{\Gamma \text{ ==> } (A \& (A \text{->} B)) \text{->} B, \Delta}$$

A Simple Proof

$$\cfrac{\cfrac{\Gamma, A, (A\texttt{->}B) \texttt{ ==> } B, \Delta}{\Gamma, (A\&(A\texttt{->}B)) \texttt{ ==> } B, \Delta}}{\Gamma \texttt{ ==> } (A\&(A\texttt{->}B))\texttt{->}B, \Delta}$$

A Simple Proof

$$\dfrac{\dfrac{\dfrac{\Gamma, A \texttt{ ==> } B, A, \Delta \qquad \Gamma, A, B \texttt{ ==> } B, \Delta}{\Gamma, A, (A\texttt{->}B) \texttt{ ==> } B, \Delta}}{\Gamma, (A\&(A\texttt{->}B)) \texttt{ ==> } B, \Delta}}{\Gamma \texttt{ ==> } (A\&(A\texttt{->}B))\texttt{->}B, \Delta}$$

A Simple Proof

$$
\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma, A \texttt{ ==> } B, A, \Delta} \qquad \dfrac{*}{\Gamma, A, B \texttt{ ==> } B, \Delta}}{\Gamma, A, (A\texttt{->}B) \texttt{ ==> } B, \Delta}}{\Gamma, (A\&(A\texttt{->}B)) \texttt{ ==> } B, \Delta}}{\Gamma \texttt{ ==> } (A\&(A\texttt{->}B))\texttt{->}B, \Delta}
$$

A Simple Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma, A \texttt{ ==> } B, A, \Delta} \qquad \dfrac{*}{\Gamma, A, B \texttt{ ==> } B, \Delta}}{\Gamma, A, (A\texttt{->}B) \texttt{ ==> } B, \Delta}}{\Gamma, (A\texttt{\&}(A\texttt{->}B)) \texttt{ ==> } B, \Delta}}{\Gamma \texttt{ ==> } (A\texttt{\&}(A\texttt{->}B))\texttt{->}B, \Delta}$$

A proof is **closed**, if all its branches are closed.

Sequent Calculus for FOL

- $\{t/t'\}\phi$ is result of replacing each occurrence of t in ϕ with t'
- $t^{z'}$ any variable free term of type $z' \prec z$
- c^z **new** constant of type z (occurs not in current proof branch)
- Equations can be reversed by commutativity

Sequent Calculus for FOL

	left side, antecedent	right side, succedent
all	$\dfrac{\Gamma, \texttt{\textbackslash forall}\ z\ x; \phi, \{x/t^{z'}\}\phi\ \texttt{==>}\ \Delta}{\Gamma, \texttt{\textbackslash forall}\ z\ x; \phi\ \texttt{==>}\ \Delta}$	$\dfrac{\Gamma\ \texttt{==>}\ \{x/c^{z}\}\phi, \Delta}{\Gamma\ \texttt{==>}\ \texttt{\textbackslash forall}\ z\ x; \phi, \Delta}$

- $\{t/t'\}\phi$ is result of replacing each occurrence of t in ϕ with t'
- $t^{z'}$ any variable free term of type $z' \prec z$
- c^z **new** constant of type z (occurs not in current proof branch)
- Equations can be reversed by commutativity

Sequent Calculus for FOL

	left side, antecedent	right side, succedent
all	$\dfrac{\Gamma, \texttt{\textbackslash forall}\ z\ x; \phi, \{x/t^{z'}\}\phi \ \texttt{==>}\ \Delta}{\Gamma, \texttt{\textbackslash forall}\ z\ x; \phi \ \texttt{==>}\ \Delta}$	$\dfrac{\Gamma \ \texttt{==>}\ \{x/c^{z}\}\phi, \Delta}{\Gamma \ \texttt{==>}\ \texttt{\textbackslash forall}\ z\ x; \phi, \Delta}$
ex	$\dfrac{\Gamma, \{x/c^{z}\}\phi \ \texttt{==>}\ \Delta}{\Gamma, \texttt{\textbackslash exists}\ z\ x; \phi \ \texttt{==>}\ \Delta}$	$\dfrac{\Gamma \ \texttt{==>}\ \{x/t^{z'}\}\phi, \texttt{\textbackslash exists}\ z}{\Gamma \ \texttt{==>}\ \texttt{\textbackslash exists}\ z\ x; \phi,}$

- $\{t/t'\}\phi$ is result of replacing each occurrence of t in ϕ with t'
- $t^{z'}$ any variable free term of type $z' \prec z$
- c^{z} **new** constant of type z (occurs not in current proof branch)
- Equations can be reversed by commutativity

Sequent Calculus for FOL

	left side, antecedent	right side, succedent
all	$\dfrac{\Gamma, \texttt{\textbackslash forall}\ z\ x; \phi, \{x/t^{z'}\}\phi \texttt{ ==> } \Delta}{\Gamma, \texttt{\textbackslash forall}\ z\ x; \phi \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } \{x/c^z\}\phi, \Delta}{\Gamma \texttt{ ==> } \texttt{\textbackslash forall}\ z\ x; \phi, \Delta}$
ex	$\dfrac{\Gamma, \{x/c^z\}\phi \texttt{ ==> } \Delta}{\Gamma, \texttt{\textbackslash exists}\ z\ x; \phi \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } \{x/t^{z'}\}\phi, \texttt{\textbackslash exists}\ z}{\Gamma \texttt{ ==> } \texttt{\textbackslash exists}\ z\ x; \phi,}$
eq	$\dfrac{\Gamma, t_1 \doteq t_2, \{t_1/t_2\}\psi \texttt{ ==> } \{t_1/t_2\}\phi, \Delta}{\Gamma, t_1 \doteq t_2, \psi \texttt{ ==> } \phi, \Delta}$	$\dfrac{}{\Gamma \texttt{ ==> } t \doteq t, \Delta}$

- $\{t/t'\}\phi$ is result of replacing each occurrence of t in ϕ with t'
- $t^{z'}$ any variable free term of type $z' \prec z$
- c^z **new** constant of type z (occurs not in current proof branch)
- Equations can be reversed by commutativity

Some Predefined Symbols in KeY Logic

Types

`int`, `boolean`, classes of the Java context of the proof obligation

Predicates on `int`

`>`, `<`, `>=`, `<=`

Functions and Constants

'`+`', '`-`', '`/`', '`%`', '`0`', '`1`', ...

'`TRUE`', '`FALSE`'

Taclets - The rule description language of KeY

$\backslash$assumes (seq) $\backslash$find ($\vdash_{opt}$ Φ) $\backslash$replacewith($\vdash_{opt}$ Φ')
$\backslash$add($\vdash$ seq')...; ...; ...
$\backslash$heuristics($name^{+}$)

Taclets - The rule description language of KeY

$\backslash$assumes (*seq*) $\backslash$find ($\vdash_{opt}\ \Phi$) $\backslash$replacewith($\vdash_{opt}\ \Phi'$)
$\backslash$add($\vdash$ *seq*$'$)...; ...; ...
$\backslash$heuristics(*name*$^+$)

SYNTAX

`\find`	sequent (max. one formula), formula or term
`\assumes`	additional condition
`\replacewith`	replaces the \find part ($\vdash_{opt}$ depends on \find)
`\add`	adds the sequent to the antecedent or succedent
;	start new subgoal
`\heuristics`	adds the taclet to the enumerated heuristics

The `and-right` rule as taclet

TEXTBOOK

$$\frac{\Gamma \vdash A,\ \Delta \qquad\qquad \Gamma \vdash B,\ \Delta}{\Gamma \vdash A \wedge B,\ \Delta}\quad (and - right)$$

The and-right rule as taclet

TEXTBOOK

$$\frac{\Gamma \vdash A,\ \Delta \qquad \Gamma \vdash B,\ \Delta}{\Gamma \vdash A \wedge B,\ \Delta}\ (and - right)$$

TACLET

```
\find( ⊢ A ∧ B )
   \replacewith ( ⊢ A );
   \replacewith ( ⊢ B )
\heuristics(simplify)
```

First-Order Formula in KeY Syntax

```
\sorts { // types are called 'sorts'
   person; // one declaration per line, end with ';'
}
\functions { // ResultType FctSymbol(ParType,..,ParType)
   int age(person); // 'int' predefined type
}
\predicates { // PredSymbol(ParType,..,ParType)
   parent(person,person);
}
\problem { // Goal formula, // '>=' predef.
 \forall person son; \forall person father; (
     parent(father,son) -> age(father) >= age(son))
}
```

Another Example

Types $\mathcal{T} = \{z\}$
Predicates $\mathcal{P} = \{p\},\ \sigma(p) = \langle z, z\rangle$
Functions $\mathcal{F} = \{\}$

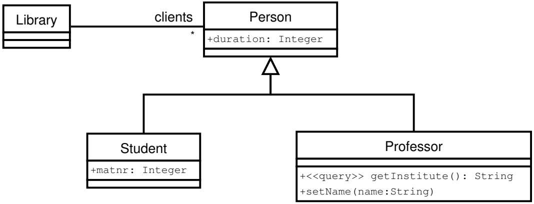
$$(\texttt{\textbackslash exists}\ z\ x; \texttt{\textbackslash exists}\ z\ y; p(x,y)\ \texttt{\&}\ \texttt{\textbackslash forall}\ z\ x; \texttt{!}p(x,x))\ \texttt{->}$$
$$\texttt{\textbackslash exists}\ z\ x; \texttt{\textbackslash exists}\ z\ y; (\texttt{!}x \doteq y)$$

Intuitive Meaning? Satisfiable? Valid?

Demo

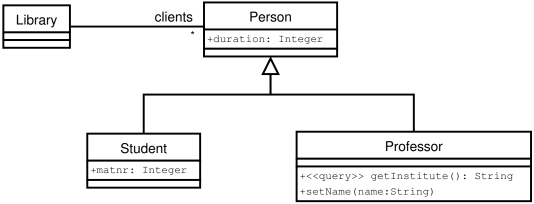
`demo1.key`

Example: JML to FOL



Types?

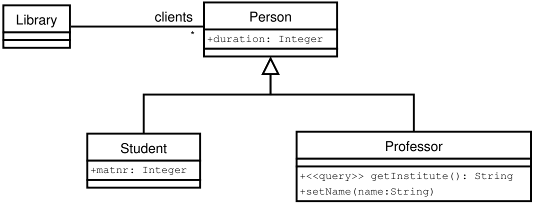
Example: JML to FOL



Types? Library, Person, Student, Professor (+ some predefined)

Functions?

Example: JML to FOL



Types? Library, Person, Student, Professor (+ some predefined)

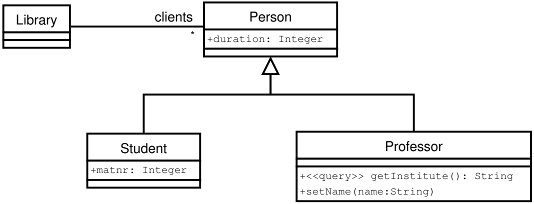
Functions?

Attributes `int Person.duration`, `int Student.matnr`

Queries `String Professor.getInstitute`

incl. some predefined

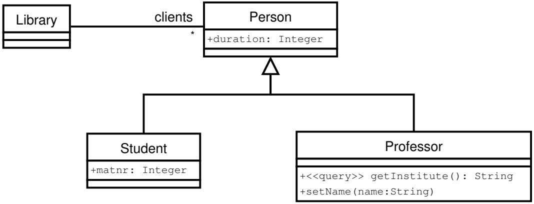
Example: JML to FOL



Meaning?

```
public class Student{
 /*@ public invariant (\forall Student s;
                           s.matnr==matnr; s==this);@*/ ..
}
```

Example: JML to FOL

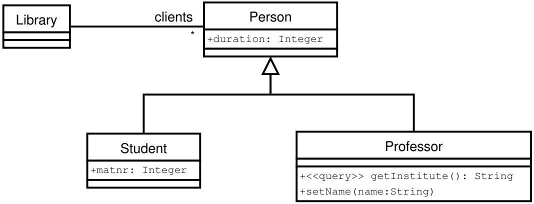


A student is uniquely identified by his/her student id (matnr)

```
public class Student{
 /*@ public invariant (\forall Student s;
                         s.matnr==matnr; s==this);@*/ ..
}
```

in FOL?

Example: JML to FOL



A student is uniquely identified by his/her student id (matnr)

```
public class Student{
 /*@ public invariant (\forall Student s;
                         s.matnr==matnr; s==this);@*/ ..
}
```

in FOL:

\forall Student p1;\forall Student p2;
    (p1.matnr = p2.matnr -> p1 = p2)

Do we really need another kind of logics?

"There is a tradition in logic, carried over into computer science, to think of pure first order logic as a universal language. In fact first order language is about as useful in verification as a Turing machine is in software engineering:

CUTE TO WATCH BUT NOT VERY USEFUL."

V. Pratt

State Dependency of Formula Evaluation

(Closed) FOL formula is either true or false wrt **interpretation** $\mathcal{D}$
Consider $\mathcal{D} = (U, I)$ 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 true or false wrt **interpretation** $\mathcal{D}$
Consider $\mathcal{D} = (U, I)$ 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 $\mathcal{D}$ such that $\mathcal{D} \models \mathtt{x} \doteq 3$
Executing $\mathtt{x} = 4$ results in $\mathcal{D}$ such that $\mathcal{D} \not\models \mathtt{x} \doteq 3$

State Dependency of Formula Evaluation

(Closed) FOL formula is either true or false wrt **interpretation** $\mathcal{D}$
Consider $\mathcal{D} = (U, I)$ 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 $\mathcal{D}$ such that $\mathcal{D} \models \mathtt{x} \doteq 3$
Executing $\mathtt{x} = 4$ results in $\mathcal{D}$ such that $\mathcal{D} \not\models \mathtt{x} \doteq 3$

Need a logic to capture state before/after program execution

Dynamic Logic (Simple Version) Signature

Definition (Signature)

$$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \mathcal{PV}, \alpha, \sigma, \mathbf{\Pi_0}, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq, \langle\cdot\rangle\cdot, [\cdot]\cdot\})$$

Type Symbols	$\mathcal{T} = \{\texttt{int}, \texttt{boolean}\}$
Logical Variables	$\mathcal{V} = \{y_i \mid i \in \mathbb{N}\}$
Predicate Symbols	$\mathcal{P} = \{>, >=, <, <=\}$
Function Symbols	$\mathcal{F} = \{+, -, *, 0, 1, \ldots\}$
Program Variables	$\mathcal{PV} = \{\texttt{x}_i \mid i \in \mathbb{N}\}$

Signature of functions/predicates as usual

Dynamic Logic (Simple Version) Signature

Definition (Signature)

$$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \mathcal{PV}, \alpha, \sigma, \mathbf{\Pi_0}, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq, \langle\cdot\rangle\cdot, [\cdot]\cdot\})$$

Type Symbols	$\mathcal{T} = \{\mathtt{int}, \mathtt{boolean}\}$
Logical Variables	$\mathcal{V} = \{y_i \mid i \in \boldsymbol{N}\}$
Predicate Symbols	$\mathcal{P} = \{>, >=, <, <=\}$
Function Symbols	$\mathcal{F} = \{+, -, *, 0, 1, \ldots\}$
Program Variables	$\mathcal{PV} = \{\mathtt{x}_i \mid i \in \boldsymbol{N}\}$

Signature of functions/predicates as usual

Atomic Programs Π_0:

Assignments $\mathtt{x} = t$ with $x \in \mathcal{PV}$, t term of type `int` w/o logical variables

Dynamic Logic (Simple Version) Signature

Definition (Signature)

$$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \mathcal{PV}, \alpha, \sigma, \mathbf{\Pi_0}, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq, \langle\cdot\rangle\cdot, [\cdot]\cdot\})$$

Type Symbols $\mathcal{T} = \{\mathtt{int}, \mathtt{boolean}\}$
Logical Variables $\mathcal{V} = \{y_i \mid i \in \boldsymbol{N}\}$
Predicate Symbols $\mathcal{P} = \{>, >=, <, <=\}$
Function Symbols $\mathcal{F} = \{+, -, *, 0, 1, \ldots\}$
Program Variables $\mathcal{PV} = \{\mathtt{x}_i \mid i \in \boldsymbol{N}\}$
Signature of functions/predicates as usual
Atomic Programs Π_0:
Assignments $\mathtt{x} = t$ with $x \in \mathcal{PV}$, t term of type $\mathtt{int}$ w/o logical variables
Modal Connectives $\langle\cdot\rangle\cdot$ “diamond”, $[\cdot]\cdot$ “box”
First argument program, second argument formula

Dynamic Logic (Simple Version) Programs

Programs Π

- If π is an atomic program, then $\pi;$ is a program

Dynamic Logic (Simple Version) Programs

Programs Π

- If π is an atomic program, then π; is a program
- If α and γ are programs, then $\alpha\gamma$ is a program

Dynamic Logic (Simple Version) Programs

Programs Π

- If π is an atomic program, then π; is a program
- If α and γ are programs, then $\alpha\gamma$ is a program
- If b is a variable-free term of type `boolean`, α and γ programs, then

  $$\textbf{if}\ (\mathrm{b})\ \{\alpha\}\ \textbf{else}\ \{\gamma\};$$

  is a program

Dynamic Logic (Simple Version) Programs

Programs Π

- If π is an atomic program, then π; is a program
- If α and γ are programs, then $\alpha\gamma$ is a program
- If *b* is a variable-free term of type `boolean`, α and γ programs, then

  ```
  if (b) {α} else {γ};
  ```

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

  ```
  while (b) {α};
  ```

  is a program

Dynamic Logic Syntax Example

An admissible DL program α:

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

What does α compute?

Dynamic Logic (Simple Version) Terms

Terms

Defined as in FOL using also $\mathcal{PV}$, **but**:

Rigid versus Flexible

- **rigid** symbols, same interpretation in **all** execution states
  Needed, for example, to hold initial value of program variable
  Logical variables and predefined functions/predicates are rigid
- **non-rigid** (or **flexible**) terms, interpretation depends on state
  Needed to capture state change after program execution
  Program variables are flexible

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

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, **but**
  Program variables are never bound in quantifiers
- If α is a program and ϕ a DL formula then $\langle\alpha\rangle\phi$ is a DL formula
  $[\alpha]\,\phi$ is a DL-Formula

Programs contain no logical variables
Modalities can be arbitrarily nested

Dynamic Logic Syntax Example

$\backslash\texttt{forall}\ \mathit{int}\ y; ((\langle\!\langle \mathtt{x} = 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y})\ \texttt{<->}\ (\langle\!\langle \mathtt{x} = 1 * 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y}))$ **Syntax ?**

Dynamic Logic Syntax Example

$\backslash\texttt{forall}\ \textit{int}\ y;\,((\langle\!\langle \mathtt{x} = 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y})\ \texttt{<->}\ (\langle\!\langle \mathtt{x} = 1 * 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y}))$ ok

Dynamic Logic Syntax Example

$\backslash\texttt{forall}\ \textit{int}\ y; ((\langle\!\langle \mathtt{x} = 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y})\ \texttt{<->}\ (\langle\!\langle \mathtt{x} = 1 * 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y}))$ ok

$\backslash\texttt{exists}\ \textit{int}\ \mathtt{x}; ([\mathtt{x} = 1;]\,(\mathtt{x} \doteq 1))$ **Syntax ?**

Dynamic Logic Syntax Example

$\backslash\texttt{forall}\ \mathit{int}\ y; ((\langle\!\langle \mathtt{x} = 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y})\ \texttt{<->}\ (\langle\!\langle \mathtt{x} = 1 * 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y}))$ ok

$\backslash\texttt{exists}\ \mathit{int}\ \mathtt{x}; ([\mathtt{x} = 1;]\,(\mathtt{x} \doteq 1))$ bad

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

Dynamic Logic Syntax Example

$\backslash\texttt{forall}\ \mathit{int}\ y; ((\langle\!\langle \texttt{x} = 1; \rangle\!\rangle \texttt{x} \doteq \texttt{y})\ \texttt{<->}\ (\langle\!\langle \texttt{x} = 1 * 1; \rangle\!\rangle \texttt{x} \doteq \texttt{y}))$ ok

$\backslash\texttt{exists}\ \mathit{int}\ \texttt{x}; ([\texttt{x} = 1;]\, (\texttt{x} \doteq 1))$ bad

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

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

Dynamic Logic Syntax Example

$\backslash\texttt{forall}\ \mathit{int}\ y; ((\langle\!\langle \mathtt{x} = 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y})\ \texttt{<->}\ (\langle\!\langle \mathtt{x} = 1 * 1; \rangle\!\rangle \mathtt{x} \doteq \mathtt{y}))$ ok

$\backslash\texttt{exists}\ \mathit{int}\ \mathtt{x}; ([\mathtt{x} = 1;]\,(\mathtt{x} \doteq 1))$ bad

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

$\langle \mathtt{x} = 1; \rangle\,([\texttt{while}\ (\mathrm{true})\ \{\}]\,\mathrm{false})$ ok

- Program formulas can appear nested

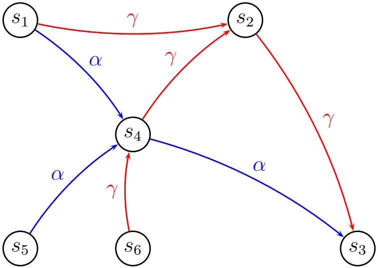
Dynamic Logic Semantics

Definition (Kripke structure)

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

- $s = (U, I) \in S$ is a **State**/**Interpretation** and
- $\rho : \Pi \to (S \to S)$ $\rho(\alpha)$, $\rho(\gamma)$ an admissible relation

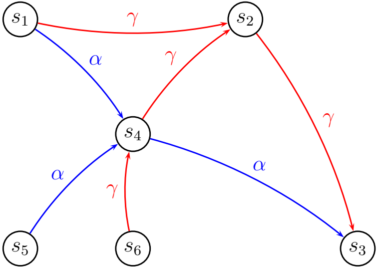
Each state is first-order interpretation



Dynamic Logic Semantics (Cont'd)

Definition (Program Formulas)

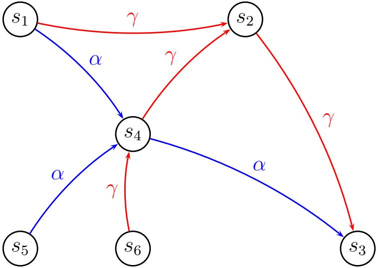
- $s, \beta \models \langle\alpha\rangle\phi$ iff $\rho(\alpha)(s), \beta \models \phi$ and $\rho(\alpha)(s)$ defined
  α **terminates and ϕ is true in the final state after execution**



Dynamic Logic Semantics (Cont'd)

Definition (Program Formulas)

- $s, \beta \models \langle\alpha\rangle\phi$ iff $\rho(\alpha)(s), \beta \models \phi$ and $\rho(\alpha)(s)$ defined
  α **terminates and ϕ is true in the final state after execution**
- $s, \beta \models [\alpha]\,\phi$ iff $\rho(\alpha)(s), \beta \models \phi$ whenever $\rho(\alpha)(s)$ defined
  If α terminates then ϕ is true in the final state after execution



Program Correctness

- $s, \beta \models \langle\alpha\rangle\phi$
  α **totally correct** (with respect to ϕ) in s, β

Program Correctness

- $s, \beta \models \langle\alpha\rangle\phi$
  α **totally correct** (with respect to ϕ) in s, β
- $s, \beta \models [\alpha]\,\phi$
  α **partially correct** (with respect to ϕ) in s, β

Program Correctness

- $s, \beta \models \langle\alpha\rangle\phi$
  α **totally correct** (with respect to ϕ) in s, β
- $s, \beta \models [\alpha]\,\phi$
  α **partially correct** (with respect to ϕ) in s, β
- **Duality** $\langle\alpha\rangle\phi$ iff $!\,[\alpha]\,!\,\phi$
  Exercise: justify this with semantic definitions

Program Correctness

- $s, \beta \models \langle\alpha\rangle\phi$
  α **totally correct** (with respect to ϕ) in s, β
- $s, \beta \models [\alpha]\,\phi$
  α **partially correct** (with respect to ϕ) in s, β
- **Duality** $\langle\alpha\rangle\phi$ iff $!\,[\alpha]\,!\,\phi$
  Exercise: justify this with semantic definitions
- **Implication** if $\langle\alpha\rangle\phi$ then $[\alpha]\,\phi$

Semantics of Sequents

Validity of DL sequents compatible validity FOL sequents
Γ ==> Δ is **valid** iff it is true in **all states** s in **all Kripke structures** K

Semantics of Sequents

Validity of DL sequents compatible validity FOL sequents
Γ ==> Δ is **valid** iff it is true in **all states** s in **all Kripke structures** K

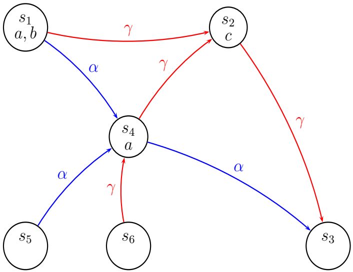
How to restrict validity to set of **initial states** $\mathcal{J} \subseteq S$?

1. Design closed FOL formula Init with
   $s \models \mathrm{Init}$ iff $s \in \mathcal{J}$
2. Use sequent Γ, Init ==> Δ

Later: simple method for specifying **initial value** of program variables

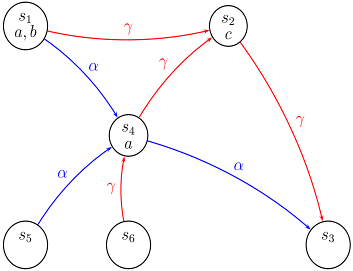
Dynamic Logic Semantics Example

Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



Dynamic Logic Semantics Example

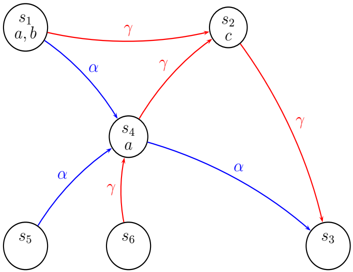
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



$s_1 \models \langle\alpha\rangle \mathtt{a}$?

Dynamic Logic Semantics Example

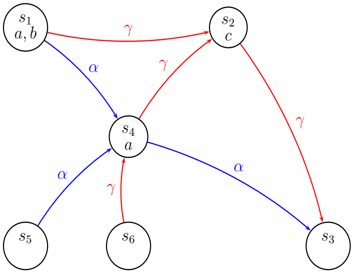
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



$s_1 \models \langle\alpha\rangle a$ (ok),

Dynamic Logic Semantics Example

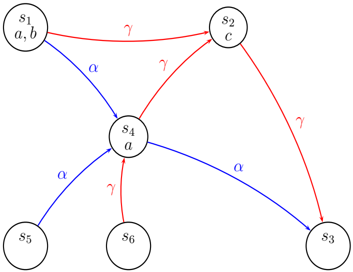
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



$s_1 \models \langle\alpha\rangle\mathtt{a}$ (ok), $\quad s_1 \models \langle\gamma\rangle\mathtt{a}$?

Dynamic Logic Semantics Example

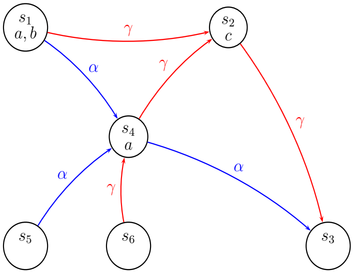
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



$s_1 \models \langle\alpha\rangle \mathtt{a}$ (ok), $\quad s_1 \models \langle\gamma\rangle \mathtt{a}$ (—)

Dynamic Logic Semantics Example

Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

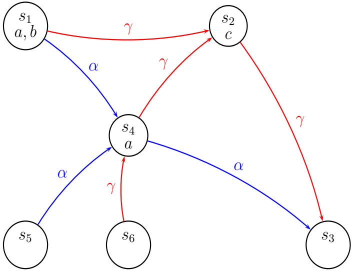


$s_1 \models \langle\alpha\rangle \mathtt{a}$ (ok), $s_1 \models \langle\gamma\rangle \mathtt{a}$ (—)

$s_5 \models \langle\gamma\rangle \mathtt{a}$?

Dynamic Logic Semantics Example

Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

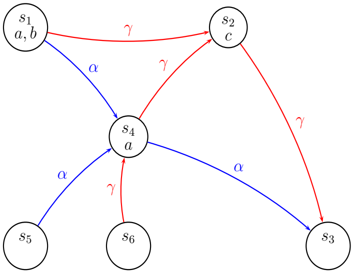


$s_1 \models \langle\alpha\rangle \mathtt{a}$ (ok), $\quad s_1 \models \langle\gamma\rangle \mathtt{a}$ (—)

$s_5 \models \langle\gamma\rangle \mathtt{a}$ (—),

Dynamic Logic Semantics Example

Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$

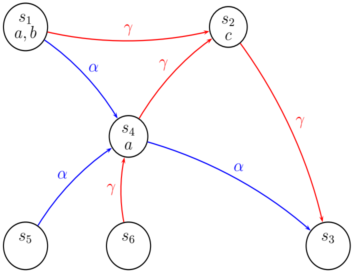


$s_1 \models \langle\alpha\rangle\mathtt{a}$ (ok), $s_1 \models \langle\gamma\rangle\mathtt{a}$ (—)

$s_5 \models \langle\gamma\rangle\mathtt{a}$ (—), $s_5 \models [\gamma]\,\mathtt{a}$?

Dynamic Logic Semantics Example

Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



$s_1 \models \langle\alpha\rangle \mathtt{a}$ (ok), $\quad s_1 \models \langle\gamma\rangle \mathtt{a}$ (—)

$s_5 \models \langle\gamma\rangle \mathtt{a}$ (—), $\quad s_5 \models [\gamma]\, \mathtt{a}$ (ok)

Dynamic Logic Semantics: States, Updates

- States $s = (U, I)$ have all the same universe U
  May assume $\rho(\alpha)$ works on interpretations I
  Define $I, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (U, I)$

Dynamic Logic Semantics: States, Updates

- States $s = (U, I)$ have all the same universe U
  May assume $\rho(\alpha)$ works on interpretations I
  Define $I, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (U, I)$
- Program variables are flexible
  Consider program variables as flexible constants in s with value $I(\mathtt{x})$

Dynamic Logic Semantics: States, Updates

- States $s = (U, I)$ have all the same universe U
  May assume $\rho(\alpha)$ works on interpretations I
  Define $I, \beta \models \phi$ as $s, \beta \models \phi$, where $s = (U, I)$
- Program variables are flexible
  Consider program variables as flexible constants in s with value $I(\mathrm{x})$

State update (cf. updated variable assignment) of I at y with $d \in U$

$$I_{\mathrm{y}}^{d}(\mathrm{x}) = \begin{cases} I(\mathrm{x}) & \mathrm{x} \neq \mathrm{y} \\ d & \mathrm{x} = \mathrm{y} \end{cases}$$

Operational Semantics of Programs

State transformation ρ defines **semantics of programs**
Same ρ for all programs, so not part of s; given β

- $\rho(\texttt{x = t;})(I) = I_{\texttt{x}}^{t^{I,\beta}}$

Operational Semantics of Programs

State transformation ρ defines **semantics of programs**
Same ρ for all programs, so not part of s; given β

- $\rho(\texttt{x = t;})(I) = I_{\texttt{x}}^{t^{I,\beta}}$
- $\rho(\texttt{if } (\mathbf{b})\ \{\alpha\} \texttt{ else } \{\gamma\}\texttt{;})(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \texttt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$

Operational Semantics of Programs

State transformation ρ defines **semantics of programs**
Same ρ for all programs, so not part of s; given β

- $\rho(\texttt{x = t;})(I) = I_{\texttt{x}}^{t^{I,\beta}}$
- $\rho(\texttt{if } (\mathbf{b}) \ \{\alpha\} \texttt{ else } \{\gamma\}\texttt{;})(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \texttt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, **undefined otherwise**

Operational Semantics of Programs

State transformation ρ defines **semantics of programs**
Same ρ for all programs, so not part of s; given β

- $\rho(\texttt{x = t;})(I) = I_{\texttt{x}}^{t^{I,\beta}}$
- $\rho(\texttt{if (}\mathbf{b}\texttt{) \{}\alpha\texttt{\} else \{}\gamma\texttt{\};})(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \texttt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, **undefined otherwise**
- $\rho(\texttt{while (}\mathbf{b}\texttt{) \{}\alpha\texttt{\};})(I) = I'$ iff there are $I = I_0, \ldots, I_n = I'$ such that

Operational Semantics of Programs

State transformation ρ defines **semantics of programs**
Same ρ for all programs, so not part of s; given β

- $\rho(\mathtt{x\ =\ t;})(I) = I_{\mathtt{x}}^{t^{I,\beta}}$
- $\rho(\mathtt{if}\ (\mathbf{b})\ \{\alpha\}\ \mathtt{else}\ \{\gamma\};)(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \mathtt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, **undefined otherwise**
- $\rho(\mathtt{while}\ (\mathbf{b})\ \{\alpha\};)(I) = I'$ iff there are $I = I_0, \ldots, I_n = I'$ such that
  - $I_j, \beta \models b \doteq \mathtt{TRUE}$ 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; given β

- $\rho(\mathtt{x\ =\ t;})(I) = I_{\mathtt{x}}^{t^{I,\beta}}$
- $\rho(\mathtt{if}\ (\mathbf{b})\ \{\alpha\}\ \mathtt{else}\ \{\gamma\};)(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \mathtt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, **undefined otherwise**
- $\rho(\mathtt{while}\ (\mathbf{b})\ \{\alpha\};)(I) = I'$ iff there are $I = I_0, \ldots, I_n = I'$ such that
  - $I_j, \beta \models b \doteq \mathtt{TRUE}$ for $0 \leq j < n$
  - $\rho(\alpha)(I_j) = I_{j+1}$ 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; given β

- $\rho(\texttt{x = t;})(I) = I_{\texttt{x}}^{t^{I,\beta}}$
- $\rho(\texttt{if (}\mathbf{b}\texttt{) \{}\alpha\texttt{\} else \{}\gamma\texttt{\};})(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \texttt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, if $\rho(\alpha)(I)$ defined, **undefined otherwise**
- $\rho(\texttt{while (}\mathbf{b}\texttt{) \{}\alpha\texttt{\};})(I) = I'$ iff there are $I = I_0, \ldots, I_n = I'$ such that
  - $I_j, \beta \models b \doteq \texttt{TRUE}$ for $0 \leq j < n$
  - $\rho(\alpha)(I_j) = I_{j+1}$ for $0 \leq j < n$
  - $I_n, \beta \models b \doteq \texttt{FALSE}$ **undefined otherwise**

Dynamic Logic Examples

Partial correctness assertion (Hoare formula)

$$\{\psi\}\ \alpha\ \{\phi\}$$

If α is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ -> $[\alpha]\,\phi$

Dynamic Logic Examples

Partial correctness assertion (Hoare formula)

$$\{\psi\}\ \alpha\ \{\phi\}$$

If α is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ -> $[\alpha]\,\phi$

Valid formulas

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

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \alpha\ \{\phi\}$$

If α is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ -> $[\alpha]\,\phi$

Valid formulas

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

Dynamic Logic Examples

Partial correctness assertion (Hoare formula)

$$\{\psi\}\ \alpha\ \{\phi\}$$

If α is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ `->` $[\alpha]\,\phi$

Valid formulas

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

Validity depends on α, γ

`\forall` *int* y; $((\langle\alpha\rangle\mathtt{x} \doteq \mathtt{y})$ `<->` $(\langle\gamma\rangle\mathtt{x} \doteq \mathtt{y}))$ **meaning ?**

Dynamic Logic Examples

Partial correctness assertion **(Hoare formula)**

$$\{\psi\}\ \alpha\ \{\phi\}$$

If α is started in a state satisfying ψ and terminates, then its final state satisfies ϕ

In DL ψ `->` $[\alpha]\,\phi$

Valid formulas

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

Validity depends on α, γ

`\forall` *int y*; $((\langle\alpha\rangle\mathtt{x} \doteq \mathtt{y})$ `<->` $(\langle\gamma\rangle\mathtt{x} \doteq \mathtt{y}))$ α,γ **equiv.** relative to `x`

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:

$$\langle \xi\gamma \rangle \phi, \qquad [\xi\gamma]\, \phi$$

where ξ one **single** admissible program statement

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:

$$\langle \xi\gamma \rangle \phi, \qquad [\xi\gamma]\,\phi$$

where ξ one **single** admissible program statement

Rules **execute symbolically** the first active statement
Proof corresponds to symbolic program execution

Dynamic Logic Calculus

$$\text{CONCATENATE}\ \frac{\Gamma\ \texttt{==>}\ \langle\alpha\rangle\left(\langle\gamma\rangle\phi\right),\Delta}{\Gamma\ \texttt{==>}\ \langle\alpha\gamma\rangle\phi,\Delta}$$

Dynamic Logic Calculus

$$\text{CONCATENATE}\ \frac{\Gamma\ \texttt{==>}\ \langle\alpha\rangle\left(\langle\gamma\rangle\phi\right),\Delta}{\Gamma\ \texttt{==>}\ \langle\alpha\gamma\rangle\phi,\Delta}$$

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

Dynamic Logic Calculus

$$\text{CONCATENATE}\ \frac{\Gamma\ \texttt{==>}\ \langle\alpha\rangle\,(\langle\gamma\rangle\phi)\,,\Delta}{\Gamma\ \texttt{==>}\ \langle\alpha\gamma\rangle\phi,\Delta}$$

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

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

Dynamic Logic Calculus

$$\text{CONCATENATE}\ \frac{\Gamma \texttt{ ==> } \langle\alpha\rangle\left(\langle\gamma\rangle\phi\right), \Delta}{\Gamma \texttt{ ==> } \langle\alpha\gamma\rangle\phi, \Delta}$$

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

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

$$\text{ASSIGNMENT}\ \frac{\Gamma^{x/x'}, x \doteq t \texttt{ ==> } \phi, \Delta^{x/x'}}{\Gamma \texttt{ ==> } \langle \texttt{x} = \texttt{t};\rangle\phi, \Delta}$$

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

But: rules dealing with programs need to account for updates

Assignment Rule Using Updates

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

Avoids renaming of program variables
But: rules dealing with programs need to account for updates

Rules work on **first active statement** after **prefix**, followed by **postfix** (remaining code)
Explicit concatenation rule not longer needed

Assignment Rule Using Updates

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

Avoids renaming of program variables
But: rules dealing with programs need to account for updates

Rules work on **first active statement** after **prefix**, followed by **postfix** (remaining code)
Explicit concatenation rule not longer needed

General form of conclusion in rule for symbolic execution

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

Prefix Active statement Postfix

Explicit State Updates

Updates record state change

Explicit State Updates

Updates record state change

Syntax

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

Explicit State Updates

Updates record state change

Syntax

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

Semantics

$I, \beta \models \{\mathtt{v} := t\}\phi \quad$ iff $\quad I_{\mathtt{v}}^{t^{I,\beta}}, \beta \models \phi$

Semantics identical to assignment

Updates work as "lazy" assignments

Computing Effect of Updates

Update followed by **program variable**

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

Computing Effect of Updates

Update followed by **program variable**

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

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

Update followed by **program variable**

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

Update followed by **complex term**

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

Update followed by **first-order formula**

$$\{\mathtt{x} := t\}(\phi \& \psi) \quad \leadsto \quad \{\mathtt{x} := t\}\phi \;\&\; \{\mathtt{x} := t\}\psi$$
$$\{\mathtt{x} := t\}(\backslash\mathtt{forall}\ z\ y; \phi) \quad \leadsto \quad \backslash\mathtt{forall}\ z\ y; (\{\mathtt{x} := t\}\phi) \quad \text{etc.}$$

Computing Effect of Updates

Update followed by **program variable**

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

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

Update followed by **complex term**

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

Update followed by **first-order formula**

$\{\mathtt{x} := t\}(\phi \& \psi) \quad \leadsto \quad \{\mathtt{x} := t\}\phi \ \& \ \{\mathtt{x} := t\}\psi$

$\{\mathtt{x} := t\}(\backslash\mathtt{forall}\ z\ y; \phi) \quad \leadsto \quad \backslash\mathtt{forall}\ z\ y; (\{\mathtt{x} := t\}\phi) \quad \text{etc.}$

Update followed by **program formula**

$\{\mathtt{x} := t\}(\langle\!\langle\alpha\rangle\!\rangle\phi) \quad \leadsto \quad \{\mathtt{x} := t\}(\langle\!\langle\alpha\rangle\!\rangle\phi)$

Update computation delayed until α symbolically executed

Example Proof

```
\programVariables {
  int i;
  int j;
}
\problem {
  \forall int x; \forall int y;
        ( i=x & j=y ->
             \<{int h = i; i = j; j = h;}\> (i=y & j=x) )
}
```

Intuitive Meaning? Satisfiable? Valid?

Demo

`~/Intro/exchange.key`