Introduction to Dynamic Logics

Andreas Roth, Richard Bubel

November 15, 2006

Syntax of Propositional Logic

Definition (Signature $\Sigma = (\mathcal{P}, \mathcal{O})$)

Propositional Variables $\mathcal{P} = \{p_i | i \in \boldsymbol{N}\}$

Syntax of Propositional Logic

Definition (Signature $\Sigma = (\mathcal{P}, \mathcal{O})$)

Propositional Variables	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$
Connectives	$\mathcal{O} = \{\texttt{true}, \texttt{false}, \texttt{\&}, \texttt{\|}, \texttt{!}, \texttt{->}, \texttt{<->}\}$

Syntax of Propositional Logic

Definition (Signature $\Sigma = (\mathcal{P}, \mathcal{O})$)

Propositional Variables	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$
Connectives	$\mathcal{O} = \{\texttt{true}, \texttt{false}, \texttt{\&}, \texttt{\|}, \texttt{!}, \texttt{->}, \texttt{<->}\}$

Definition (Propositional Formulas For_0^{Σ})

- Truth constants '`true`', '`false`' and variables $\mathcal{P}$ are formulas

Syntax of Propositional Logic

Definition (Signature $\Sigma = (\mathcal{P}, \mathcal{O})$)

Propositional Variables	$\mathcal{P} = \{p_i \mid i \in \mathbb{N}\}$
Connectives	$\mathcal{O} = \{\texttt{true}, \texttt{false}, \texttt{\&}, \texttt{\|}, \texttt{!}, \texttt{->}, \texttt{<->}\}$

Definition (Propositional Formulas For_0^{Σ})

- Truth constants '`true`', '`false`' and variables $\mathcal{P}$ are formulas
- If G and H are formulas then

  $$\texttt{!}G, \quad (G \texttt{\&} H), \quad (G \texttt{|} H), \quad (G \texttt{->} H), \quad (G \texttt{<->} H)$$

  are also formulas

Syntax of Propositional Logic

Definition (Signature $\Sigma = (\mathcal{P}, \mathcal{O})$)

Propositional Variables	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$
Connectives	$\mathcal{O} = \{\texttt{true}, \texttt{false}, \texttt{\&}, \texttt{\|}, \texttt{!}, \texttt{->}, \texttt{<->}\}$

Definition (Propositional Formulas For_0^Σ)

- Truth constants '`true`', '`false`' and variables $\mathcal{P}$ are formulas
- If G and H are formulas then

  $$!G, \quad (G\&H), \quad (G \mid H), \quad (G\text{->}H), \quad (G\text{<->}H)$$

  are also formulas
- There are no other formulas

Semantic Notions

Given the functions

- $I : \mathcal{P} \rightarrow \{true, false\}$ and
- its continuation $val_I : For_0^\Sigma \rightarrow \{true, false\}$

Semantic Notions

Given the functions

- $I : \mathcal{P} \to \{true, false\}$ and
- its continuation $val_I : For_0^\Sigma \to \{true, false\}$

Let $G \in For_0^\Sigma$, $\Gamma \subset For_0^\Sigma$

- I is a **model** for G iff $val_I(G) = true$ (write: $I \models G$)
  A formula that has a model is **satisfable**

Semantic Notions

Given the functions

- $I : \mathcal{P} \to \{true, false\}$ and
- its continuation $val_I : For_0^\Sigma \to \{true, false\}$

Let $G \in For_0^\Sigma$, $\Gamma \subset For_0^\Sigma$

- I is a **model** for G iff $val_I(G) = true$ (write: $I \models G$)
  A formula that has a model is **satisfable**
- G **follows from** Γ ($\Gamma \models G$) iff for all interpretations I:

$$I \models H \text{ for all } H \in \Gamma \text{ then also } I \models G$$

Semantic Notions

Given the functions

- $I : \mathcal{P} \rightarrow \{true, false\}$ and
- its continuation $val_I : For_0^\Sigma \rightarrow \{true, false\}$

Let $G \in For_0^\Sigma$, $\Gamma \subset For_0^\Sigma$

- I is a **model** for G iff $val_I(G) = true$ (write: $I \models G$)
  A formula that has a model is **satisfable**
- G **follows from** Γ ($\Gamma \models G$) iff for all interpretations I:
  $$I \models H \text{ for all } H \in \Gamma \text{ then also } I \models G$$
- If any interpretation is a model of G, i.e
  $$\emptyset \models G \quad (\textbf{short}: \ \models G)$$
  then G is called **valid**

Propositional Logic Example

$$p \ \& \ ((\,!\,p) \mid q)$$

- Satisfiable?

Propositional Logic Example

$$p \mathbin{\&} ((!\,p) \mid q)$$

- Satisfiable? Yes

Propositional Logic Example

$$p \,\&\, ((!\,p) \mid q)$$

- Satisfiable? Yes
- Model?

Propositional Logic Example

$$p \,\&\, ((\,!\,p) \mid q)$$

- Satisfiable? Yes
- Model? $I(p) = true$, $I(q) = true$

Propositional Logic Example

$$p \,\&\, ((\,!\,p) \mid q)$$

- Satisfiable? Yes
- Model? $I(p) = true$, $I(q) = true$

$$p \,\&\, ((\,!\,p) \mid q) \models q \mid r$$

Does this hold?

Propositional Logic Example

$$p \mathbin{\&} ((!\,p) \mid q)$$

- Satisfiable? Yes
- Model? $I(p) = \mathit{true}$, $I(q) = \mathit{true}$

$$p \mathbin{\&} ((!\,p) \mid q) \models q \mid r$$

Does this hold? Yes Why?

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**

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

has same semantics as

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

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) $\models (\Gamma_1 \texttt{==>} \Delta_1 \,\&\, \cdots \,\&\, \Gamma_r \texttt{==>} \Delta_r) \texttt{->} (\Gamma \texttt{==>} \Delta)$

Sequent Calculus Rules

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

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

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

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

Sequent Calculus Rules

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

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

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

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

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 $\mathcal{S}_1, \ldots, \mathcal{S}_r$
- tree structure with goal as root
- **close** proof branch when rule without premise encountered

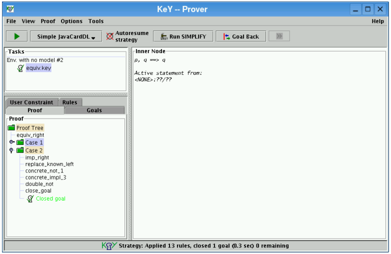
Sequent Calculus Proofs

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	$\dfrac{\Gamma \texttt{ ==> } A, \Delta}{\Gamma, !A \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } \Delta}{\Gamma \texttt{ ==> } !A, \Delta}$
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	$\dfrac{\Gamma, A \texttt{ ==> } \Delta \quad \Gamma, B \texttt{ ==> } \Delta}{\Gamma, A \mid B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } A, B, \Delta}{\Gamma \texttt{ ==> } A \mid B, \Delta}$

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	$\dfrac{\Gamma, A \texttt{ ==> } \Delta \quad \Gamma, B \texttt{ ==> } \Delta}{\Gamma, A \mid B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } A, B, \Delta}{\Gamma \texttt{ ==> } A \mid B, \Delta}$
imp	$\dfrac{\Gamma \texttt{ ==> } A, \Delta \quad \Gamma, B \texttt{ ==> } \Delta}{\Gamma, A\texttt{->}B \texttt{ ==> } \Delta}$	$\dfrac{\Gamma, A \texttt{ ==> } B, \Delta}{\Gamma \texttt{ ==> } A\texttt{->}B, \Delta}$

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

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

Justification of Rules

Compute rules by applying semantics definition of connectives

Justification of Rules

Compute rules by applying semantics definition of connectives

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

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

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

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

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

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

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

Propositional Logic is insufficient

Propositional Logic is insufficient

A ALL PERSONS ARE HAPPY

Propositional Logic is insufficient

A	ALL PERSONS ARE HAPPY
B	PAT IS A PERSON

Propositional Logic is insufficient

A	All persons are happy
B	Pat is a person
?	Pat is happy

Propositional Logic is insufficient

$$\frac{\begin{array}{c} A \\ B \end{array}}{?} \qquad \frac{\begin{array}{l} \textsc{All persons are happy} \\ \textsc{Pat is a person} \end{array}}{\textsc{Pat is happy}}$$

Propositional logic lacks possibility to talk about individuals
In particular, need to model objects, attributes, associations, etc.

Propositional Logic is insufficient

A	All persons are happy
B	Pat is a person
?	Pat is happy

Propositional logic lacks possibility to talk about individuals
In particular, need to model objects, attributes, associations, etc.

$\Rightarrow$ First-Order Logic (FOL)

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols $\quad \mathcal{T} = \{z_1, \ldots, z_r\}$, $r \geq 1$, partial order $\prec$

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols	$\mathcal{T} = \{z_1, \ldots, z_r\}, \ r \geq 1,$ partial order $\prec$
Variables	$\mathcal{V} = \{x_i \mid i \in \boldsymbol{N}\}$

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols	$\mathcal{T} = \{z_1, \ldots, z_r\}$, $r \geq 1$, partial order $\prec$
Variables	$\mathcal{V} = \{x_i \mid i \in \boldsymbol{N}\}$
Predicate Symbols	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols	$\mathcal{T} = \{z_1, \ldots, z_r\}$, $r \geq 1$, partial order $\prec$
Variables	$\mathcal{V} = \{x_i \mid i \in \boldsymbol{N}\}$
Predicate Symbols	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$
Function Symbols	$\mathcal{F} = \{f_i^z \mid i \in \boldsymbol{N},\ z \in \mathcal{T}\}$

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols	$\mathcal{T} = \{z_1, \ldots, z_r\}, \; r \geq 1, \;$ partial order $\prec$
Variables	$\mathcal{V} = \{x_i \mid i \in \boldsymbol{N}\}$
Predicate Symbols	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$
Function Symbols	$\mathcal{F} = \{f_i^z \mid i \in \boldsymbol{N}, \, z \in \mathcal{T}\}$

for $q \in \mathcal{P} \cup \mathcal{F}$ let $\alpha(q) \in \boldsymbol{N}$ arity and $\sigma(q) \in \mathcal{T}^{\alpha(q)}$ signature of q

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols	$\mathcal{T} = \{z_1, \ldots, z_r\}$, $r \geq 1$, partial order $\prec$
Variables	$\mathcal{V} = \{x_i \mid i \in \boldsymbol{N}\}$
Predicate Symbols	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$
Function Symbols	$\mathcal{F} = \{f_i^z \mid i \in \boldsymbol{N}, z \in \mathcal{T}\}$

for $q \in \mathcal{P} \cup \mathcal{F}$ let $\alpha(q) \in \boldsymbol{N}$ arity and $\sigma(q) \in \mathcal{T}^{\alpha(q)}$ signature of q

Connectives	$\mathcal{O} = \{$`true`, `false`, `&`, `|`, `!`, `->`, `<->`$\}$

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols	$\mathcal{T} = \{z_1, \ldots, z_r\}$, $r \geq 1$, partial order $\prec$
Variables	$\mathcal{V} = \{x_i \mid i \in \boldsymbol{N}\}$
Predicate Symbols	$\mathcal{P} = \{p_i \mid i \in \boldsymbol{N}\}$
Function Symbols	$\mathcal{F} = \{f_i^z \mid i \in \boldsymbol{N},\ z \in \mathcal{T}\}$

for $q \in \mathcal{P} \cup \mathcal{F}$ let $\alpha(q) \in \boldsymbol{N}$ arity and $\sigma(q) \in \mathcal{T}^{\alpha(q)}$ signature of q

Connectives	$\mathcal{O} = \{\texttt{true}, \texttt{false}, \texttt{\&}, \texttt{\|}, \texttt{!}, \texttt{->}, \texttt{<->}\}$
Quantifiers	$\mathcal{Q} = \{\texttt{\textbackslash forall}, \texttt{\textbackslash exists}\}$

Signature of First-Order Logic

Definition (Signature)

$\Sigma = (\mathcal{T}, \mathcal{V}, \mathcal{P}, \mathcal{F}, \alpha, \sigma, \mathcal{O} \cup \mathcal{Q} \cup \{\doteq\})$

Type Symbols	$\mathcal{T} = \{z_1, \ldots, z_r\}$, $r \geq 1$, partial order $\prec$
Variables	$\mathcal{V} = \{x_i \mid i \in \mathbf{N}\}$
Predicate Symbols	$\mathcal{P} = \{p_i \mid i \in \mathbf{N}\}$
Function Symbols	$\mathcal{F} = \{f_i^z \mid i \in \mathbf{N}, z \in \mathcal{T}\}$

for $q \in \mathcal{P} \cup \mathcal{F}$ let $\alpha(q) \in \mathbf{N}$ arity and $\sigma(q) \in \mathcal{T}^{\alpha(q)}$ signature of q

Connectives	$\mathcal{O} = \{\texttt{true}, \texttt{false}, \texttt{\&}, \texttt{\|}, \texttt{!}, \texttt{->}, \texttt{<->}\}$
Quantifiers	$\mathcal{Q} = \{\texttt{\\forall}, \texttt{\\exists}\}$
Equality symbol	$\doteq$

First-Order Signature Example

Sticks and stones may break your bones, but words will never hurt

First-Order Signature Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\mathrm{Weapon}, \mathrm{Word}, \mathrm{Any}\}$

$\mathrm{Weapon} \prec \mathrm{Any}, \;\; \mathrm{Word} \prec \mathrm{Any}$

First-Order Signature Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\text{Weapon}, \text{Word}, \text{Any}\}$

$\text{Weapon} \prec \text{Any},\ \text{Word} \prec \text{Any}$

Predicates $\mathcal{P} = \{\text{hurts}\},\ \sigma(\text{hurts}) = \langle\text{Any}\rangle$

First-Order Signature Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\mathrm{Weapon}, \mathrm{Word}, \mathrm{Any}\}$

$\mathrm{Weapon} \prec \mathrm{Any},\ \mathrm{Word} \prec \mathrm{Any}$

Predicates $\mathcal{P} = \{\mathrm{hurts}\},\ \sigma(\mathrm{hurts}) = \langle \mathrm{Any} \rangle$

Functions $\mathcal{F} = \{\mathrm{stick}^{\mathrm{Weapon}}, \mathrm{stone}^{\mathrm{Weapon}}, \mathrm{blockhead}^{\mathrm{Word}}\}$

Terms of First-Order Logic

Definition (Terms)

The set of terms $Term_\Sigma$ is inductively defined as

- Variable $x \in \mathcal{V}$ is term

Terms of First-Order Logic

Definition (Terms)

The set of terms $Term_\Sigma$ is inductively defined as

- Variable $x \in \mathcal{V}$ is term
- If $f^z \in \mathcal{F}$, $\sigma(f) = \langle z_1, \ldots, z_r \rangle$ and t_i term of type $z'_i \prec z_i$ or $t_i \in \mathcal{V}$ for $1 \leq i \leq r$, then $f^z(t_1, \ldots, t_r)$ is term of type z
  When $r = 0$ call it **constant**

Formulas of First-Order Logic

Definition (First-Order Formulas)

The set of first-order formulas For^{Σ} is inductively defined as

- If $p \in \mathcal{P}$, $\sigma(p) = \langle z_1, \ldots, z_r \rangle$ and t_i term of type $z_i' \prec z_i$ or $t_i \in \mathcal{V}$ for $1 \leq i \leq r$, then $p(t_1, \ldots, t_r)$ is a first-order formula

Use brackets and usual precedence rules to avoid syntactic ambiguity

Formulas of First-Order Logic

Definition (First-Order Formulas)

The set of first-order formulas For^{Σ} is inductively defined as

- If $p \in \mathcal{P}$, $\sigma(p) = \langle z_1, \ldots, z_r \rangle$ and t_i term of type $z_i' \prec z_i$ or $t_i \in \mathcal{V}$ for $1 \leq i \leq r$, then $p(t_1, \ldots, t_r)$ is a first-order formula
- If t_1, t_2 are terms of same type or variable, then $t_1 \doteq t_2$ is a first-order formula

Use brackets and usual precedence rules to avoid syntactic ambiguity

Formulas of First-Order Logic

Definition (First-Order Formulas)

The set of first-order formulas For^{Σ} is inductively defined as

- If $p \in \mathcal{P}$, $\sigma(p) = \langle z_1, \ldots, z_r \rangle$ and t_i term of type $z_i' \prec z_i$ or $t_i \in \mathcal{V}$ for $1 \leq i \leq r$, then $p(t_1, \ldots, t_r)$ is a first-order formula
- If t_1, t_2 are terms of same type or variable, then $t_1 \doteq t_2$ is a first-order formula
- Truth constants, connectives as in propositional logic

Use brackets and usual precedence rules to avoid syntactic ambiguity

Formulas of First-Order Logic

Definition (First-Order Formulas)

The set of first-order formulas For^{Σ} is inductively defined as

- If $p \in \mathcal{P}$, $\sigma(p) = \langle z_1, \ldots, z_r \rangle$ and t_i term of type $z_i' \prec z_i$ or $t_i \in \mathcal{V}$ for $1 \leq i \leq r$, then $p(t_1, \ldots, t_r)$ is a first-order formula
- If t_1, t_2 are terms of same type or variable, then $t_1 \doteq t_2$ is a first-order formula
- Truth constants, connectives as in propositional logic
- If $x \in \mathcal{V}$, $z \in \mathcal{T}$, ϕ a first-order formula with no occurrence of $x : z'$, and all occurrences of x in ϕ are in symbols with type signature $z \prec z'$ for the argument where x appears, then `\forall` $z\ x; \phi$, `\exists` $z\ x; \phi$ are first-order formulas; x **declared** of type z and **scope** ϕ

Use brackets and usual precedence rules to avoid syntactic ambiguity

First-Order Syntax Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\mathrm{Weapon}, \mathrm{Word}, \mathrm{Any}\}$

$\mathrm{Weapon} \prec \mathrm{Any},\ \mathrm{Word} \prec \mathrm{Any}$

Predicates $\mathcal{P} = \{\mathrm{hurts}\},\ \sigma(\mathrm{hurts}) = \langle \mathrm{Any} \rangle$

Functions $\mathcal{F} = \{\mathrm{stick}^{\mathrm{Weapon}}, \mathrm{stone}^{\mathrm{Weapon}}, \mathrm{blockhead}^{\mathrm{Word}}\}$

First-Order Syntax Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\text{Weapon}, \text{Word}, \text{Any}\}$

$\text{Weapon} \prec \text{Any}, \ \text{Word} \prec \text{Any}$

Predicates $\mathcal{P} = \{\text{hurts}\}, \ \sigma(\text{hurts}) = \langle \text{Any} \rangle$

Functions $\mathcal{F} = \{\text{stick}^{\text{Weapon}}, \text{stone}^{\text{Weapon}}, \text{blockhead}^{\text{Word}}\}$

$$\backslash\texttt{forall}\ \text{Weapon}\ x; \text{hurts}(x) \quad \texttt{\&}$$

$$\backslash\texttt{forall}\ \text{Word}\ y; \texttt{!}\,\text{hurts}(y)$$

Semantics of First-Order Logic

Definition (Interpretation)

An *interpretation* $\mathcal{D} = (U, I)$ consists of

- U is the non-empty **universe**
  For each type z there is a subuniverse U^z such that $U^z \subseteq U^{z'}$ if $z \prec z'$

Semantics of First-Order Logic

Definition (Interpretation)

An *interpretation* $\mathcal{D} = (U, I)$ consists of

- U is the non-empty **universe**
  For each type z there is a subuniverse U^z such that $U^z \subseteq U^{z'}$ if $z \prec z'$
- If $\sigma(p) = \langle z_1, \ldots, z_r \rangle$, then $p^I \subseteq U^{z_1} \times \cdots \times U^{z_r}$

Semantics of First-Order Logic

Definition (Interpretation)

An *interpretation* $\mathcal{D} = (U, I)$ consists of

- U is the non-empty **universe**
  For each type z there is a subuniverse U^z such that $U^z \subseteq U^{z'}$ if $z \prec z'$
- If $\sigma(p) = \langle z_1, \ldots, z_r \rangle$, then $p^I \subseteq U^{z_1} \times \cdots \times U^{z_r}$
- If $\sigma(f^z) = \langle z_1, \ldots, z_r \rangle$, then $f^I : U^{z_1} \times \cdots \times U^{z_r} \to U^z$

Semantics of First-Order Logic

Definition (Interpretation)

An *interpretation* $\mathcal{D} = (U, I)$ consists of

- U is the non-empty **universe**
  For each type z there is a subuniverse U^z such that $U^z \subseteq U^{z'}$ if $z \prec z'$
- If $\sigma(p) = \langle z_1, \dots, z_r \rangle$, then $p^I \subseteq U^{z_1} \times \dots \times U^{z_r}$
- If $\sigma(f^z) = \langle z_1, \dots, z_r \rangle$, then $f^I : U^{z_1} \times \dots \times U^{z_r} \to U^z$

Definition (Variable Assignment)

A *variable assignment* is a function $\beta : \mathcal{V} \to U$

Updated variable assignment: for $d \in U$ let $\beta_y^d(x) = \begin{cases} \beta(x) & x \neq y \\ d & x = y \end{cases}$

Semantics of First-Order Logic, Cont'd

- $x^{\mathcal{D},\beta} = \beta(x)$
- Let $\sigma(f^z) = \langle z_1, \ldots, z_r \rangle$, then
  $(f^z(t_1, \ldots, t_r))^{\mathcal{D},\beta} = f^I((t_1)^{\mathcal{D},\beta}, \ldots, (t_r)^{\mathcal{D},\beta})$
- Let $\sigma(p) = \langle z_1, \ldots, z_r \rangle$, then
  $val_{\mathcal{D},\beta}(p(t_1, \ldots, t_r)) = \begin{cases} true & \langle (t_1)^{\mathcal{D},\beta}, \ldots, (t_r)^{\mathcal{D},\beta} \rangle \in p^I \\ false & \text{otherwise} \end{cases}$
  (Assume that $\beta(t_i) \in U^{z_i}$ when $t_i \in \mathcal{V}$ — **well-defined**:)
- $val_{\mathcal{D},\beta}(\texttt{\textbackslash forall}\ z\ x; \phi) =$
  $\begin{cases} true & \text{for all } u \in U^z : val_{\mathcal{D},\beta_x^u}(\phi) = true \\ false & otherwise \end{cases}$

`\exists` similar than `\forall`, $\doteq$ identity on U

First-Order Semantic Notions

Satisfiability, **truth**, and **validity**

$\mathcal{D}, \beta \models \phi$	iff	$\mathit{val}_{\mathcal{D},\beta}(\phi) = \mathit{true}$	(ϕ is **satisfiable**)
$\mathcal{D} \models \phi$	iff	for all $\beta : \mathcal{D}, \beta \models \phi$	(ϕ is **true** in $\mathcal{D}$)
$\models \phi$	iff	for all $\mathcal{D} : \mathcal{D} \models \phi$	(ϕ is **valid**)

A formula containing only declared variables is **closed**
Closed formulas that are satisfiable are also true: only one notion
For closed formulas, type of variable assignment well-defined

From now on only *closed* formulas are considered.

First-Order Logic Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\text{Weapon}, \text{Word}, \text{Any}\}$

$\text{Weapon} \prec \text{Any}, \ \text{Word} \prec \text{Any}$

Predicates $\mathcal{P} = \{\text{hurts}\}, \ \sigma(\text{hurts}) = \langle \text{Any} \rangle$

Functions $\mathcal{F} = \{\text{stick}^{\text{Weapon}}, \text{stone}^{\text{Weapon}}, \text{blockhead}^{\text{Word}}\}$

$$\backslash\texttt{forall}\ \text{Weapon}\ x; \text{hurts}(x) \quad \texttt{\&} \quad \backslash\texttt{forall}\ \text{Word}\ y; \texttt{!}\,\text{hurts}(y)$$

Satisfiable? Valid?

First-Order Logic Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\text{Weapon}, \text{Word}, \text{Any}\}$

$\text{Weapon} \prec \text{Any},\ \ \text{Word} \prec \text{Any}$

Predicates $\mathcal{P} = \{\text{hurts}\},\ \sigma(\text{hurts}) = \langle\text{Any}\rangle$

Functions $\mathcal{F} = \{\text{stick}^{\text{Weapon}}, \text{stone}^{\text{Weapon}}, \text{blockhead}^{\text{Word}}\}$

$\texttt{\textbackslash forall}\ \text{Weapon}\ x; \text{hurts}(x) \quad \texttt{\&} \quad \texttt{\textbackslash forall}\ \text{Word}\ y; \texttt{!}\text{hurts}(y)$

Satisfiable? Valid? Model

$U^{\text{Weapon}} = \{\mathit{towel}\},\ U^{\text{Word}} = \{\mathit{rosebud}\},\ U = U^{\text{Word}} \cup U^{\text{Weapon}}$

$I(\text{hurts}) = \{\langle\mathit{towel}\rangle\}$

$I(\text{stick}) = I(\text{stone}) = \mathit{towel},\ I(\text{blockhead}) = \mathit{rosebud}$

First-Order Logic Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\text{Weapon}, \text{Word}, \text{Any}\}$
$\text{Weapon} \prec \text{Any},\ \text{Word} \prec \text{Any}$

Predicates $\mathcal{P} = \{\text{hurts}\},\ \sigma(\text{hurts}) = \langle\text{Any}\rangle$

Functions $\mathcal{F} = \{\text{stick}^{\text{Weapon}}, \text{stone}^{\text{Weapon}}, \text{blockhead}^{\text{Word}}\}$

$\backslash\texttt{forall}$ Weapon x; hurts(x) `&` $\backslash\texttt{forall}$ Word y; `!`hurts(y)

How to express that there are **at least two different weapons**?

First-Order Logic Example

Sticks and stones may break your bones, but words will never hurt

Types $\mathcal{T} = \{\text{Weapon}, \text{Word}, \text{Any}\}$

$\text{Weapon} \prec \text{Any}, \ \text{Word} \prec \text{Any}$

Predicates $\mathcal{P} = \{\text{hurts}\}, \ \sigma(\text{hurts}) = \langle \text{Any} \rangle$

Functions $\mathcal{F} = \{\text{stick}^{\text{Weapon}}, \text{stone}^{\text{Weapon}}, \text{blockhead}^{\text{Word}}\}$

$$\backslash\texttt{forall Weapon } x; \text{hurts}(x) \quad \texttt{\&} \quad \backslash\texttt{forall Word } y; \texttt{!}\text{hurts}(y)$$

How to express that there are **at least two different weapons**?

$$\backslash\texttt{exists Weapon } x, y; (\texttt{!} x \doteq y)$$

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	$\dfrac{\Gamma, \texttt{\\forall}\ z\ x; \phi, \{x/t^{z'}\}\phi\ \texttt{==>}\ \Delta}{\Gamma, \texttt{\\forall}\ z\ x; \phi\ \texttt{==>}\ \Delta}$	$\dfrac{\Gamma\ \texttt{==>}\ \{x/c^{z}\}\phi, \Delta}{\Gamma\ \texttt{==>}\ \texttt{\\forall}\ z\ x; \phi, \Delta}$
ex	$\dfrac{\Gamma, \{x/c^{z}\}\phi\ \texttt{==>}\ \Delta}{\Gamma, \texttt{\\exists}\ z\ x; \phi\ \texttt{==>}\ \Delta}$	$\dfrac{\Gamma\ \texttt{==>}\ \{x/t^{z'}\}\phi, \texttt{\\exists}\ z}{\Gamma\ \texttt{==>}\ \texttt{\\exists}\ z\ x; \phi,}$

- $\{t/t'\}\phi$ 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	$\dfrac{\Gamma, \backslash\texttt{forall}\ z\ x; \phi, \{x/t^{z'}\}\phi \texttt{ ==> } \Delta}{\Gamma, \backslash\texttt{forall}\ z\ x; \phi \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } \{x/c^z\}\phi, \Delta}{\Gamma \texttt{ ==> } \backslash\texttt{forall}\ z\ x; \phi, \Delta}$
ex	$\dfrac{\Gamma, \{x/c^z\}\phi \texttt{ ==> } \Delta}{\Gamma, \backslash\texttt{exists}\ z\ x; \phi \texttt{ ==> } \Delta}$	$\dfrac{\Gamma \texttt{ ==> } \{x/t^{z'}\}\phi, \backslash\texttt{exists}\ z}{\Gamma \texttt{ ==> } \backslash\texttt{exists}\ z\ x; \phi,}$
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

$$\begin{array}{lll}\texttt{if}\ (\mathit{seq}) & \texttt{find}\ (\vdash_{\texttt{opt}}\ \Phi) & \texttt{replacewith}(\vdash_{\texttt{opt}}\ \Phi') \\ & & \texttt{add}(\vdash \mathit{seq}) \ldots; \ldots; \ldots \\ & & \texttt{heuristics}(\mathit{name}^{+})\end{array}$$

Taclets - The rule description language of KeY

$$\texttt{if}\ (\mathit{seq})\quad \texttt{find}\ (\vdash_{\texttt{opt}}\ \Phi)\quad \begin{array}{l}\texttt{replacewith}(\vdash_{\texttt{opt}}\ \Phi')\\ \texttt{add}(\vdash\ \mathit{seq})\ldots;\ldots;\ldots\\ \texttt{heuristics}(\mathit{name}^{+})\end{array}$$

SYNTAX

`find`	sequent (max. one formula), formula or term
`if`	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

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

The `and-right` rule as taclet

TEXTBOOK

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

TACLET

```
\find( ⊢ A ∧ B )
   \replacewith ( ⊢ A );
   \replacewith ( ⊢ B )
\rulesets(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	$\mathcal{F} = \{\}$

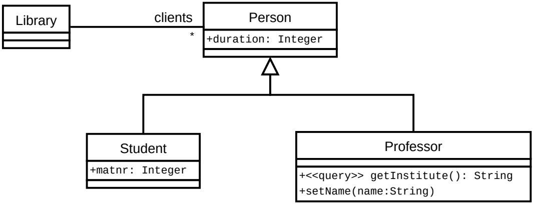
$$(\texttt{\textbackslash exists}\ z\ x; \texttt{\textbackslash exists}\ z\ y; p(x,y)\ \texttt{\&}\ \texttt{\textbackslash forall}\ z\ x; \texttt{!}\,p(x,x))\ \texttt{->}$$
$$\texttt{\textbackslash exists}\ z\ x; \texttt{\textbackslash exists}\ z\ y; (\texttt{!}\,x \doteq y)$$

Intuitive Meaning? Satisfiable? Valid?

Demo

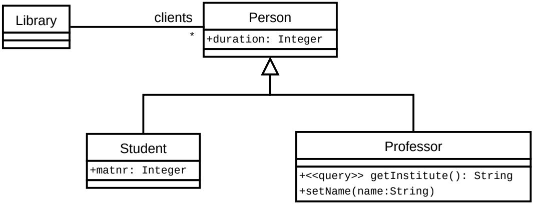
`demo1.key`

Example: JML to FOL



Types?

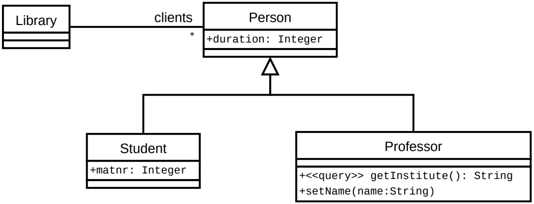
Example: JML to FOL



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

Functions?

Example: JML to FOL



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

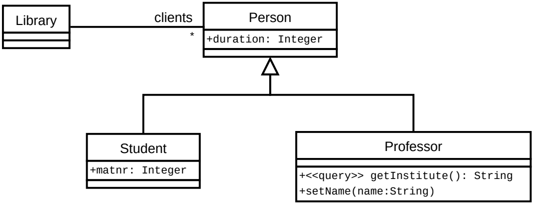
Functions?

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

Queries `String Professor.getInstitute`

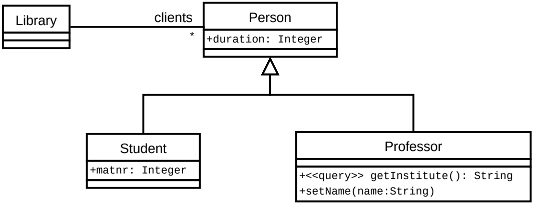
incl. some predefined

Example: JML to FOL



Meaning?

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

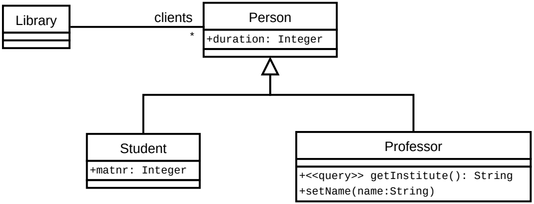


A student is uniquely identified by his/her student id (matnr)

```
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:

$\backslash\texttt{forall}$ Student p1; $\backslash\texttt{forall}$ 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	$\mathcal{V} = \{y_i \mid i \in \boldsymbol{N}\}$
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

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 $\mathcal{T} = \{\texttt{int}, \texttt{boolean}\}$
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 $\texttt{int}$ w/o logical variables
Modal Connectives \\$\langle\cdot\backslash\rangle\cdot$ "diamond", \\$[\cdot\backslash]\cdot$ "box"
First argument program, second argument formula

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 $\pi;$ 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

  ```
  if (b) {α} else {γ};
  ```

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

  $$\textbf{while}\ (\mathrm{b})\ \{\alpha\};$$

  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
  $\backslash\langle\alpha\backslash\rangle\phi$ is a DL formula
  $\backslash[\alpha\backslash]\,\phi$ is a DL-Formula

Programs contain no logical variables
Modalities can be arbitrarily nested

Dynamic Logic Syntax Example

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

Dynamic Logic Syntax Example

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

Dynamic Logic Syntax Example

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

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

Dynamic Logic Syntax Example

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

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

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

Dynamic Logic Syntax Example

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

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

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

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

Dynamic Logic Syntax Example

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

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

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

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

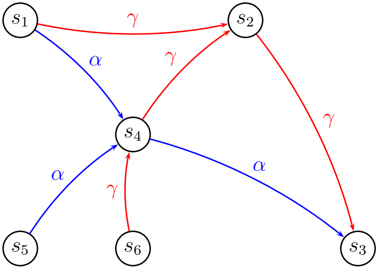
- Program formulas can appear nested

Definition (Kripke structure)

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

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

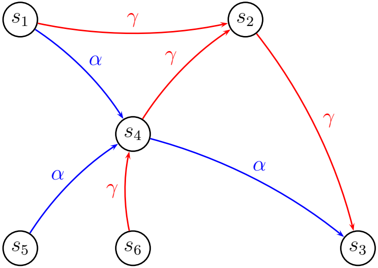
Each state is first-order interpretation



Dynamic Logic Semantics (Cont'd)

Definition (Program Formulas)

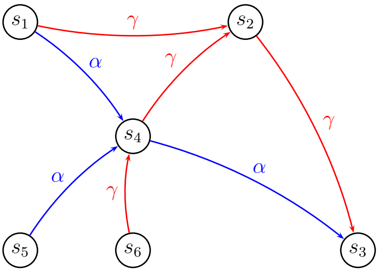
- $s, \beta \models \langle\!\langle \alpha \rangle\!\rangle \phi$ iff $\rho(\alpha)(s), \beta \models \phi$ and $\rho(\alpha)(s)$ defined
  α **terminates and ϕ is true in the final state after execution**



Dynamic Logic Semantics (Cont'd)

Definition (Program Formulas)

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

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

Program Correctness

- $s, \beta \models \langle\!\langle \alpha \rangle\!\rangle \phi$
  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** $\quad \langle\alpha\rangle\phi \quad$ iff $\quad !\,[\alpha]\,!\,\phi$
  Exercise: justify this with semantic definitions
- **Implication** $\quad$ if $\quad \langle\alpha\rangle\phi \quad$ then $\quad [\alpha]\,\phi$

Semantics of Sequents

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

Semantics of Sequents

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

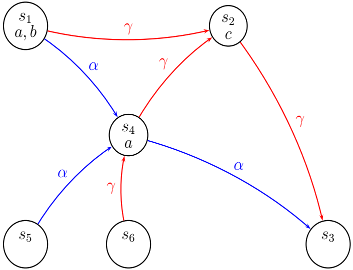
How to restrict validity to set of **initial states** $\mathcal{J} \subseteq S$?

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

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

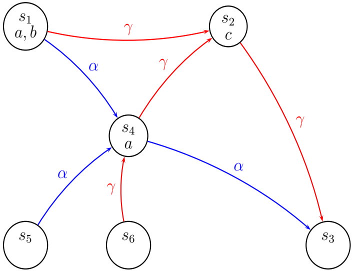
Dynamic Logic Semantics Example

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



Dynamic Logic Semantics Example

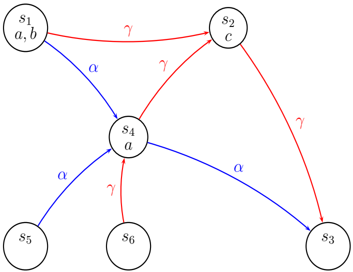
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



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

Dynamic Logic Semantics Example

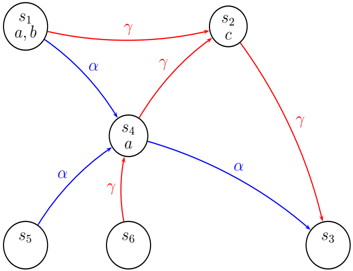
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



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

Dynamic Logic Semantics Example

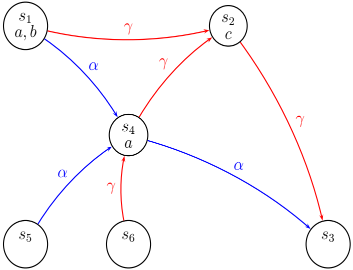
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



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

Dynamic Logic Semantics Example

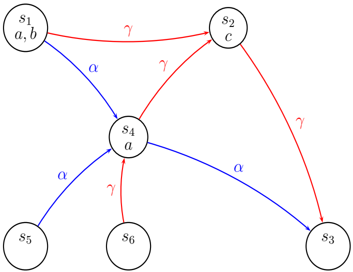
Predicate symbols (prop. vars.) $\mathcal{P} = \{a, b, c\}$



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

Dynamic Logic Semantics Example

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

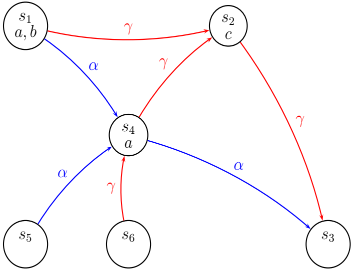


$s_1 \models \langle\!\langle\alpha\rangle\!\rangle \mathtt{a}$ (ok), $\qquad s_1 \models \langle\!\langle\gamma\rangle\!\rangle \mathtt{a}$ (—)

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

Dynamic Logic Semantics Example

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

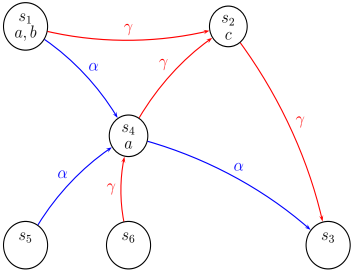


$s_1 \models \langle\!\langle \alpha \rangle\!\rangle a$ (ok), $s_1 \models \langle\!\langle \gamma \rangle\!\rangle a$ (—)

$s_5 \models \langle\!\langle \gamma \rangle\!\rangle a$ (—),

Dynamic Logic Semantics Example

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

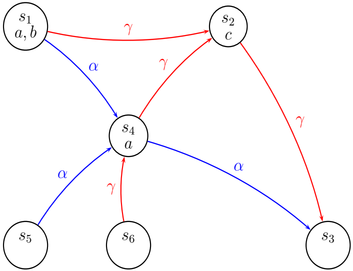


$s_1 \models \langle\alpha\rangle a$ (ok), $s_1 \models \langle\gamma\rangle a$ (—)

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

Dynamic Logic Semantics Example

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



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

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

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

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

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

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**
- $\rho(\texttt{while } (\mathbf{b}) \ \{\alpha\}\texttt{;})(I) = I'$ 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

- $\rho(\texttt{x = t;})(I) = I_{\texttt{x}}^{t^{I,\beta}}$
- $\rho(\texttt{if (}\mathbf{b}\texttt{) \{}\alpha\texttt{\} else \{}\gamma\texttt{\};})(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \texttt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, 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 $0 \leq j < n$

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

- $\rho(\mathtt{x\ =\ t;})(I) = I_{\mathtt{x}}^{t^{I,\beta}}$
- $\rho(\mathtt{if}\ (\mathbf{b})\ \{\alpha\}\ \mathtt{else}\ \{\gamma\};)(I) = \begin{cases} \rho(\alpha)(I) & I, \beta \models b \doteq \mathtt{TRUE} \\ \rho(\gamma)(I) & \text{otherwise} \end{cases}$
- $\rho(\alpha\gamma)(I) = \rho(\gamma)(\rho(\alpha)(I))$, 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 $0 \leq j < n$
  - $I_n, \beta \models b \doteq \mathtt{FALSE}$ **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 -> \[α\] ϕ

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 -> \[α\] ϕ

Valid formulas

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

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 -> \[α\] ϕ

Valid formulas

\[`x` $=$ `1`;\] (`x` $\doteq$ `1`)

\[`while` (true) `{x` $=$ `x;};` \] 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 `->` $\backslash[\alpha\backslash]\ \phi$

Valid formulas

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

Validity depends on α, γ

$\backslash\mathtt{forall}\ \mathit{int}\ y;\ ((\backslash\langle\alpha\backslash\rangle\mathtt{x} \doteq \mathtt{y})$ `<->` $(\backslash\langle\gamma\backslash\rangle\mathtt{x} \doteq \mathtt{y}))$ **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 -> \[α\] ϕ

Valid formulas

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

Validity depends on α, γ

\forall *int y*; ((\<α\>x $\doteq$ y) <-> (\<γ\>x $\doteq$ 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:

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

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:

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

where ξ one **single** admissible program statement

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

Dynamic Logic Calculus

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

Dynamic Logic Calculus

$$\textsc{concatenate}\ \frac{\Gamma \texttt{ ==> } \backslash\langle\alpha\backslash\rangle\,(\backslash\langle\gamma\backslash\rangle\phi)\,,\Delta}{\Gamma \texttt{ ==> } \backslash\langle\alpha\gamma\backslash\rangle\phi,\Delta}$$

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

Dynamic Logic Calculus

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

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

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

Assignment Rule Using Updates

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

Avoids renaming of program variables

But: rules dealing with programs need to account for updates

Assignment Rule Using Updates

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

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

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

Assignment Rule Using Updates

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

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

Solution: 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

$$\Gamma ==> \langle \underbrace{\pi}_{\text{Prefix}} \quad \underbrace{\xi;}_{\text{Active statement}} \quad \underbrace{\omega}_{\text{Postfix}} \rangle\phi, \Delta$$

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

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

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

Update followed by **complex term**

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

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**

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

Computing Effect of Updates

Update followed by **program variable**

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

Update followed by **complex term**

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

Update followed by **first-order formula**

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

Update followed by **program formula**

$$\{\mathtt{x} := t\}(\backslash\langle\alpha\backslash\rangle\phi) \rightsquigarrow \{\mathtt{x} := t\}(\backslash\langle\alpha\backslash\rangle\phi)$$

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

`dlIntro/exchange.key`