Software Engineering using Formal Methods

Reasoning about Programs with Dynamic Logic

Wolfgang Ahrendt

9 October 2014

Part I

Where are we?

Where Are We?

before specification of JAVA programs with JML

Where Are We?

before specification of JAVA programs with JML

now dynamic logic (DL) for resoning about JAVA programs

Where Are We?

before specification of JAVA programs with JML

now dynamic logic (DL) for resoning about JAVA programs

after that generating DL from JML+JAVA

Where Are We?

before specification of JAVA programs with JML

now dynamic logic (DL) for resoning about JAVA programs

after that generating DL from JML+JAVA

$+$ verifying the resulting proof obligations

Motivation

Consider the method

```
public void doubleContent(int[] a) {
  int i = 0;
  while (i < a.length) {
    a[i] = a[i] * 2;
    i++;
  }
}
```

Motivation

Consider the method

```
public void doubleContent(int[] a) {
  int i = 0;
  while (i < a.length) {
    a[i] = a[i] * 2;
    i++;
  }
}
```

We want a logic/calculus allowing to express/prove properties like, e.g.:

Motivation

Consider the method

```
public void doubleContent(int[] a) {
  int i = 0;
  while (i < a.length) {
    a[i] = a[i] * 2;
    i++;
  }
}
```

We want a logic/calculus allowing to express/prove properties like, e.g.:

> *If* `a` $\neq$ `null`
> *then* `doubleContent` terminates normally
> *and* afterwards all elements of `a` are twice the old value

Motivation (contd.)

One such logic is dynamic logic (DL).

Motivation (contd.)

One such logic is dynamic logic (DL).
The above statemet in DL would be:

$$
\begin{array}{l}
\quad\ \ \mathtt{a} \neq \mathtt{null} \\
\quad \land\, \mathtt{a} \neq \mathtt{b} \\
\quad \land\, \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a}[\mathtt{i}] = \mathtt{b}[\mathtt{i}]) \\
\rightarrow \langle \mathtt{doubleContent(a);} \rangle \\
\quad \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a}[\mathtt{i}] = 2 * \mathtt{b}[\mathtt{i}])
\end{array}
$$

Motivation (contd.)

One such logic is dynamic logic (DL).
The above statemet in DL would be:

$$
\begin{array}{rl}
 & \mathtt{a} \neq \mathtt{null} \\
 & \land\, \mathtt{a} \neq \mathtt{b} \\
 & \land\, \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a.length}) \to \mathtt{a[i]} = \mathtt{b[i]}) \\
\to & \langle \mathtt{doubleContent(a);} \rangle \\
 & \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a.length}) \to \mathtt{a[i]} = 2 * \mathtt{b[i]})
\end{array}
$$

- DL combines first-order logic (FOL) with programs

Motivation (contd.)

One such logic is dynamic logic (DL).
The above statemet in DL would be:

$$
\begin{array}{ll}
 & \mathtt{a} \neq \mathtt{null} \\
 & \land\, \mathtt{a} \neq \mathtt{b} \\
 & \land\, \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a}.\mathtt{length}) \rightarrow \mathtt{a}[\mathtt{i}] = \mathtt{b}[\mathtt{i}]) \\
\rightarrow & \langle \mathtt{doubleContent(a);} \rangle \\
 & \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a}.\mathtt{length}) \rightarrow \mathtt{a}[\mathtt{i}] = 2 * \mathtt{b}[\mathtt{i}])
\end{array}
$$

- DL combines first-order logic (FOL) with programs
- Theory of DL extends theory of FOL

Motivation (contd.)

One such logic is dynamic logic (DL).
The above statemet in DL would be:

$$
\begin{aligned}
& \quad \mathtt{a} \neq \mathtt{null} \\
& \land\, \mathtt{a} \neq \mathtt{b} \\
& \land\, \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a[i]} = \mathtt{b[i]}) \\
\rightarrow\ & \langle \mathtt{doubleContent(a);} \rangle \\
& \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \land \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a[i]} = 2 * \mathtt{b[i]})
\end{aligned}
$$

- DL combines first-order logic (FOL) with programs
- Theory of DL extends theory of FOL
- Necessary to look closer at FOL at first

Motivation (contd.)

One such logic is dynamic logic (DL).
The above statemet in DL would be:

$$
\begin{array}{rl}
& \mathtt{a} \neq \mathtt{null} \\
& \wedge\, \mathtt{a} \neq \mathtt{b} \\
& \wedge\, \forall \mathtt{int\ i}; ((0 \leq \mathtt{i} \wedge \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a[i]} = \mathtt{b[i]}) \\
\rightarrow & \langle \mathtt{doubleContent(a);} \rangle \\
& \forall \mathtt{int\ i}; ((0 \leq \mathtt{i} \wedge \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a[i]} = 2 * \mathtt{b[i]})
\end{array}
$$

- DL combines first-order logic (FOL) with programs
- Theory of DL extends theory of FOL
- Necessary to look closer at FOL at first
- Then extend towards DL

Today

introducing dynamic logic for JAVA

- recap first-order logic (FOL)
- semantics of FOL
- dynamic logic = extending FOL with
  - dynamic interpretations
  - programs to describe state change

Repetition: First-Order Logic

Signature

A first-order signature Σ consists of

- a set T_Σ of types
- a set F_Σ of function symbols
- a set P_Σ of predicate symbols

Repetition: First-Order Logic

Signature

A first-order signature Σ consists of

- a set T_Σ of types
- a set F_Σ of function symbols
- a set P_Σ of predicate symbols

Type Declarations

- $\tau\ x$; 'variable x has type τ'
- $p(\tau_1, \ldots, \tau_r)$; 'predicate p has argument types $\tau_1, \ldots, \tau_r$'
- $\tau\ f(\tau_1, \ldots, \tau_r)$; 'function f has argument types $\tau_1, \ldots, \tau_r$ and result type τ'

Part II

First-Order Semantics

First-Order Semantics

From propositional to first-order semantics

- In prop. logic, an interpretation of variables with $\{T, F\}$ sufficed
- In first-order logic we must assign meaning to:
  - function symbols (incl. constants)
  - predicate symbols
- Respect typing: **int** `i`, `List l` must denote different elements

First-Order Semantics

From propositional to first-order semantics

- In prop. logic, an interpretation of variables with $\{T, F\}$ sufficed
- In first-order logic we must assign meaning to:
  - function symbols (incl. constants)
  - predicate symbols
- Respect typing: `int i`, `List l` must denote different elements

What we need (to interpret a first-order formula)

1. A collection of typed universes of elements
2. A mapping from variables to elements
3. For each function symbol, a mapping from arguments to results
4. For each predicate symbol, a set of argument tuples where that predicate holds

First-Order Domains/Universes

1. A collection of typed universes of elements

Definition (Universe/Domain)

A non-empty set $\mathcal{D}$ of elements is a universe or domain.
Each element of $\mathcal{D}$ has a fixed type given by $\delta : \mathcal{D} \to T_\Sigma$

- Notation for the domain elements of type $\tau \in T_\Sigma$:
  $\mathcal{D}^\tau = \{d \in \mathcal{D} \mid \delta(d) = \tau\}$
- Each type $\tau \in T_\Sigma$ must 'contain' at least one domain element:
  $\mathcal{D}^\tau \neq \emptyset$

First-Order States

3. For each function symbol, a mapping from arguments to results
4. For each predicate symbol, a set of argument tuples where that predicate holds

Definition (First-Order State)

Let $\mathcal{D}$ be a domain with typing function δ.

For each f be declared as $\tau\ f(\tau_1, \ldots, \tau_r)$;

and each p be declared as $p(\tau_1, \ldots, \tau_r)$;

First-Order States

3. For each function symbol, a mapping from arguments to results
4. For each predicate symbol, a set of argument tuples where that predicate holds

Definition (First-Order State)

Let $\mathcal{D}$ be a domain with typing function δ.

For each f be declared as $\tau\ f(\tau_1, \ldots, \tau_r)$;

and each p be declared as $p(\tau_1, \ldots, \tau_r)$;

$\mathcal{I}(f)$ is a mapping $\mathcal{I}(f) : \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r} \to \mathcal{D}^{\tau}$

First-Order States

3. For each function symbol, a mapping from arguments to results
4. For each predicate symbol, a set of argument tuples where that predicate holds

Definition (First-Order State)

Let $\mathcal{D}$ be a domain with typing function δ.

For each f be declared as $\tau\ f(\tau_1, \ldots, \tau_r)$;

and each p be declared as $p(\tau_1, \ldots, \tau_r)$;

$\mathcal{I}(f)$ is a mapping $\mathcal{I}(f) : \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r} \to \mathcal{D}^{\tau}$

$\mathcal{I}(p)$ is a set $\mathcal{I}(p) \subseteq \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r}$

First-Order States

3. For each function symbol, a mapping from arguments to results
4. For each predicate symbol, a set of argument tuples where that predicate holds

Definition (First-Order State)

Let $\mathcal{D}$ be a domain with typing function δ.

For each f be declared as $\tau\ f(\tau_1, \ldots, \tau_r)$;

and each p be declared as $p(\tau_1, \ldots, \tau_r)$;

$\mathcal{I}(f)$ is a mapping $\mathcal{I}(f) : \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r} \to \mathcal{D}^{\tau}$

$\mathcal{I}(p)$ is a set $\mathcal{I}(p) \subseteq \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r}$

Then $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ is a first-order state

First-Order States Cont'd

Example

Signature: $\textbf{int}$ `i`; $\textbf{int}$ `j`; $\textbf{int}$ `f(`$\textbf{int}$`)`; `Object obj`; `<(`$\textbf{int}$`,`$\textbf{int}$`)`;
$\mathcal{D} = \{17, 2, o\}$

First-Order States Cont'd

Example

Signature: `int i; int j; int f(int); Object obj; <(int,int);`
$\mathcal{D} = \{17,\ 2,\ o\}$
The following $\mathcal{I}$ is a possible interpretation:

$$\mathcal{I}(i) = 17$$
$$\mathcal{I}(j) = 17$$
$$\mathcal{I}(\texttt{obj}) = o$$

$\mathcal{D}^{\mathbf{int}}$	$\mathcal{I}(f)$
2	2
17	2

$\mathcal{D}^{\mathbf{int}} \times \mathcal{D}^{\mathbf{int}}$	in $\mathcal{I}(<)$?
$(2, 2)$	*no*
$(2, 17)$	*yes*
$(17, 2)$	*no*
$(17, 17)$	*no*

One of uncountably many possible first-order states!

Semantics of Reserved Signature Symbols

Definition

Reserved predicate symbol for equality: $=$

Semantics of Reserved Signature Symbols

Definition

Reserved predicate symbol for equality: $=$

Interpretation is fixed as $\mathcal{I}(=) =$

Semantics of Reserved Signature Symbols

Definition

Reserved predicate symbol for equality: $=$

Interpretation is fixed as $\mathcal{I}(=) = \{(d, d) \mid d \in \mathcal{D}\}$

Semantics of Reserved Signature Symbols

Definition

Reserved predicate symbol for equality: $=$

Interpretation is fixed as $\mathcal{I}(=) = \{(d, d) \mid d \in \mathcal{D}\}$

Exercise: write down all elements of the set $\mathcal{I}(=)$ for example domain

Signature Symbols vs. Domain Elements

- Domain elements different from the terms representing them
- First-order formulas and terms have no access to domain

Example

Signature: `Object obj1, obj2;`
Domain: $\mathcal{D} = \{o\}$

Signature Symbols vs. Domain Elements

- Domain elements different from the terms representing them
- First-order formulas and terms have no access to domain

Example

Signature: `Object obj1, obj2;`
Domain: $\mathcal{D} = \{o\}$

In this state, necessarily $\mathcal{I}(\texttt{obj1}) = \mathcal{I}(\texttt{obj2}) = o$

Variable Assignments

2. A mapping from variables to domain elements

Definition (Variable Assignment)

A variable assignment β maps variables to domain elements
It respects the variable type, i.e., if x has type τ then $\beta(x) \in \mathcal{D}^\tau$

Semantic Evaluation of Terms

Given a first-order state $\mathcal{S}$ and a variable assignment β
it is possible to evaluate first-order terms under $\mathcal{S}$ and β

Definition (Valuation of Terms)

$val_{\mathcal{S},\beta}$: Term $\rightarrow \mathcal{D}$ such that $val_{\mathcal{S},\beta}(t) \in \mathcal{D}^{\tau}$ for $t \in \text{Term}_{\tau}$:

- $val_{\mathcal{S},\beta}(x) =$

Semantic Evaluation of Terms

Given a first-order state $\mathcal{S}$ and a variable assignment β
it is possible to evaluate first-order terms under $\mathcal{S}$ and β

Definition (Valuation of Terms)

$val_{\mathcal{S},\beta}$: Term $\rightarrow \mathcal{D}$ such that $val_{\mathcal{S},\beta}(t) \in \mathcal{D}^{\tau}$ for $t \in \text{Term}_{\tau}$:

- $val_{\mathcal{S},\beta}(x) = \beta(x)$

Semantic Evaluation of Terms

Given a first-order state $\mathcal{S}$ and a variable assignment β
it is possible to evaluate first-order terms under $\mathcal{S}$ and β

Definition (Valuation of Terms)

$val_{\mathcal{S},\beta}$: Term $\rightarrow \mathcal{D}$ such that $val_{\mathcal{S},\beta}(t) \in \mathcal{D}^{\tau}$ for $t \in \mathrm{Term}_{\tau}$:

- $val_{\mathcal{S},\beta}(x) = \beta(x)$
- $val_{\mathcal{S},\beta}(f(t_1, \ldots, t_r)) =$

Semantic Evaluation of Terms

Given a first-order state $\mathcal{S}$ and a variable assignment β
it is possible to evaluate first-order terms under $\mathcal{S}$ and β

Definition (Valuation of Terms)

$val_{\mathcal{S},\beta}$: Term $\rightarrow \mathcal{D}$ such that $val_{\mathcal{S},\beta}(t) \in \mathcal{D}^{\tau}$ for $t \in \text{Term}_{\tau}$:

- $val_{\mathcal{S},\beta}(x) = \beta(x)$
- $val_{\mathcal{S},\beta}(f(t_1, \ldots, t_r)) = \mathcal{I}(f)(val_{\mathcal{S},\beta}(t_1), \ldots, val_{\mathcal{S},\beta}(t_r))$

Semantic Evaluation of Terms Cont'd

Example

Signature: **int** `i`; **int** `j`; **int** `f(`**int**`)`;

$\mathcal{D} = \{17, 2, o\}$ Variables: `Object obj`; **int** `x`;

$\mathcal{I}(\mathtt{i}) = 17$
$\mathcal{I}(\mathtt{j}) = 17$

$\mathcal{D}^{\mathbf{int}}$	$\mathcal{I}(\mathtt{f})$
2	17
17	2

Var	β
`obj`	o
`x`	17

- $val_{\mathcal{S},\beta}(\mathtt{f(f(i))})$?
- $val_{\mathcal{S},\beta}(\mathtt{f(f(x))})$?
- $val_{\mathcal{S},\beta}(\mathtt{obj})$?

Preparing for Semantic Evaluation of Formulas

Definition (Modified Variable Assignment)

Let y be variable of type τ, β variable assignment, $d \in \mathcal{D}^\tau$:

$$\beta_y^d(x) := \begin{cases} \beta(x) & x \neq y \\ d & x = y \end{cases}$$

Semantic Evaluation of Formulas

Definition (Valuation of Formulas)

$val_{\mathcal{S},\beta}(\phi)$ for $\phi \in For$

- $val_{\mathcal{S},\beta}(p(t_1, \ldots, t_r)) = T \quad$ iff $\quad (val_{\mathcal{S},\beta}(t_1), \ldots, val_{\mathcal{S},\beta}(t_r)) \in \mathcal{I}(p)$

Semantic Evaluation of Formulas

Definition (Valuation of Formulas)

$val_{\mathcal{S},\beta}(\phi)$ for $\phi \in For$

- $val_{\mathcal{S},\beta}(p(t_1, \ldots, t_r)) = T$ iff $(val_{\mathcal{S},\beta}(t_1), \ldots, val_{\mathcal{S},\beta}(t_r)) \in \mathcal{I}(p)$
- $val_{\mathcal{S},\beta}(\phi \land \psi) = T$ iff $val_{\mathcal{S},\beta}(\phi) = T$ and $val_{\mathcal{S},\beta}(\psi) = T$
- ... as in propositional logic

Semantic Evaluation of Formulas

Definition (Valuation of Formulas)

$val_{\mathcal{S},\beta}(\phi)$ for $\phi \in For$

- $val_{\mathcal{S},\beta}(p(t_1, \ldots, t_r)) = T$ iff $(val_{\mathcal{S},\beta}(t_1), \ldots, val_{\mathcal{S},\beta}(t_r)) \in \mathcal{I}(p)$
- $val_{\mathcal{S},\beta}(\phi \wedge \psi) = T$ iff $val_{\mathcal{S},\beta}(\phi) = T$ and $val_{\mathcal{S},\beta}(\psi) = T$
- ... as in propositional logic
- $val_{\mathcal{S},\beta}(\forall\, \tau\ x;\ \phi) = T$ iff $val_{\mathcal{S},\beta_x^d}(\phi) = T$ for all $d \in \mathcal{D}^\tau$

Semantic Evaluation of Formulas

Definition (Valuation of Formulas)

$val_{\mathcal{S},\beta}(\phi)$ for $\phi \in For$

- $val_{\mathcal{S},\beta}(p(t_1, \ldots, t_r)) = T \quad$ iff $\quad (val_{\mathcal{S},\beta}(t_1), \ldots, val_{\mathcal{S},\beta}(t_r)) \in \mathcal{I}(p)$
- $val_{\mathcal{S},\beta}(\phi \wedge \psi) = T \quad$ iff $\quad val_{\mathcal{S},\beta}(\phi) = T$ and $val_{\mathcal{S},\beta}(\psi) = T$
- ... as in propositional logic
- $val_{\mathcal{S},\beta}(\forall\, \tau\; x;\, \phi) = T \quad$ iff $\quad val_{\mathcal{S},\beta_x^d}(\phi) = T$ for all $d \in \mathcal{D}^\tau$
- $val_{\mathcal{S},\beta}(\exists\, \tau\; x;\, \phi) = T \quad$ iff $\quad val_{\mathcal{S},\beta_x^d}(\phi) = T$ for at least one $d \in \mathcal{D}^\tau$

Semantic Evaluation of Formulas Cont'd

Example

Signature: `int j; int f(int); Object obj; <(int,int);`

$\mathcal{D} = \{17,\ 2,\ o\}$, $\mathcal{D}^{\mathtt{int}} = \{17,\ 2\}$, $\mathcal{D}^{\mathtt{Object}} = \{o\}$

$$\mathcal{I}(j) = 17$$
$$\mathcal{I}(\mathtt{obj}) = o$$

$\mathcal{D}^{\mathtt{int}}$	$\mathcal{I}(f)$
2	2
17	2

$\mathcal{D}^{\mathtt{int}} \times \mathcal{D}^{\mathtt{int}}$	in $\mathcal{I}(<)$?
$(2, 2)$	F
$(2, 17)$	T
$(17, 2)$	F
$(17, 17)$	F

Semantic Evaluation of Formulas Cont'd

Example

Signature: `int j; int f(int); Object obj; <(int,int);`

$\mathcal{D} = \{17,\ 2,\ o\}$, $\mathcal{D}^{\texttt{int}} = \{17,\ 2\}$, $\mathcal{D}^{\texttt{Object}} = \{o\}$

$\mathcal{I}(j) = 17$

$\mathcal{I}(\texttt{obj}) = o$

$\mathcal{D}^{\texttt{int}}$	$\mathcal{I}(f)$
2	2
17	2

$\mathcal{D}^{\texttt{int}} \times \mathcal{D}^{\texttt{int}}$	in $\mathcal{I}(<)$?
$(2, 2)$	F
$(2, 17)$	T
$(17, 2)$	F
$(17, 17)$	F

- $val_{\mathcal{S},\beta}(f(j) < j)$?
- $val_{\mathcal{S},\beta}(\exists\, \texttt{int}\ x;\ f(x) = x)$?
- $val_{\mathcal{S},\beta}(\forall\, \texttt{Object}\ o1;\ \forall\, \texttt{Object}\ o2;\ o1 = o2)$?

Semantic Notions

Definition (Satisfiability, Truth, Validity)

$val_{\mathcal{S},\beta}(\phi) = T$			($\mathcal{S}, \beta$ **satisfies** ϕ)
$\mathcal{S} \models \phi$	iff	for all β : $val_{\mathcal{S},\beta}(\phi) = T$	(ϕ is **true** in $\mathcal{S}$)
$\models \phi$	iff	for all $\mathcal{S}$: $\mathcal{S} \models \phi$	(ϕ is **valid**)

Semantic Notions

Definition (Satisfiability, Truth, Validity)

$val_{\mathcal{S},\beta}(\phi) = T$			($\mathcal{S}, \beta$ **satisfies** ϕ)
$\mathcal{S} \models \phi$	iff	for all β : $val_{\mathcal{S},\beta}(\phi) = T$	(ϕ is **true** in $\mathcal{S}$)
$\models \phi$	iff	for all $\mathcal{S}$: $\mathcal{S} \models \phi$	(ϕ is **valid**)

Example

- $f(j) < j$ is true in $\mathcal{S}$
- $\exists\, \mathbf{int}\ x;\ i = x$ is valid
- $\exists\, \mathbf{int}\ x;\ \neg(x = x)$ is not satisfiable

Part III

Towards Dynamic Logic

Type Hierarchy

First, we *refine the type system* of FOL:

Definition (Type Hierarchy)

- T_Σ is set of types
- Given subtype relation '$\sqsubseteq$', with top element '*any*'
- $\tau \sqsubseteq \mathit{any}$ for all $\tau \in T_\Sigma$

Type Hierarchy

First, we *refine the type system* of FOL:

Definition (Type Hierarchy)

- T_Σ is set of types
- Given subtype relation '$\sqsubseteq$', with top element '*any*'
- $\tau \sqsubseteq any$ for all $\tau \in T_\Sigma$

Example (A Minimal Type Hierarchy)

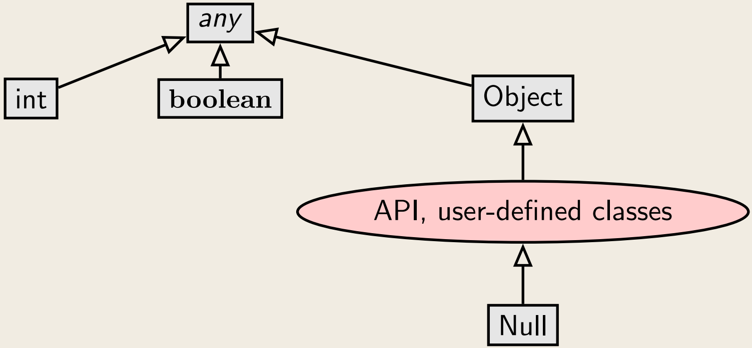
$\mathcal{T} = \{any\}$
All signature symbols have same type *any*.

Example (Type Hierarchy for Java)

(see next slide)

Modelling Java in FOL: Fixing a Type Hierarchy

Signature based on Java's type hierarchy (simplified)



Each class in API and target program is a type, with appropriate subtyping.

Modelling Classes and Fields in FOL

Modeling instance fields

Person
`int age` `int id`
`int setAge(int newAge)` `int getId()`

- domain of all Person objects: $\mathcal{D}^{\text{Person}}$

Modelling Classes and Fields in FOL

Modeling instance fields

Person
int age int id
int setAge(int newAge) int getId()

- domain of all Person objects: $\mathcal{D}^{\mathsf{Person}}$
- each $o \in \mathcal{D}^{\mathsf{Person}}$ has associated age value

Modelling Classes and Fields in FOL

Modeling instance fields

Person
`int age` `int id`
`int setAge(int newAge)` `int getId()`

- domain of all Person objects: $\mathcal{D}^{\mathsf{Person}}$
- each $o \in \mathcal{D}^{\mathsf{Person}}$ has associated age value
- $\mathcal{I}(\mathsf{age})$ is mapping from $\mathcal{D}^{\mathsf{Person}}$ to $\mathcal{D}^{\mathbf{int}}$

Modelling Classes and Fields in FOL

Modeling instance fields

```
Person
------------------------------
int age
int id
------------------------------
int setAge(int newAge)
int getId()
```

- domain of all Person objects: $\mathcal{D}^{\mathsf{Person}}$
- each $o \in \mathcal{D}^{\mathsf{Person}}$ has associated age value
- $\mathcal{I}$(age) is mapping from $\mathcal{D}^{\mathsf{Person}}$ to $\mathcal{D}^{\mathbf{int}}$
- for each class C with field τ a:

  FSym declares function τ a(C);

Modelling Classes and Fields in FOL

Modeling instance fields

Person
`int age` `int id`
`int setAge(int newAge)` `int getId()`

- domain of all Person objects: $\mathcal{D}^{\text{Person}}$
- each $o \in \mathcal{D}^{\text{Person}}$ has associated age value
- $\mathcal{I}(\text{age})$ is mapping from $\mathcal{D}^{\text{Person}}$ to $\mathcal{D}^{\texttt{int}}$
- for each class C with field τ a:
  FSym declares function τ a(C);

Field Access

Signature FSym: $\texttt{int}$ age(Person); Person p;

Java/JML expression `p.age >= 0`

Typed FOL age(p)>=0

KeY postfix notation for FOL `p.age >= 0`

Navigation expressions in KeY look exactly as in JAVA/JML

Dynamic View

Only static properties expressable in typed FOL, e.g.,

- Values of fields in a certain range

Dynamic View

Only static properties expressable in typed FOL, e.g.,

- Values of fields in a certain range
- Property (invariant) of a subclass implies property of a superclass

Dynamic View

Only static properties expressable in typed FOL, e.g.,

- Values of fields in a certain range
- Property (invariant) of a subclass implies property of a superclass
- ...

Dynamic View

Only static properties expressable in typed FOL, e.g.,

- Values of fields in a certain range
- Property (invariant) of a subclass implies property of a superclass
- ...

Considers only one state at a time.

Dynamic View

Only static properties expressable in typed FOL, e.g.,

- Values of fields in a certain range
- Property (invariant) of a subclass implies property of a superclass
- ...

Considers only one state at a time.

Goal: Express functional properties of a program, e.g.

> If method `setAge` is called on an object o of type `Person`
> and the method argument `newAge` is positive
> then afterwards field `age` has same value as `newAge`.

Observation

Need a logic that allows us to

- relate different program states, i.e., before and after execution, within one formula

Observation

Need a logic that allows us to

- relate different program states, i.e., before and after execution, within one formula
- program variables/fields represented by constant/function symbols that depend on program state

Observation

Need a logic that allows us to

- relate different program states, i.e., before and after execution, within one formula
- program variables/fields represented by constant/function symbols that depend on program state

Dynamic Logic meets the above requirements.

Dynamic Logic

(JAVA) Dynamic Logic

Typed FOL

- $+$ programs $\mathtt{p}$

Dynamic Logic

(Java) Dynamic Logic

Typed FOL

- $+$ programs $\mathtt{p}$
- $+$ modalities $\langle \mathtt{p} \rangle \phi$, $[\mathtt{p}]\phi$ ($\mathtt{p}$ program, ϕ DL formula)

Dynamic Logic

(JAVA) Dynamic Logic

Typed FOL

- $+$ programs $\mathtt{p}$
- $+$ modalities $\langle \mathtt{p} \rangle \phi$, $[\mathtt{p}]\phi$ ($\mathtt{p}$ program, ϕ DL formula)
- $+$ ... (later)

Dynamic Logic

(JAVA) Dynamic Logic

Typed FOL

- $+$ programs $\texttt{p}$
- $+$ modalities $\langle\texttt{p}\rangle\phi$, $[\texttt{p}]\phi$ ($\texttt{p}$ program, ϕ DL formula)
- $+$ ... (later)

An Example

$$\texttt{i} > 5 \;\rightarrow\; [\texttt{i = i + 10;}]\texttt{i} > 15$$

Meaning?

Dynamic Logic

(JAVA) Dynamic Logic

Typed FOL

- $+$ programs $\mathtt{p}$
- $+$ modalities $\langle \mathtt{p} \rangle \phi$, $[\mathtt{p}]\phi$ ($\mathtt{p}$ program, ϕ DL formula)
- $+$ ... (later)

An Example

$$\mathtt{i} > 5 \;\rightarrow\; [\mathtt{i = i + 10;}]\mathtt{i} > 15$$

Meaning?

If program variable `i` is greater than 5, then after executing `i = i + 10;`, `i` is greater than 15.

Program Variables

Dynamic Logic = Typed FOL + ...

$$\mathtt{i} > 5 \;\rightarrow\; [\mathtt{i\ =\ i\ +\ 10;}]\mathtt{i} > 15$$

Program variable `i` refers to different values before and after execution of a program.

- Program variables like `i` are state-dependent constant symbols.
- Value of state dependent symbols changeable by program.

Program Variables

Dynamic Logic = Typed FOL + ...

$$\mathtt{i} > 5 \;\rightarrow\; [\mathtt{i\ =\ i\ +\ 10;}]\mathtt{i} > 15$$

Program variable `i` refers to different values before and after execution of a program.

- Program variables like `i` are state-dependent constant symbols.
- Value of state dependent symbols changeable by program.

Three words one meaning: flexible, state-dependent, non-rigid

Rigid versus Flexible Symbols

Signature of dynamic logic defined as in FOL, but:
In addition there are flexible symbols

Rigid versus Flexible

- Rigid symbols, same interpretation in all program states
  - First-order variables (aka logical variables)
  - Built-in functions and predicates such as `0,1,...,+,*,...,<,...`
- Flexible (or non-rigid) symbols, interpretation depends on state

  Capture side effects on state during program execution
  - Functions modeling program variables and fields are flexible

Any term containing at least one flexible symbol is also flexible

Signature of Dynamic Logic

Definition (Dynamic Logic Signature)

$\Sigma = (\mathsf{PSym}_r, \mathsf{FSym}_r, \mathsf{FSym}_f, \alpha), \quad \mathsf{FSym}_r \cap \mathsf{FSym}_f = \emptyset$

Rigid Predicate Symbols	$\mathsf{PSym}_r = \{>, >=, \ldots\}$
Rigid Function Symbols	$\mathsf{FSym}_r = \{+, -, *, 0, 1, \ldots\}$
Flexible Function Symbols	$\mathsf{FSym}_f = \{i, j, k, \ldots\}$

Standard typing: **boolean** `TRUE`; `<(`**int**`,`**int**`)`; etc.

Flexible constant/function symbols FSym_f used to model

- program variables (flexible constants) and
- fields (flexible unary functions)

Dynamic Logic Signature - KeY input file

```
\sorts {
  // only additional sorts (predefined:  int/boolean/any)
}
\functions {
 // only additional rigid functions
 // (arithmetic functions like +,- etc.  predefined)
}
\predicates {  /* same as for functions */  }
```

Empty sections can be left out.

Dynamic Logic Signature - KeY input file

```
\sorts {
  // only additional sorts (predefined:  int/boolean/any)
}
\functions {
 // only additional rigid functions
 // (arithmetic functions like +,- etc.  predefined)
}
\predicates {  /* same as for functions */  }

\programVariables { // flexible functions
   int i, j;
   boolean b;
}
```

Empty sections can be left out.

Variables

Logical Variables

Typed logical variables (rigid), declared locally in quantifiers as `T x;`

Program Variables

Flexible constants **int** `i;` **boolean** `p;` used as program variables

Dynamic Logic Programs

Dynamic Logic = Typed FOL + programs ...

Programs here: any legal sequence of JAVA statements.

Dynamic Logic Programs

Dynamic Logic = Typed FOL + programs ...

Programs here: any legal sequence of JAVA statements.

Example

Signature for FSym_f: int r; int i; int n;

Signature for FSym_r: int 0; int +(int,int); int -(int,int);

Signature for PSym_r: <(int,int);

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

Dynamic Logic Programs

Dynamic Logic = Typed FOL + programs ...

Programs here: any legal sequence of JAVA statements.

Example

Signature for $FSym_f$: `int r; int i; int n;`

Signature for $FSym_r$: `int 0; int +(int,int); int -(int,int);`

Signature for $PSym_r$: `<(int,int);`

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

Which value does the program compute in `r`?

Relating Program States: Modalities

DL extends FOL with two additional (mix-fix) operators:

- $\langle \mathtt{p} \rangle \phi$ (diamond)
- $[p]\phi$ (box)

with $\mathtt{p}$ a program, ϕ another DL formula

Relating Program States: Modalities

DL extends FOL with two additional (mix-fix) operators:

- $\langle \mathtt{p} \rangle \phi$ (diamond)
- $[p]\phi$ (box)

with $\mathtt{p}$ a program, ϕ another DL formula

Intuitive Meaning

- $\langle \mathtt{p} \rangle \phi$: $\mathtt{p}$ terminates and formula ϕ holds in final state (total correctness)

Relating Program States: Modalities

DL extends FOL with two additional (mix-fix) operators:

- $\langle \mathrm{p} \rangle \phi$ (diamond)
- $[p]\phi$ (box)

with p a program, ϕ another DL formula

Intuitive Meaning

- $\langle \mathrm{p} \rangle \phi$: p terminates and formula ϕ holds in final state (total correctness)
- $[p]\phi$: If p terminates then formula ϕ holds in final state (partial correctness)

Relating Program States: Modalities

DL extends FOL with two additional (mix-fix) operators:

- $\langle \mathrm{p} \rangle \phi$ (diamond)
- $[p]\phi$ (box)

with p a program, ϕ another DL formula

Intuitive Meaning

- $\langle \mathrm{p} \rangle \phi$: p terminates and formula ϕ holds in final state (total correctness)
- $[p]\phi$: If p terminates then formula ϕ holds in final state (partial correctness)

Attention: JAVA programs are deterministic, i.e., if a JAVA program terminates then exactly one state is reached from a given initial state.

Dynamic Logic - Examples

Let `i`, `j`, `old_i`, `old_j` denote program variables.
Give the meaning in natural language:

1. $\mathtt{i} = \mathtt{old\_i} \rightarrow \langle\mathtt{i\ =\ i\ +\ 1;}\rangle \mathtt{i} > \mathtt{old\_i}$

Dynamic Logic - Examples

Let `i`, `j`, `old_i`, `old_j` denote program variables.
Give the meaning in natural language:

1. $\texttt{i} = \texttt{old\_i} \rightarrow \langle\texttt{i = i + 1;}\rangle \texttt{i} > \texttt{old\_i}$

   If `i = i + 1;` is executed in a state where `i` and `old_i` have the same value, then the program terminates and in its final state the value of `i` is greater than the value of `old_i` .

Dynamic Logic - Examples

Let i, j, old_i, old_j denote program variables.
Give the meaning in natural language:

1. $\mathtt{i} = \mathtt{old\_i} \rightarrow \langle \mathtt{i\ =\ i\ +\ 1;} \rangle \mathtt{i} > \mathtt{old\_i}$

   If `i = i + 1;` is executed in a state where `i` and `old_i` have the same value, then the program terminates and in its final state the value of `i` is greater than the value of `old_i` .
2. $\mathtt{i} = \mathtt{old\_i} \rightarrow [\mathtt{while(true)}\{\mathtt{i\ =\ old\_i\ -\ 1;}\}] \mathtt{i} > \mathtt{old\_i}$

Dynamic Logic - Examples

Let `i`, `j`, `old_i`, `old_j` denote program variables.
Give the meaning in natural language:

1. $\mathtt{i} = \mathtt{old\_i} \rightarrow \langle \mathtt{i = i + 1;} \rangle \mathtt{i} > \mathtt{old\_i}$

   If `i = i + 1;` is executed in a state where `i` and `old_i` have the same value, then the program terminates and in its final state the value of `i` is greater than the value of `old_i` .

2. $\mathtt{i} = \mathtt{old\_i} \rightarrow [\mathtt{while(true)}\{\mathtt{i = old\_i - 1;}\}]\mathtt{i} > \mathtt{old\_i}$

   If the program is executed in a state where `i` and `old_i` have the same value and if the program terminates then in its final state the value of `i` is greater than the value of `old_i`.

Dynamic Logic - Examples

Let `i`, `j`, `old_i`, `old_j` denote program variables.
Give the meaning in natural language:

1. $\mathtt{i} = \mathtt{old\_i} \rightarrow \langle \mathtt{i\ =\ i\ +\ 1;} \rangle \mathtt{i} > \mathtt{old\_i}$

   If `i = i + 1;` is executed in a state where `i` and `old_i` have the same value, then the program terminates and in its final state the value of `i` is greater than the value of `old_i` .

2. $\mathtt{i} = \mathtt{old\_i} \rightarrow [\mathtt{while(true)}\{\mathtt{i\ =\ old\_i\ -\ 1;}\}]\mathtt{i} > \mathtt{old\_i}$

   If the program is executed in a state where `i` and `old_i` have the same value and if the program terminates then in its final state the value of `i` is greater than the value of `old_i`.

3. $\forall\, x.\ (\ \langle \mathrm{p} \rangle\ \mathtt{i} = x\ \leftrightarrow\ \langle \mathrm{q} \rangle\ \mathtt{i} = x\ )$

Dynamic Logic - Examples

Let `i`, `j`, `old_i`, `old_j` denote program variables.
Give the meaning in natural language:

1. $\mathtt{i} = \mathtt{old\_i} \rightarrow \langle \mathtt{i = i + 1;} \rangle \mathtt{i} > \mathtt{old\_i}$

   If `i = i + 1;` is executed in a state where `i` and `old_i` have the same value, then the program terminates and in its final state the value of `i` is greater than the value of `old_i` .

2. $\mathtt{i} = \mathtt{old\_i} \rightarrow [\mathtt{while(true)}\{\mathtt{i = old\_i - 1;}\}]\mathtt{i} > \mathtt{old\_i}$

   If the program is executed in a state where `i` and `old_i` have the same value and if the program terminates then in its final state the value of `i` is greater than the value of `old_i`.

3. $\forall\, x.\ (\ \langle \mathtt{p} \rangle\ \mathtt{i} = x\ \leftrightarrow\ \langle \mathtt{q} \rangle\ \mathtt{i} = x\ )$

   `p` and `q` are equivalent concerning termination and the final value of `i`.

Dynamic Logic - KeY input file

KeY

```
\programVariables {  // Declares global program variables
       int i, j;
       int old_i, old_j;
}
```

KeY

Dynamic Logic - KeY input file

KeY

```
\programVariables {  // Declares global program variables
        int i, j;
        int old_i, old_j;
}


\problem {  // The problem to verify is stated here
        i = old_i -> \<{   i = i + 1;  }\> i > old_i
}
```

KeY

Dynamic Logic - KeY input file

KeY

```
\programVariables {  // Declares global program variables
       int i, j;
       int old_i, old_j;
}


\problem {  // The problem to verify is stated here
       i = old_i -> \<{   i = i + 1;  }\> i > old_i
}
```

KeY

Visibility: Program variables declared

- global can be accessed anywhere in the formula.
- inside modality like $pre \rightarrow \langle$`int j; p`$\rangle post$ only visible in `p`

Dynamic Logic Formulas

Definition (Dynamic Logic Formulas (DL Formulas))

- Each FOL formula is a DL formula
- If p is a program and ϕ a DL formula then $\left\{ \begin{matrix} \langle \mathrm{p} \rangle \phi \\ [\mathrm{p}]\phi \end{matrix} \right\}$ is a DL formula
- DL formulas closed under FOL quantifiers and connectives

Dynamic Logic Formulas

Definition (Dynamic Logic Formulas (DL Formulas))

- Each FOL formula is a DL formula
- If p is a program and ϕ a DL formula then $\begin{Bmatrix} \langle \mathrm{p} \rangle \phi \\ [\mathrm{p}] \phi \end{Bmatrix}$ is a DL formula
- DL formulas closed under FOL quantifiers and connectives

- Program variables are flexible constants: never bound in quantifiers
- Program variables need not be declared or initialized in program
- Programs contain no logical variables
- Modalities can be arbitrarily nested

Dynamic Logic Formulas Cont'd

Example (Well-formed? If yes, under which signature?)

- $\forall\,\mathbf{int}\ y; ((\langle \texttt{x = 2;}\rangle \texttt{x} = y) \leftrightarrow (\langle \texttt{x = 1; x++;}\rangle \texttt{x} = y))$

Dynamic Logic Formulas Cont'd

Example (Well-formed? If yes, under which signature?)

- $\forall\,\mathbf{int}\ y; \ ((\langle \texttt{x = 2;}\rangle \texttt{x} = y) \leftrightarrow (\langle \texttt{x = 1; x++;}\rangle \texttt{x} = y))$

  Well-formed if FSym_f contains $\mathbf{int}\ \texttt{x;}$

Dynamic Logic Formulas Cont'd

Example (Well-formed? If yes, under which signature?)

- $\forall\,\mathbf{int}\ y;\ ((\langle \texttt{x = 2;} \rangle \texttt{x} = y) \leftrightarrow (\langle \texttt{x = 1; x++;} \rangle \texttt{x} = y))$

  Well-formed if FSym_f contains $\mathbf{int}\ \texttt{x;}$
- $\exists\,\mathbf{int}\ x;\ [x = 1;](x = 1)$

Dynamic Logic Formulas Cont'd

Example (Well-formed? If yes, under which signature?)

- $\forall\, \mathbf{int}\ y; ((\langle \texttt{x = 2;} \rangle \texttt{x} = y) \leftrightarrow (\langle \texttt{x = 1; x++;} \rangle \texttt{x} = y))$

  Well-formed if FSym_f contains $\mathbf{int}\ \texttt{x;}$
- $\exists\, \mathbf{int}\ x; [x = 1;](x = 1)$

  Not well-formed, because logical variable occurs in program

Dynamic Logic Formulas Cont'd

Example (Well-formed? If yes, under which signature?)

- $\forall\,\mathbf{int}\ y;\ ((\langle \texttt{x = 2;} \rangle \texttt{x} = y) \leftrightarrow (\langle \texttt{x = 1; x++;} \rangle \texttt{x} = y))$

  Well-formed if FSym_f contains $\mathbf{int}\ \texttt{x;}$
- $\exists\,\mathbf{int}\ x;\ [x = 1;](x = 1)$

  Not well-formed, because logical variable occurs in program
- $\langle \texttt{x = 1;} \rangle ([\mathbf{while}\ (\mathbf{true})\ \texttt{\{\}}]\mathbf{false})$

Dynamic Logic Formulas Cont'd

Example (Well-formed? If yes, under which signature?)

- $\forall\, \mathbf{int}\ y; (((\langle \texttt{x = 2;} \rangle \mathtt{x} = y) \leftrightarrow (\langle \texttt{x = 1; x++;} \rangle \mathtt{x} = y))$

  Well-formed if FSym_f contains $\mathbf{int}\ \texttt{x;}$
- $\exists\, \mathbf{int}\ x; [x = 1;](x = 1)$

  Not well-formed, because logical variable occurs in program
- $\langle \texttt{x = 1;} \rangle ([\mathbf{while}\ (\mathbf{true})\ \{\}]\mathbf{false})$

  Well-formed if FSym_f contains $\mathbf{int}\ \texttt{x;}$
  program formulas can be nested

Dynamic Logic Semantics: States

First-order state can be considered as program state

- Interpretation of flexible symbols can vary from state to state (eg, program variables, field values)
- Interpretation of rigid symbols is the same in all states (eg, built-in functions and predicates)

Program states as first-order states

From now, consider program state s as first-order state $(\mathcal{D}, \delta, \mathcal{I})$

- Only interpretation $\mathcal{I}$ of flexible symbols in FSym_f can change
- *States* is set of all states s

Kripke Structure

Definition (Kripke Structure)

Kripke structure or Labelled transition system $K = (States, \rho)$

- State (=first-order model) $s = (\mathcal{D}, \delta, \mathcal{I}) \in States$
- Transition relation $\rho : Program \to (States \rightharpoonup States)$

$$\rho(\mathrm{p})(s1) = s2$$

iff.

program p executed in state $s1$ terminates and its final state is $s2$, otherwise undefined.

- ρ is the semantics of programs $\in Program$
- $\rho(\mathrm{p})(s)$ can be undefined ('$\rightharpoonup$'):
  p may not terminate when started in s
- Our programs are deterministic (unlike PROMELA):
  $\rho(\mathrm{p})$ is a function (at most one value)

Semantic Evaluation of Program Formulas

Definition (Validity Relation for Program Formulas)

- $s \models \langle \mathrm{p} \rangle \phi$ iff $\rho(\mathrm{p})(s)$ is defined and $\rho(\mathrm{p})(s) \models \phi$

  (p terminates and ϕ is true in the final state after execution)

Semantic Evaluation of Program Formulas

Definition (Validity Relation for Program Formulas)

- $s \models \langle \mathrm{p} \rangle \phi$ iff $\rho(\mathrm{p})(s)$ is defined and $\rho(\mathrm{p})(s) \models \phi$

  (p terminates and ϕ is true in the final state after execution)
- $s \models [\mathrm{p}]\phi$ iff $\rho(\mathrm{p})(s) \models \phi$ whenever $\rho(\mathrm{p})(s)$ is defined

  (If p terminates then ϕ is true in the final state after execution)

Semantic Evaluation of Program Formulas

Definition (Validity Relation for Program Formulas)

- $s \models \langle \mathrm{p} \rangle \phi$ iff $\rho(\mathrm{p})(s)$ is defined and $\rho(\mathrm{p})(s) \models \phi$

  (p terminates and ϕ is true in the final state after execution)
- $s \models [\mathrm{p}]\phi$ iff $\rho(\mathrm{p})(s) \models \phi$ whenever $\rho(\mathrm{p})(s)$ is defined

  (If p terminates then ϕ is true in the final state after execution)

- Duality: $\langle \mathrm{p} \rangle \phi$ iff $\neg[\mathrm{p}]\neg\phi$
  Exercise: justify this with help of semantic definitions
- Implication: if $\langle \mathrm{p} \rangle \phi$ then $[\mathrm{p}]\phi$
  Total correctness implies partial correctness
  - converse is false
  - holds only for deterministic programs

More Examples

valid?
meaning?

Example

$\forall\, \tau\ y;\ (((\langle \mathtt{p} \rangle \mathtt{x} = y)\ \leftrightarrow\ (\langle \mathtt{q} \rangle \mathtt{x} = y))$

More Examples

valid?
meaning?

Example

$\forall\, \tau\ y; ((\langle \mathtt{p} \rangle \mathtt{x} = y) \leftrightarrow (\langle \mathtt{q} \rangle \mathtt{x} = y))$

Not valid in general

Programs $\mathtt{p}$ behave $\mathtt{q}$ equivalently on variable τ $\mathtt{x}$

More Examples

valid?
meaning?

Example

$\forall\, \tau\ y; ((\langle \mathrm{p} \rangle \mathrm{x} = y) \ \leftrightarrow\ (\langle \mathrm{q} \rangle \mathrm{x} = y))$

Not valid in general

Programs p behave q equivalently on variable τ x

Example

$\exists\, \tau\ y; (\mathrm{x} = y \ \rightarrow\ \langle \mathrm{p} \rangle \mathbf{true})$

More Examples

valid?
meaning?

Example

$\forall\, \tau\ y; ((\langle \mathrm{p} \rangle \mathrm{x} = y) \leftrightarrow (\langle \mathrm{q} \rangle \mathrm{x} = y))$

Not valid in general

Programs p behave q equivalently on variable τ x

Example

$\exists\, \tau\ y; (\mathrm{x} = y \rightarrow \langle \mathrm{p} \rangle \mathbf{true})$

Not valid in general

Program p terminates if initial value of x is suitably chosen

Semantics of Programs

In labelled transition system $K = (\mathit{States}, \rho)$:
$\rho : \mathit{Program} \rightarrow (\mathit{States} \rightharpoonup \mathit{States})$ is semantics of programs $\mathrm{p} \in \mathit{Program}$

ρ defined recursively on programs

Semantics of Programs

In labelled transition system $K = (States, \rho)$:
$\rho : Program \to (States \rightharpoonup States)$ is semantics of programs $\mathtt{p} \in Program$

ρ defined recursively on programs

Example (Semantics of assignment)

States s interpret flexible symbols f with $\mathcal{I}_s(f)$

$\rho(\mathtt{x=t;})(s) = s'$ where s' identical to s except $\mathcal{I}_{s'}(x) = val_s(t)$

Semantics of Programs

In labelled transition system $K = (States, \rho)$:
$\rho : Program \to (States \rightharpoonup States)$ is semantics of programs $\mathtt{p} \in Program$

ρ defined recursively on programs

Example (Semantics of assignment)

States s interpret flexible symbols f with $\mathcal{I}_s(f)$

$\rho(\mathtt{x=t;})(s) = s'$ where s' identical to s except $\mathcal{I}_{s'}(x) = val_s(t)$

Very tedious task to define ρ for JAVA. $\Rightarrow$ Not in this course.
Next lecture, we go directly to calculus for program formulas!

Literature for this Lecture

KeY Book Verification of Object-Oriented Software (see course web page), Chapter 10: Using KeY

KeY Book Verification of Object-Oriented Software (see course web page), Chapter 3: Dynamic Logic (Sections 3.1, 3.2, 3.4, 3.5, 3.6.1, 3.6.3, 3.6.4)