Formal Specification and Verification of Object-Oriented Programs

Dynamic Logic

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

Consider the JAVA method

```
public 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 such properties as:

If a $\neq$ null
then doubleContent terminates normally
and afterwards all elements of a have twice their old value

Motivation Cont'd

One such logic is dynamic logic (DL)

The above statement can be expressed in DL as follows:
(assuming a suitable signature)

$$
\begin{array}{ll}
 & \neg \mathtt{a} \doteq \mathtt{null} \\
 & \wedge\ \neg \mathtt{a} \doteq \mathtt{old\_a} \\
 & \wedge\ \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \wedge \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a[i]} \doteq \mathtt{old\_a[i]}) \\
\rightarrow & \langle \mathtt{doubleContent(a);} \rangle \\
 & \forall \mathtt{int}\ \mathtt{i};((0 \leq \mathtt{i} \wedge \mathtt{i} < \mathtt{a.length}) \rightarrow \mathtt{a[i]} \doteq 2 * \mathtt{old\_a[i]})
\end{array}
$$

Observations

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

Towards Dynamic Logic

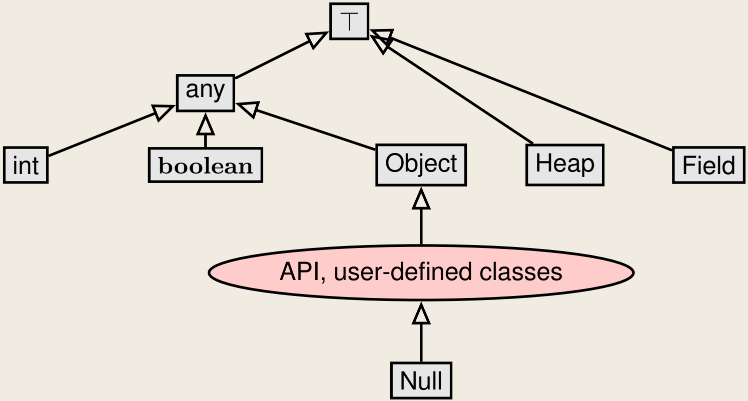
Reasoning about JAVA programs requires extensions of FOL

- JAVA type hierarchy
- JAVA program variables
- JAVA heap for reference types (next week)
- formalisation of arithmetics etc.

Modelling JAVA in FOL: Fixing a Type Hierarchy

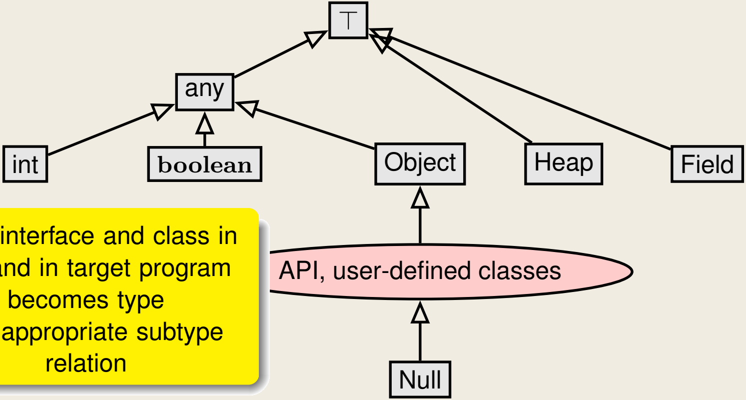
TECHNISCHE UNIVERSITÄT DARMSTADT

Signature based on JAVA's type hierarchy



Modelling JAVA in FOL: Fixing a Type Hierarchy

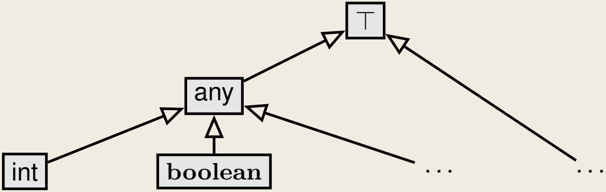
Signature based on JAVA's type hierarchy



Each interface and class in API and in target program becomes type with appropriate subtype relation

Subset of Types

Signature based on JAVA's type hierarchy



int and boolean are the only types for today
Class, interface types, etc., in next lecture

Modelling Dynamic Properties

TECHNISCHE UNIVERSITÄT DARMSTADT

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 program state at a time

Goal: Express behavior 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`

Requirements

TECHNISCHE UNIVERSITÄT DARMSTADT

Requirements for a logic to reason about programs

- can relate different program states, i.e., before and after execution, within a single formula
- program variables are represented by constant symbols that depend on current program state

Dynamic Logic meets the above requirements

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 in current state, then after executing the JAVA statement “`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

- Program variables such as `i` are state-dependent constant symbols
- Value of state-dependent symbols changeable by a program

Three words one meaning: $\boxed{\text{state-dependent, non-rigid, flexible}}$

Rigid versus Flexible Symbols

TECHNISCHE UNIVERSITÄT DARMSTADT

Signature of program logic defined as in FOL, but:
In addition there are program variables

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,...,+,*,...,<,...`
- Non-rigid (or flexible) symbols, interpretation depends on state

  Capture side effects on state during program execution
  - Program variables are flexible

Any term containing at least one flexible symbol is also flexible

Signature of Dynamic Logic

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition (Dynamic Logic Signature)

$\Sigma = (P_\Sigma,\ F_\Sigma,\ PV_\Sigma,\ V_\Sigma)$

(Rigid) Predicate Symbols	$P_\Sigma = \{>,\ >=, \ldots\}$
(Rigid) Function Symbols	$F_\Sigma = \{+,\ -,\ *,\ 0,\ 1, \ldots\}$
(Rigid) Logic Variable Symbols	$V_\Sigma = \{x, y, \ldots\}$
Non-rigid Program variables	$PV_\Sigma = \{i, j, k, \ldots\}$

All sets are pairwise disjoint.

Standard typing of JAVA symbols: **boolean** `TRUE; <(`**int**`,`**int**`); ...`

Dynamic Logic Signature - KeY input file

TECHNISCHE UNIVERSITÄT DARMSTADT

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

\programVariables { // non-rigid
   int i, j;
   boolean b;
}
```

Empty sections can be left out

Again: Two Kinds of Variables

Rigid:

Definition (First-Order/Logical Variables)

Typed logical variables (rigid), declared locally in quantifiers as `T x;` They may not occur in programs!

Non-rigid:

Program Variables

- Are not FO variables
- Cannot be quantified
- May occur in programs (and formulas)

Terms

- First-order terms defined as in FOL
- In addition:
  A program variable $\mathtt{v}$ of type T is a term of type T

Example

Signature for PV_Σ: $\mathbf{int}\ \mathtt{j};\ \mathbf{boolean}\ \mathtt{g}$
Quantified first-order variables: $\mathbf{int}\ x$; $\mathbf{boolean}\ b$;

- $\mathtt{j}$ and $\mathtt{j} + x$ are flexible terms of type $\mathbf{int}$
- $\mathtt{g}$ is a flexible term of type $\mathbf{boolean}$
- $x + x$ is a rigid term of type $\mathbf{int}$
- $\mathtt{j} + b$ and $\mathtt{j} + \mathtt{g}$ are not well-typed

Dynamic Logic Programs

Dynamic Logic = Typed FOL + programs . . .
Programs here: any legal sequence of JAVA statements

Example

Signature for PV_Σ: int r; int i; int n;
Signature for F_Σ: int 0; int +(int,int); int -(int,int);
Signature for P_Σ: <(int,int);

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

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 its final state (total correctness)
- $[p]\phi$: If $\mathtt{p}$ terminates then formula ϕ holds in its 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

TECHNISCHE UNIVERSITÄT DARMSTADT

Let `i`, `j`, `old_i`, `old_j` denote program variables of type **int**
Give the meaning in natural language:

1. $\mathtt{i} \doteq \mathtt{old\_i} \rightarrow \langle \mathtt{i = i + 1;} \rangle \mathtt{i} > \mathtt{old\_i}$
   "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. $\mathtt{i} \doteq \mathtt{old\_i} \rightarrow [\mathtt{while(true)}\{\mathtt{i = old\_i - 1;}\}]\mathtt{i} > \mathtt{old\_i}$
   "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. $\forall\, \mathbf{int}\ x;\ \left(\langle \mathtt{p} \rangle\ \mathtt{i} \doteq x \ \leftrightarrow\ \langle \mathtt{q} \rangle\ \mathtt{i} \doteq x \right)$
   "programs `p` and `q` are equivalent concerning termination and the final value of `i`"

Dynamic Logic: KeY Input File


```
\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
}
```

Visibility

- Program variables declared globally can be accessed anywhere
- Program variables declared inside a modality such as "*pre* $\rightarrow \langle$`int j; p`$\rangle$*post*" only visible in `p`

Dynamic Logic Formulas

Definition (Dynamic Logic (DL) Formulas, inductive definition)

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

Observations

- Program variables are flexible and never bound in quantifiers
- Program variables need not be initialized and can be declared outside of a program (unlike JAVA)
- Programs contain no logical variables
- Modality formulas can appear nested inside each other

Dynamic Logic Formulas Cont'd

TECHNISCHE UNIVERSITÄT DARMSTADT

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

- $\forall\,\mathbf{int}\ y; \big((\langle \mathtt{x\ =\ 1;}\rangle \mathtt{x} \doteq y) \leftrightarrow (\langle \mathtt{x\ =\ 1*1;}\rangle \mathtt{x} \doteq y)\big)$

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

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

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

Dynamic Logic Semantics: States

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

- Interpretation of (non-rigid) program variables can vary from state to state
- Interpretation of rigid symbols is the same in all states
  (e.g., built-in functions and predicates)

Program states as first-order states

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

- Only interpretation $\mathcal{I}$ of program variables can change
  $\Rightarrow$ only record values of $\mathtt{v} \in PV_\Sigma$
- Set of all states $\mathcal{S}$ is called *States*

Kripke Structure

Definition (First-Order Kripke Structure)

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

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

$$\rho(\mathrm{p})(\mathcal{S}_1) = \mathcal{S}_2$$

iff

program p executed in state $\mathcal{S}_1$ terminates and its final state is $\mathcal{S}_2$, otherwise undefined

- $\mathcal{D}$ is identical for all states (constant domain assumption) and fixed for pre-defined types, e.g., $D^{\mathbf{int}} = \mathbb{Z}$
- $\mathcal{I}$ is fixed for pre-defined symbols like $+, -, /, \%$ to their canonical meaning
- except for program variables: $\mathcal{I}$ is identical f. a. $S \in States$ of a given K

Kripke Structure

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition (First-Order Kripke Structure)

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

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

$$\rho(\mathrm{p})(\mathcal{S}_1) = \mathcal{S}_2$$

iff

program p executed in state $\mathcal{S}_1$ terminates and its final state is $\mathcal{S}_2$, otherwise undefined

- $\mathcal{D}$ is identical for all states (constant domain assumption) and fixed for pre-defined types, e.g., $D^{\mathbf{int}} = \mathbb{Z}$
- $\mathcal{I}$ is fixed for pre-defined symbols like $+, -, /, \%$ to their canonical meaning
- except for program variables: $\mathcal{I}$ is identical f. a. $S \in States$ of a given K

Semantic Evaluation of DL Formulas

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition (Validity Relation for DL Formulas)

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

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

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

A DL formula ϕ is valid iff $\mathcal{S} \models \phi$ for all states $\mathcal{S}$.

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

Valid?
Intuitive meaning?

Example

$\forall\, T\ y;\ \big((\langle \mathrm{p}\rangle \mathrm{x} \doteq y) \ \leftrightarrow\ (\langle \mathrm{q}\rangle \mathrm{x} \doteq y)\big)$

Not valid in general

Programs p and q behave equivalently on variable $T\ \mathrm{x}$.

Example

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

Not valid in general

Program p terminates provided that initial value of x is suitably chosen

Semantics of Programs

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

ρ defined recursively on programs

Example (Semantics of assignment)

States $\mathcal{S}$ interpret program variables $\mathtt{v}$ with $\mathcal{I}_{\mathcal{S}}(\mathtt{v})$

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

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