Formal Methods for Software Development

Reasoning about Programs with Dynamic Logic

Andreas Lööw, slides by Wolfgang Ahrendt

10 October 2019

Part I

Where are we?

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++;
  }
}
```

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

Dynamic Logic (Preview)

One such logic is dynamic logic (DL)

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

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

Observations

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

Today

introducing dynamic logic for JAVA

- short recap first-order logic (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 type symbols
- 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'

Recap: First-Order States

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 $\mathcal{I}(f) : \mathcal{D}^{\tau_1} \times \cdots \times \mathcal{D}^{\tau_r} \rightarrow \mathcal{D}^{\tau}$

$\mathcal{I}(p)$ 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

Part II

Towards Dynamic Logic

Towards Dynamic Logic

Reasoning about Java programs requires extensions of FOL

- JAVA type hierarchy
- JAVA program variables
- JAVA heap for reference types (next lecture)

Type Hierarchy

Definition (Type Hierarchy)

- T_Σ is set of types
- Subtype relation $\sqsubseteq \; \subseteq \; T_\Sigma \times T_\Sigma$ with top element $\top$
  - $\tau \sqsubseteq \top$ for all $\tau \in T_\Sigma$

Example (A Minimal Type Hierarchy)

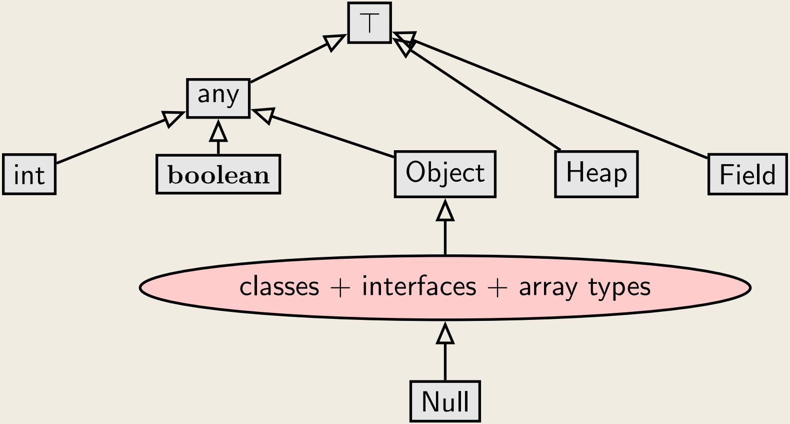
$T_\Sigma = \{\top\}$
All signature symbols have same type $\top$

Example (Type Hierarchy for Java)

(see next slide)

Modelling Java in FOL: Fixing a Type Hierarchy

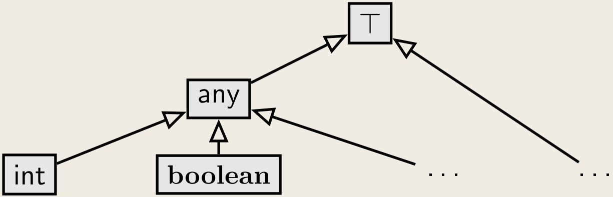
Signature based on Java's type hierarchy (sketch)



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, interfaces, arrays: next lecture.

Modelling Dynamic Properties

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

- Values of fields in a certain range
- Invariant of a class implies invariant of its interface

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

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, whose value depend on program state

Dynamic Logic meets the above requirements

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)
- $+ \ldots$ (later)

An Example

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

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 $+ \ldots$

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

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

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

Rigid versus Flexible

- Rigid symbols, meaning insensitive to program states
  - First-order variables (aka logical variables)
  - Built-in functions and predicates such as `0,1,...,+,*,...,<,...`
- Flexible (or non-rigid) symbols, meaning depends on state. Capture side effects on state during program execution
  - Program variables are flexible

Any term containing at least one flexible symbol is called flexible

Signature of Dynamic Logic

Definition (Dynamic Logic Signature)

$\Sigma = (P_\Sigma,\, F_\Sigma,\, PV_\Sigma,\, \alpha_\Sigma), \quad F_\Sigma \cap PV_\Sigma = \emptyset$

(Rigid) Predicate Symbols $P_\Sigma = \{>, >=, \ldots\}$
(Rigid) Function Symbols $F_\Sigma = \{+, -, *, 0, 1, \ldots\}$
Flexible Program variables e.g. $PV_\Sigma = \{\mathtt{i}, \mathtt{j}, \mathtt{ready}, \ldots\}$

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

Dynamic Logic Signature - KeY input file

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

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

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 must not occur in programs!

Flexible:

Program Variables

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

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;
```

Which value does the program compute in r?

Relating Program States: Modalities

DL extends FOL with two additional operators:

- $\langle \mathrm{p} \rangle \phi$ (diamond)
- $[\mathrm{p}]\phi$ (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)
- $[\mathrm{p}]\phi$: 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. $\mathtt{i} = \mathtt{old\_i} \rightarrow \langle \mathtt{i = i + 1;} \rangle \mathtt{i} > \mathtt{old\_i}$

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

   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. $\forall\, x.\ (\ \langle prog_1 \rangle\ \mathtt{i} = x\ \leftrightarrow\ \langle prog_2 \rangle\ \mathtt{i} = x\ )$

   $prog_1$ and $prog_2$ are equivalent concerning termination and the final value of `i`.

Dynamic Logic: KeY Input File

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

```
\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 only visible therein. E.g., in “$\textit{pre} \rightarrow \langle \texttt{\textbf{int} j; p} \rangle \textit{post}$”, j not visible in post

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

- 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, e.g., $\langle \mathrm{p}\rangle[\mathrm{q}]\phi$

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 PV_Σ contains $\mathbf{int}\ \texttt{x;}$
- $\exists\,\mathbf{int}\ x; [x = 1;](x = 1)$

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

  Well-formed if PV_Σ 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) 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

We identify first-order state $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ with program state.

- Interpretation $\mathcal{I}$ only changes on program variables.
  $\Rightarrow$ Enough to record values of variables $\in PV_{\Sigma}$
- Set of all states $\mathcal{S}$ is called *States*

Kripke Structure

Definition (Kripke Structure)

Kripke Structure or Labelled Transition System $K = (\mathit{States}, \rho)$

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

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

iff.

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

- ρ is the semantics of programs $\in \mathit{Program}$
- $\rho(\mathrm{p})(\mathcal{S})$ can be undefined ('$\rightharpoonup$'):
  p may not terminate when started in $\mathcal{S}$
- JAVA 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)

- $\mathcal{S} \models \langle \mathrm{p} \rangle \phi$ 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

Meaning?

Example

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

Programs p and q behave equivalently on variable τ x.

Example

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

Program p terminates if initial value of x is suitably chosen.

Semantics of Programs

In labelled transition system $K = (\mathit{States}, \rho)$:
$\rho : \mathit{Program} \to (\mathit{States} \rightharpoonup \mathit{States})$ 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 $\mathcal{I}_{\mathcal{S}}(\mathtt{v})$

$$\rho(\mathtt{x=t;})(\mathcal{S}) = \mathcal{S}' \quad \text{where} \quad \mathcal{I}_{\mathcal{S}'}(y) := \begin{cases} \mathcal{I}_{\mathcal{S}}(y) & y \neq \mathtt{x} \\ \mathit{val}_{\mathcal{S}}(\mathtt{t}) & y = \mathtt{x} \end{cases}$$

Very advanced task to define ρ for JAVA ⇒ Not done in this course
Next lecture, we go directly to calculus for program formulas!

Literature for this Lecture

KeYbook *W. Ahrendt*, B. Beckert, R. Bubel, R. Hähnle, P. Schmitt, M. Ulbrich, editors.
Deductive Software Verification - The KeY Book
Vol 10001 of *LNCS*, Springer, 2016
(E-book at `link.springer.com`)

- *W. Ahrendt*, S. Grebing, Using the KeY Prover
  Chapter 15 in [KeYbook]

further reading:

- B. Beckert, V. Klebanov, B. Weiß, Dynamic Logic for Java
  Chapter 3 in [KeYbook]