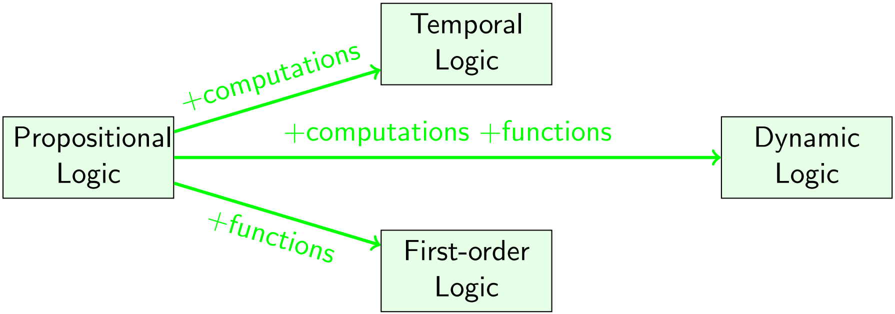
Formal Specification and Verification

Reasoning about Programs with Dynamic Logic

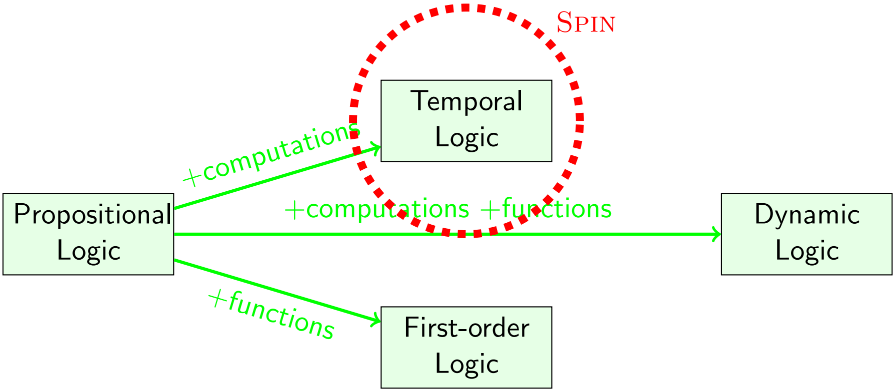
Bernhard Beckert

Based on a lecture by Wolfgang Ahrendt and Reiner Hähnle at Chalmers University, Göteborg

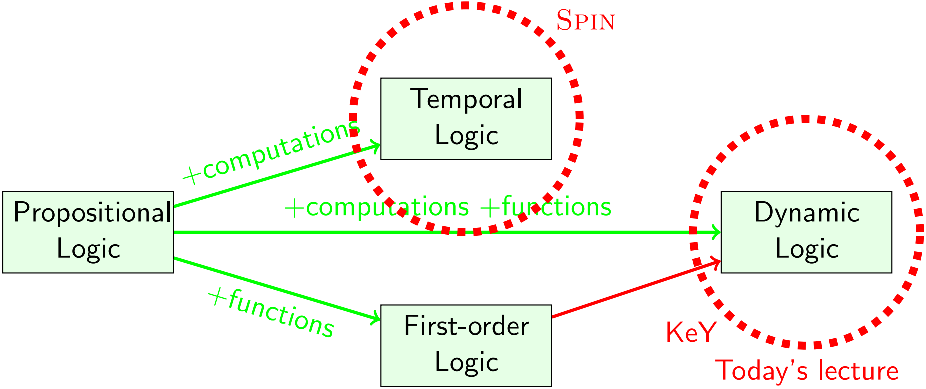
Beyond Propositional Logic



Beyond Propositional Logic



Beyond Propositional Logic



State Dependence of Formula Evaluation

Closed FOL formula is either valid or not valid wrt model $\mathcal{M}$

Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ as program state

Let `x` be (local) program variable or attribute

Execution of program `p` may change program state, i.e., value of `x`

State Dependence of Formula Evaluation

Closed FOL formula is either valid or not valid wrt model $\mathcal{M}$

Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ as program state

Let $\mathtt{x}$ be (local) program variable or attribute

Execution of program $\mathtt{p}$ may change program state, i.e., value of $\mathtt{x}$

Example

Executing `x=3;` results in $\mathcal{M}$ such that $\mathcal{M} \models \mathtt{x} \doteq 3$

Executing `x=4;` results in $\mathcal{M}$ such that $\mathcal{M} \not\models \mathtt{x} \doteq 3$

State Dependence of Formula Evaluation

Closed FOL formula is either valid or not valid wrt model $\mathcal{M}$

Consider $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ as program state

Let `x` be (local) program variable or attribute

Execution of program `p` may change program state, i.e., value of `x`

Example

Executing `x=3;` results in $\mathcal{M}$ such that $\mathcal{M} \models \mathtt{x} \doteq 3$

Executing `x=4;` results in $\mathcal{M}$ such that $\mathcal{M} \not\models \mathtt{x} \doteq 3$

Need a logic to capture state before/after program execution

Rigid versus Flexible Symbols

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

Rigid versus Flexible

- Rigid symbols, same interpretation in all program states
  - First-order variables (aka logical variables)
    Used to hold initial values of program 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

  - Functions modeling program variables and attributes are flexible

Any term containing at least one flexible symbol is also flexible

Signature of Dynamic Logic (Simple Version)

Definition (Dynamic Logic Signature)

First-order signature $\Sigma = (\mathsf{PSym}_r, \mathsf{FSym}_r, \mathsf{FSym}_{nr}, \alpha)$

Rigid Predicate Symbols	$\mathsf{PSym} = \{>, >=, \ldots\}$
Rigid Function Symbols	$\mathsf{FSym} = \{+, -, *, 0, 1, \ldots, \mathbf{true}, \mathbf{false}\}$
Non-rigid Function Symbols	$\mathsf{FSym} = \{i, j, k, \ldots, p, q, r, \ldots\}$

Type hierarchy

$\mathtt{T} = \{\bot, \mathbf{int}, \mathbf{boolean}, \top\}$ with **int**, **boolean** incomparable

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

Variables

Definition (First-Order/Logical Variables)

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

Program Variables

Non-rigid constants **int** `i;` **boolean** `p` used as program variables

Terms

- First-order terms defined as in FOL
- First-order terms may contain rigid and non-rigid symbols
- $\mathrm{FSym}_r \cap \mathrm{FSym}_{nr} = \emptyset$

Example

Signature for FSym_{nr}: $\mathbf{int}\ \mathtt{j};\ \mathbf{boolean}\ \mathtt{p}$
Variables $\mathbf{int}\ x;\ \mathbf{boolean}\ b;$

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

Atomic Programs

Definition (Atomic Programs)

The atomic programs Π_0 are assignments of the form $\mathtt{j} = t$ where:

- `T j;` is a program variable (flexible constant)
- t is a first-order term of type `T` without logical variables

Example

Signature for FSym_{nr}: **int** `j`; **boolean** `p`
Variables **int** x; **boolean** b;

- `j=j+1`, `j=0` and `p=`**false** are assignments
- `j=j+`x contains a logical variable on the right
- x`=1` contains a logical variable on the left
- $\mathtt{j} \doteq \mathtt{j}$ is equality, not assignment
- `p=0` is ill-typed

Dynamic Logic Programs (Simple Version)

Definition (Program)

Inductive definition of the set of (DL) programs Π:

- If π is an atomic program, then π; is a program
- If p and q are programs, then pq is a program
- If b is a variable-free term of type `boolean`, p and q programs, then

  if (b) p **else** q;　　　　**if** (b) p;

  are programs
- If b is a variable-free term of type `boolean`, p a program, then

  while (b) p;

  is a program

Dynamic Logic Programs (Simple Version)

Definition (Program)

Inductive definition of the set of (DL) programs Π:

- If π is an atomic program, then π; is a program
- If p and q are programs, then pq is a program
- If b is a variable-free term of type `boolean`, p and q programs, then

  if (b) p **else** q;   **if** (b) p;

  are programs
- If b is a variable-free term of type `boolean`, p a program, then

  while (b) p;

  is a program

Programs contain no logical variables!

Dynamic Logic Programs Cont'd

Example (Admissible Program)

Signature for FSym_{nr}: **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 Cont'd

Example (Admissible Program)

Signature for FSym_{nr}: `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`?

Dynamic Logic Formulas (Simple Version)

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 (Simple Version)

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

Dynamic Logic Formulas Cont'd

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

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

Dynamic Logic Formulas Cont'd

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

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

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

Dynamic Logic Formulas Cont'd

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

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

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

Dynamic Logic Formulas Cont'd

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

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

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

  Not well-formed, because logical variable occurs in program

Dynamic Logic Formulas Cont'd

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

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

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

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

Dynamic Logic Formulas Cont'd

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

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

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

  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 FSym_{nr} contains $\mathbf{int}\ \texttt{x;}$
  program formulas can be nested

Dynamic Logic Semantics: States

First-order model can be considered as program state

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

States as first-order models

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

- Only interpretation $\mathcal{I}$ of non-rigid symbols in FSym_{nr} can change
  $\Rightarrow$ only record values of $f \in \mathrm{FSym}_{nr}$: use s (for state) instead of $\mathcal{M}$
- Set of all states s is S

Dynamic Logic Semantics: Kripke Structure

Definition (Kripke Structure (aka Labelled Transition System))

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

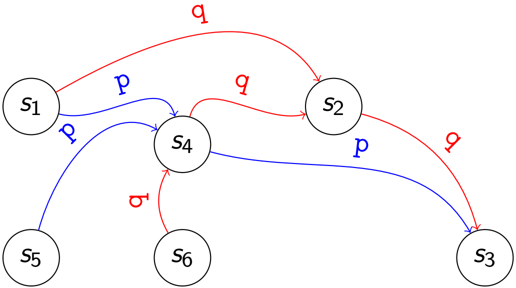
- State (=first-order model) $s = (\mathcal{D}, \delta, \mathcal{I}) \in S$
- Transition relation $\rho : \Pi \rightarrow (S \rightarrow S)$
  - ρ is the operational semantics of programs Π
  - Each program $\mathrm{p} \in \Pi$ transforms a start state s into end state $\rho(\mathrm{p})(s)$
  - $\rho(\mathrm{p})(s)$ can be undefined: p does not terminate when started in s
  - Our programs are deterministic (unlike PROMELA): $\rho(\mathrm{p})$ is a function (at most one value)

Dynamic Logic Semantics: Kripke Structure Cont'd

Example (Kripke Structure)

Two programs p and q

Show $\rho(\mathtt{p})$ and $\rho(\mathtt{q})$, states $S = \{s1, \ldots, s_6\}$



When p is started in s_5 it terminates in s_4, etc.

In general, Π and S are infinite!

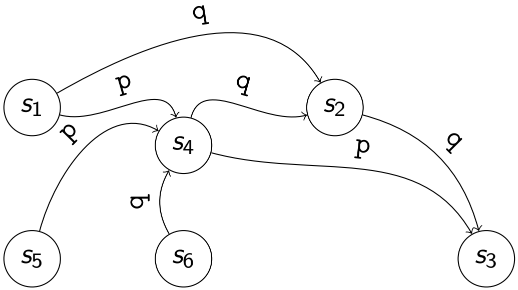
Definition (Validity Relation for Program Formulas)

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

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

- $s, \beta \models [\mathtt{p}] \phi$ iff $\rho(\mathtt{p})(s), \beta \models \phi$ whenever $\rho(\mathtt{p})(s)$ is defined

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

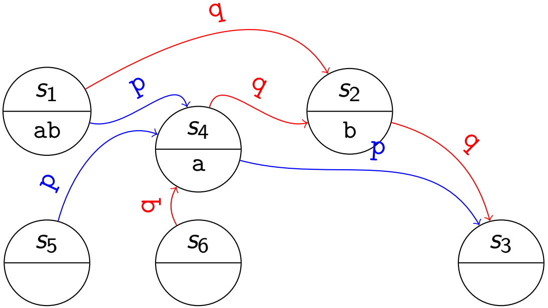


Dynamic Logic Semantics: Kripke Structure Cont'd

Example (Semantic Evaluation of Program Formulas)

Signature FSym_{nr}: $\mathbf{boolean}\ \mathtt{a};\ \mathbf{boolean}\ \mathtt{b};$

Notation: $\mathcal{I}(x) = T$ iff x appears in lower compartment



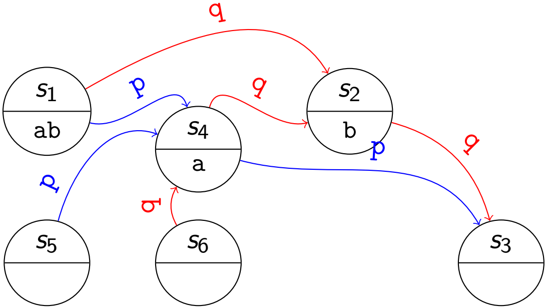
Question 1: $s_1 \models \langle \mathrm{p} \rangle (\mathtt{a} \doteq \mathbf{true})$?

Dynamic Logic Semantics: Kripke Structure Cont'd

Example (Semantic Evaluation of Program Formulas)

Signature FSym_{nr}: **boolean** `a`; **boolean** `b`;

Notation: $\mathcal{I}(x) = T$ iff x appears in lower compartment



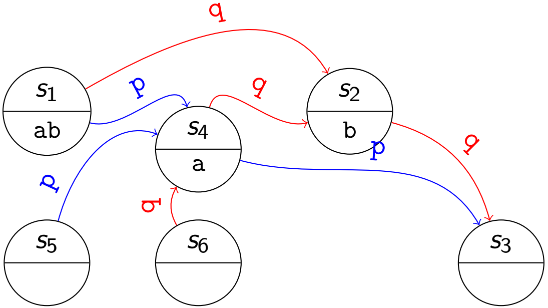
Question 2: $s_1 \models \langle \mathtt{q} \rangle (\mathtt{a} \doteq \mathbf{true})$?

Dynamic Logic Semantics: Kripke Structure Cont'd

Example (Semantic Evaluation of Program Formulas)

Signature FSym_{nr}: **boolean** `a`; **boolean** `b`;

Notation: $\mathcal{I}(x) = T$ iff x appears in lower compartment

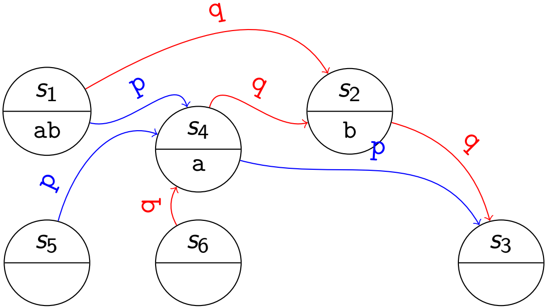


Question 3: $s_5 \models \langle \mathtt{q} \rangle (\mathtt{a} \doteq \mathbf{true})$?

Example (Semantic Evaluation of Program Formulas)

Signature FSym_{nr}: **boolean** `a`; **boolean** `b`;

Notation: $\mathcal{I}(x) = T$ iff x appears in lower compartment



Question 4: $s_5 \models [\mathtt{q}](\mathtt{a} \doteq \mathbf{true})$?

Program Correctness

Definition (Notions of Correctness)

- If $s, \beta \models \langle \mathrm{p} \rangle \phi$ then
  p totally correct (with respect to ϕ) in s, β
- If $s, \beta \models [\mathrm{p}]\phi$ then
  p partially correct (with respect to ϕ) in s, β

- Duality $\langle \mathrm{p} \rangle \phi$ iff $![\mathrm{p}]\,!\,\phi$
  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

Semantics of Sequents

$\Gamma = \{\phi_1, \ldots, \phi_n\}$ and $\Delta = \{\psi_1, \ldots, \psi_m\}$ sets of program formulas where all logical variables occur bound

Recall: $s \models (\Gamma \Longrightarrow \Delta)$ iff $s \models (\phi_1 \;\&\; \cdots \;\&\; \phi_n) \;\texttt{->}\; (\psi_1 \mid \cdots \mid \psi_m)$

Define semantics of DL sequents identical to semantics of FOL sequents

Definition (Validity of Sequents over Program Formulas)

A sequent $\Gamma \Longrightarrow \Delta$ over program formulas is valid iff

$$s \models (\Gamma \Longrightarrow \Delta) \text{ in all states } s$$

Semantics of Sequents

$\Gamma = \{\phi_1, \ldots, \phi_n\}$ and $\Delta = \{\psi_1, \ldots, \psi_m\}$ sets of program formulas where all logical variables occur bound

Recall: $s \models (\Gamma \Longrightarrow \Delta)$ iff $s \models (\phi_1 \,\&\, \cdots \,\&\, \phi_n) \;\texttt{->}\; (\psi_1 \mid \cdots \mid \psi_m)$

Define semantics of DL sequents identical to semantics of FOL sequents

Definition (Validity of Sequents over Program Formulas)

A sequent $\Gamma \Longrightarrow \Delta$ over program formulas is valid iff

$$s \models (\Gamma \Longrightarrow \Delta) \text{ in all states } s$$

Consequence for program variables

Initial value of program variables implicitly "universally quantified"

Initial States

Java initial states

KeY prover "starts" programs in initial states according to JAVA convention:

- Values of array entries initialized to default values: **int**[] to 0, etc.
- Static object initialization
- No objects created

How to restrict validity to set of initial states $S_0 \subseteq S$?

1. Design closed FOL formula Init with
   $s \models \mathrm{Init}$ iff $s \in S_0$
2. Use sequent $\Gamma, \mathrm{Init} \Longrightarrow \Delta$

Later: simple method for specifying initial value of program variables

Operational Semantics of Programs

In labelled transition system $K = (S, \rho)$:
$\rho : \Pi \to (S \to S)$ is operational semantics of programs $\mathtt{p} \in \Pi$

How is ρ defined for concrete programs and states?

Operational Semantics of Programs

In labelled transition system $K = (S, \rho)$:
$\rho : \Pi \to (S \to S)$ is operational semantics of programs $\mathtt{p} \in \Pi$

How is ρ defined for concrete programs and states?

Example (Operational semantics of assignment)

States s interpret non-rigid 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 ...
$\Rightarrow$ go directly to calculus for program formulas!

Symbolic Execution of Programs

Sequent calculus decomposes top-level operator in formula
What is "top-level" in a sequential program `p; q; r` ?

Symbolic Execution (King, late 60s)

- Follow the natural control flow when analysing a program
- Values of some variables unknown: symbolic state representation

Symbolic Execution of Programs

Sequent calculus decomposes top-level operator in formula
What is "top-level" in a sequential program `p; q; r` ?

Symbolic Execution (King, late 60s)

- Follow the natural control flow when analysing a program
- Values of some variables unknown: symbolic state representation

Example

Compute the final state after termination of

```
int x; int y; x=x+y; y=x-y; x=x-y;
```

Symbolic Execution of Programs Cont'd

General form of rule conclusions in symbolic execution calculus

$$\langle \texttt{stmt; rest} \rangle \phi, \qquad [\texttt{stmt; rest}]\phi$$

- Rules must symbolically execute first statement
- Repeated application of rules in a proof corresponds to symbolic program execution

Symbolic Execution of Programs Cont'd

Symbolic execution of assignment

$$\text{assign}\ \frac{\{\mathtt{x}/\mathtt{x}_{old}\}\Gamma,\ \mathtt{x} \doteq \{\mathtt{x}/\mathtt{x}_{old}\}t \quad \Longrightarrow \quad \langle\mathtt{rest}\rangle\phi,\ \{\mathtt{x}/\mathtt{x}_{old}\}\Delta}{\Gamma \Longrightarrow \langle\mathtt{x\ =\ t;\ rest}\rangle\phi, \Delta}$$

$\mathtt{x}_{old}$ new program variable that "rescues" old value of $\mathtt{x}$

Symbolic Execution of Programs Cont'd

Symbolic execution of assignment

$$\text{assign}\quad \frac{\{\mathtt{x}/\mathtt{x}_{old}\}\Gamma,\ \mathtt{x} \doteq \{\mathtt{x}/\mathtt{x}_{old}\}t \quad\Longrightarrow\quad \langle\mathtt{rest}\rangle\phi,\ \{\mathtt{x}/\mathtt{x}_{old}\}\Delta}{\Gamma \Longrightarrow \langle\mathtt{x\ =\ t;\ rest}\rangle\phi, \Delta}$$

$\mathtt{x}_{old}$ new program variable that "rescues" old value of $\mathtt{x}$

Example

Conclusion matching: $\{\mathtt{x}/\mathtt{x}\}$, $\{\mathtt{t}/\mathtt{x+y}\}$, $\{\mathtt{rest}/\mathtt{y=x-y;\ x=x-y;}\}$, $\{\phi/(\mathtt{x} \doteq y_0\ \&\ \mathtt{y} \doteq x_0)\}$, $\{\Gamma/\mathtt{x} \doteq x_0, \mathtt{y} \doteq y_0\}$, $\{\Delta/\emptyset\}$

$$\frac{\mathtt{x}_{old} \doteq x_0,\ \mathtt{y} \doteq y_0,\ \mathtt{x} \doteq \mathtt{x}_{old}\mathtt{+y} \Longrightarrow \langle\mathtt{y=x-y;\ \ x=x-y;}\rangle(\mathtt{x} \doteq y_0\ \&\ \mathtt{y} \doteq x_0)}{\mathtt{x} \doteq x_0,\ \mathtt{y} \doteq y_0 \Longrightarrow \langle\mathtt{x=x+y;\ \ y=x-y;\ \ x=x-y;}\rangle(\mathtt{x} \doteq y_0\ \&\ \mathtt{y} \doteq x_0)}$$

Proving Partial Correctness

Partial correctness assertion

If program p is started in a state satisfying Pre and terminates, then its final state satisfies Post

In Hoare logic $\{\text{Pre}\}\ \mathtt{p}\ \{\text{Post}\}$ (Pre, Post must be FOL)

In DL Pre −> $[\mathtt{p}]$Post (Pre, Post any DL formula)

Proving Partial Correctness

Partial correctness assertion

If program p is started in a state satisfying Pre and terminates, then its final state satisfies Post

In Hoare logic {Pre} p {Post} (Pre, Post must be FOL)

In DL Pre –> [p]Post (Pre, Post any DL formula)

Example (In KeY Syntax, Demo automatic proof)

```
\programVariables {
  int x; int y; }

\problem {
  (\forall int x0; \forall int y0; ((x=x0 & y=y0) ->
     \<{x=x+y; y=x-y; x=x-y;}\>(x=y0 & y=x0)))
}
```

More Properties

Example

$$\forall\, T\ y;\ ((\langle \mathtt{p} \rangle \mathtt{x} \doteq y)\ \ \texttt{<->}\ \ (\langle \mathtt{q} \rangle \mathtt{x} \doteq y))$$

More Properties

Example

$\forall\ T\ y; ((\langle \mathtt{p} \rangle \mathtt{x} \doteq y)\ \ \texttt{<->}\ \ (\langle \mathtt{q} \rangle \mathtt{x} \doteq y))$

Not valid in general

Programs `p` behave `q` equivalently on variable T `x`

More Properties

Example

$\forall\, T\ y;\ (((\langle \mathtt{p} \rangle \mathtt{x} \doteq y)\ \texttt{<->}\ (\langle \mathtt{q} \rangle \mathtt{x} \doteq y))$

Not valid in general

Programs p behave q equivalently on variable T x

Example

$\exists\, T\ y;\ (\mathtt{x} \doteq y\ \texttt{->}\ \langle \mathtt{p} \rangle \mathbf{true})$

More Properties

Example

$\forall\, T\ y; (((\langle \mathrm{p} \rangle \mathrm{x} \doteq y)\ \ \texttt{<->}\ \ (\langle \mathrm{q} \rangle \mathrm{x} \doteq y))$

Not valid in general

Programs p behave q equivalently on variable T x

Example

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

Not valid in general

Program p terminates in all states where x has suitable initial value

Symbolic Execution of Programs Cont'd

Symbolic execution of conditional

$$\text{if}\ \frac{\Gamma, \mathtt{b} \doteq \mathbf{true} \Longrightarrow \langle \mathtt{p;\ rest} \rangle \phi, \Delta \qquad \Gamma, \mathtt{b} \doteq \mathbf{false} \Longrightarrow \langle \mathtt{q;\ rest} \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \mathbf{if}\ \mathtt{(b)\ \{\ p\ \}}\ \mathbf{else}\ \mathtt{\{\ q\ \}\ ;\ rest} \rangle \phi, \Delta}$$

Symbolic execution must consider all possible execution branches

Symbolic Execution of Programs Cont'd

Symbolic execution of conditional

$$\text{if}\ \frac{\Gamma, \mathtt{b} \doteq \mathbf{true} \Longrightarrow \langle \mathtt{p;\ rest} \rangle \phi, \Delta \qquad \Gamma, \mathtt{b} \doteq \mathbf{false} \Longrightarrow \langle \mathtt{q;\ rest} \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \mathbf{if}\ \mathtt{(b)\ \{\ p\ \}}\ \mathbf{else}\ \mathtt{\{\ q\ \}\ ;\ rest} \rangle \phi, \Delta}$$

Symbolic execution must consider all possible execution branches

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \langle \mathbf{if}\ \mathtt{(b)\ \{\ p;}\ \mathbf{while}\ \mathtt{(b)\ p\};\ r} \rangle \phi, \Delta}{\Gamma \Longrightarrow \langle \mathbf{while}\ \mathtt{(b)\ \{p\};\ r} \rangle \phi, \Delta}$$

Quantifying over Program Variables

How to express correctness for any initial value of program variable?

Quantifying over Program Variables

How to express correctness for any initial value of program variable?

Not allowed: $\forall\, T\ \mathtt{i};\ \langle \mathtt{...i...} \rangle \phi$ (program $\neq$ logical variable)

Quantifying over Program Variables

How to express correctness for any initial value of program variable?

Not allowed: $\forall\, T\ \mathtt{i};\, \langle \mathtt{...i...} \rangle \phi$ (program $\neq$ logical variable)

Not intended: $\Longrightarrow \langle \mathtt{...i...} \rangle \phi$ (Validity of sequents: quantification over all states)

Quantifying over Program Variables

How to express correctness for any initial value of program variable?

Not allowed: $\forall\, T\ \mathtt{i};\ \langle \mathtt{...i...} \rangle \phi$ (program $\neq$ logical variable)

Not intended: $\Longrightarrow \langle \mathtt{...i...} \rangle \phi$ (Validity of sequents: quantification over all states)

As previous: $\forall\, T\ i_0;\ (i_0 \doteq \mathtt{i}\ \ \texttt{->}\ \ \langle \mathtt{...i...} \rangle \phi)$

Quantifying over Program Variables

How to express correctness for any initial value of program variable?

Not allowed: $\forall\, T\ \mathtt{i};\ \langle \ldots \mathtt{i} \ldots \rangle \phi$ (program $\neq$ logical variable)

Not intended: $\Longrightarrow \langle \ldots \mathtt{i} \ldots \rangle \phi$ (Validity of sequents: quantification over all states)

As previous: $\forall\, T\ i_0;\ (i_0 \doteq \mathtt{i}\ \texttt{->}\ \langle \ldots \mathtt{i} \ldots \rangle \phi)$

Solution

Use explicit construct to record values in current state

Update $\forall\, T\ i_0;\ (\{\mathtt{i} := i_0\} \langle \ldots \mathtt{i} \ldots \rangle \phi)$

Explicit State Updates

Updates specify computation state where formula is evaluated

Definition (Syntax of Updates)

If $\mathtt{v}$ is program variable, t FOL term type-compatible with $\mathtt{v}$,
t' any FOL term, and ϕ any DL formula, then

- $\{\mathtt{v} := t\}t'$ is DL term
- $\{\mathtt{v} := t\}\phi$ is DL formula

Definition (Semantics of Updates)

State s interprets non-rigid symbols f with $\mathcal{I}_s(f)$
β variable assignment for logical variables in t

$\rho(\{\mathtt{v} := t\})(s) = s'$ where s' identical to s except $\mathcal{I}_{s'}(x) = val_{s,\beta}(t)$

Explicit State Updates Cont'd

Facts about updates $\{\mathtt{v} := t\}$

- Update semantics identical to assignment
- Value of update depends on logical variables in t: use β
- Updates as "lazy" assignments (no term substitution done)
- Updates are not assignments: right-hand side is FOL term

  $\{\mathtt{x} := n\}\phi$ cannot be turned into assignment (n logical variable)

  $\langle\mathtt{x=i++;}\rangle\phi$ cannot directly be turned into update
- Updates are not equations: change value of non-rigid terms

Computing Effect of Updates (Automatic)

Rewrite rules for update followed by ...

program variable
$$\begin{cases} \{\mathrm{x} := t\}\mathrm{y} & \leadsto \quad \mathrm{y} \\ \{\mathrm{x} := t\}\mathrm{x} & \leadsto \quad t \end{cases}$$

logical variable $\{\mathrm{x} := t\}w \leadsto w$

complex term $\{\mathrm{x} := t\}f(t_1, \ldots, t_n) \leadsto f(\{\mathrm{x} := t\}t_1, \ldots, \{\mathrm{x} := t\}t_n)$

FOL formula
$$\begin{cases} \{\mathrm{x} := t\}(\phi \;\&\; \psi) \leadsto \{\mathrm{x} := t\}\phi \;\&\; \{\mathrm{x} := t\}\psi \\ \qquad\qquad \ldots \\ \{\mathrm{x} := t\}(\forall\, T\; y;\, \phi) \leadsto \forall\, T\; y;\, (\{\mathrm{x} := t\}\phi) \end{cases}$$

program formula $\{\mathrm{x} := t\}(\langle \mathrm{p} \rangle \phi) \leadsto \{\mathrm{x} := t\}(\langle \mathrm{p} \rangle \phi)$ unchanged!

Update computation delayed until p symbolically executed

Assignment Rule Using Updates

Symbolic execution of assignment using updates

$$\text{assign}\ \frac{\Gamma \Longrightarrow \{\mathtt{x} := t\}\langle\mathtt{rest}\rangle\phi, \Delta}{\Gamma \Longrightarrow \langle\mathtt{x\ =\ t;\ rest}\rangle\phi, \Delta}$$

- Avoids renaming of program variables
- Works as long as t has no side effects (ok in simple DL)
- Special cases for $\mathtt{x} = t_1 + t_2$, etc.

Demo

`Examples/lect11/swap.key`

Example Proof

Example

```
\programVariables {
  int x;
}
\problem {
   (\exists int y;
     ({x := y}\<{while (x > 0) {x = x-1;}}\> x=0 ))
}
```

Intuitive Meaning? Satisfiable? Valid?

Demo

`Examples/lect11/term.key`

Example Proof

Example

```
\programVariables {
  int x;
}
\problem {
  (\exists int y;
    ({x := y}\<{while (x > 0) {x = x-1;}}\> x=0 ))
}
```

Intuitive Meaning? Satisfiable? Valid?

Demo

`Examples/lect11/term.key`

What to do when we cannot determine a concrete loop bound?

Parallel Updates

How to apply updates on updates?

Example

Symbolic execution of

```
int x; int y; x=x+y; y=x-y; x=x-y;
```

yields:

$\{x := x+y\}\{y := x-y\}\{x := x-y\}$

Need to compose three sequential state changes into a single one!

Parallel Updates Cont'd

Definition (Parallel Update)

A parallel update is expression of the form $\{l_1 := v_1 || \cdots || l_n := v_n\}$ where each $\{l_i := v_i\}$ is simple update

- All v_i computed in old state before update is applied
- Updates of all locations l_i executed simultaneously
- Upon conflict $l_i = l_j,\ v_i \neq v_j$ later update ($\max\{i, j\}$) wins

Parallel Updates Cont'd

Definition (Parallel Update)

A parallel update is expression of the form $\{l_1 := v_1 || \cdots || l_n := v_n\}$ where each $\{l_i := v_i\}$ is simple update

- All v_i computed in old state before update is applied
- Updates of all locations l_i executed simultaneously
- Upon conflict $\quad l_i = l_j,\ v_i \neq v_j \quad$ later update $(\max\{i, j\})$ wins

Definition (Composition Sequential Updates/Conflict Resolution)

$$\{l_1 := r_1\}\{l_2 := r_2\} \; = \; \{l_1 := r_1 || l_2 := \{l_1 := r_1\}r_2\}$$

$$\{l_1 := v_1 || \cdots || l_n := v_n\}\mathtt{x} \; = \; \begin{cases} \mathtt{x} & \text{if } \mathtt{x} \notin \{l_1, \ldots, l_n\} \\ v_k & \text{if } \mathtt{x} = l_k,\ \mathtt{x} \notin \{l_{k+1}, \ldots, l_n\} \end{cases}$$

Parallel Updates Cont'd

Example

```
({x := x+y}{y := x-y}){x := x-y} =
{x := x+y || y := (x+y)-y}{x := x-y} =
{x := x+y || y := (x+y)-y || x := (x+y)-((x+y)-y)} =
{x := x+y || y := x || x := y} =
{y := x || x := y}
```

KeY automatically deletes overwritten (unnecessary) updates

Demo

`Examples/lect11/swap.key`

Parallel Updates Cont'd

Example

```
({x := x+y}{y := x-y}){x := x-y} =
{x := x+y || y := (x+y)-y}{x := x-y} =
{x := x+y || y := (x+y)-y || x := (x+y)-((x+y)-y)} =
{x := x+y || y := x || x := y} =
{y := x || x := y}
```

KeY automatically deletes overwritten (unnecessary) updates

Demo

`Examples/lect11/swap.key`

Parallel updates to store intermediate state of symbolic computation

A Warning

First-order rules that substitute arbitrary terms

$$\exists\text{-right}\ \frac{\Gamma \Longrightarrow [x/t']\,\phi,\ \exists\, T\, x;\ \phi, \Delta}{\Gamma \Longrightarrow \exists\, T\, x;\ \phi, \Delta} \qquad \forall\text{-left}\ \frac{\Gamma, \forall\, T\, x;\ \phi,\ [x/t']\,\phi \Longrightarrow \Delta}{\Gamma, \forall\, T\, x;\ \phi \Longrightarrow \Delta}$$

$$\text{applyEq}\ \frac{\Gamma, t \doteq t', [t/t']\,\psi \Longrightarrow [t/t']\,\phi, \Delta}{\Gamma, t \doteq t', \psi \Longrightarrow \phi, \Delta}$$

t, t' must be rigid, because all occurrences must have the same value

Example

$$\frac{\Gamma, \mathtt{i} \doteq 0 \ \text{->}\ \langle \mathtt{i++} \rangle \mathtt{i} \doteq 0 \Longrightarrow \Delta}{\Gamma, \forall\, T\, x;\ (x \doteq 0 \ \text{->}\ \langle \mathtt{i++} \rangle x \doteq 0) \Longrightarrow \Delta}$$

Logically valid formula would result in unsatisfiable antecedent!
KeY prohibits unsound substitutions

Literature for this Lecture

Essential

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)