COMP2111 Week 1
Term 1, 2024
Calculating with Logic II

Monday Recap

1. Logic is *really simple*: it's like elementary set theory, but with formulae instead of pictures.

Monday Recap

1. Logic is *really simple*: it's like elementary set theory, but with formulae instead of pictures.
2. Um, actually.. set theory is not that simple.
   Can you draw a picture of the empty set?
   What is $\{x | x \notin x\}$?
3. ...so maybe logic is not as simple as it looks either.

Monday recap

Indeed, logic is not as simple as it looks.

Why do we care?

Monday recap

Indeed, logic is not as simple as it looks.

Why do we care?

Because we need a logic that works, to make sure that our programs work.

Today

D12–D13, then D.1–D.3 from the IFM book.

- State
- Bound and free variables
- Substitution

All these relate to *quantifiers*.

$\forall$ for all
$\exists$ exists

State

States interpret terms (and by extension, formulae)

$n \mapsto 1$	integer
$b \mapsto \texttt{true}$	boolean
$x \mapsto 3.5$	real
$x \mapsto \{1, 2, 3\}$	set (of integers)
$x \mapsto [1, 2, 3]$	sequence (of integers)

A state is a mapping from *variables* to *values*.

Free and bound variables

$$
\begin{array}{l}
\{ \quad \text{int } y = 2; \\
\quad\; \mathit{printf}(\text{"}\%d, \%d.\text{"}, x, y); \\
\}
\end{array}
$$

Free and bound variables

```
{   int y = 2;
    printf("%d, %d.", x, y);
}
```

x, y are variables.

Free and bound variables

```
{   int y = 2;
    printf("%d, %d.", x, y);
}
```

x, y are variables.

x is a free variable.

Free and bound variables

```
{   int y = 2;
    printf("%d, %d.", x, y);
}
```

x, y are variables.

x is a free variable.

y is a bound variable, whose scope is delimited by { }

Free and bound variables

```
{   int y = 2;                    ⟵ binding occurence of y
    printf("%d, %d.", x, y);      ⟵ bound occurence of y
}
```

x, y are variables.

x is a free variable.

y is a bound variable, whose scope is delimited by { }

Free and bound variables

```
{   int y = 2;                          ⟵ binding occurence of y
    printf("%d, %d.", x, y);            ⟵ bound occurence of y
}
```

x, y are variables.

x is a free variable.

y is a bound variable, whose scope is delimited by $\{\ \}$

The program depends on x.

We could rename y to anything (except x) without changing the program.

Free and bound variables

$$\int_0^1 x^y \, dy$$

x is a free variable.

y is a bound variable with scope x^y.

The ~~program~~ integral depends on x.

We could rename y to anything (except x) without changing the integral.

Free and bound variables

$$(\forall y \cdot y^2 \geq x)$$

x is a free variable.

y is a bound variable with scope delimited by ( ).

The ~~program~~ formula depends on x.

We could rename y to anything (except x) without changing the formula.

Substitution

$$\text{E.60} \quad (\forall x \cdot A) \Rrightarrow A[x := E]$$
$$\text{E.61} \quad A[x := E] \Rrightarrow (\exists x \cdot A)$$

$A[x := E]$ replaces all *free* occurences of x in A with E.

Watch out for variable capture!

Quantifier examples

(live)

Checking programs with logic

E.60 $(\forall x \cdot A) \Rrightarrow A[x := E]$
E.61 $A[x := E] \Rrightarrow (\exists x \cdot A)$

Idea

$//A[x := E]$	*If this is true initially...*
$x = E$	*...and we execute this line of code...*
$//A$	*...then this is true afterwards.*

Notice that execution flows forwards, but the substitution flows backwards!

Checking programs with logic

$$\text{E.60} \quad (\forall x \cdot A) \Rrightarrow A[x := E]$$
$$\text{E.61} \quad A[x := E] \Rrightarrow (\exists x \cdot A)$$

Idea

$//x = 5$	*If this is true initially...*
$x = x + 2$	*...and we execute this line of code...*
$//x = 7$	*...then this is true afterwards.*

Notice that execution flows forwards, but the substitution flows backwards!

Checking programs with logic

Example (Swapping two variables)

$// \ ?$	What must be true here...
$t = x$	
$x = y$	
$y = t$	
$//x = Y \wedge y = X$	...to make this true here?

Use the substitution principle from the previous slide repeatedly to check whether this swapping procedure works!

Checking programs with logic

Similar reasoning principles apply to other programming constructs!

Idea

$//A$
if B then
  $//A \wedge B$
  $\langle code \rangle$
  $//C$
else
  $//A \wedge \neg B$
  $\langle code \rangle$
  $//C$
$//C$

Checking programs with logic

Here's while:

Idea

$//A$
while B
  $//A \wedge B$
  $\langle code \rangle$
  $//A$
$//A \wedge \neg B$

A here is a *loop invariant*.

Checking programs with logic

Example: exponentiation

(live)

That was *Hoare logic*. You can use it to synthesise programs from specifications, or to check that your programs are correct.

Main idea: annotate your code with comments that say *what's true here*. Follow the comments backwards using substitution to check them.

Hoare Logic combines *first-order predicate logic* and *program semantics*. We will study their foundations—that is, why they work—later in the course.