System Modelling and Design
COMP2111

+ tutors:
Zhuo (Zoey) Chen
Raphael Douglas Giles
Charran Kethees

Johannes Åman Pohjola
System Modelling and Design
COMP2111

+ tutors:
Zhuo (Zoey) Chen
Raphael Douglas Giles
Charran Kethees

Johannes Åman Pohjola
System Modelling and Design
COMP2111

Credit for the material also goes to:
Paul Hunter,
Christine Rizkallah,
Liam O’Connor,
and Carroll Morgan

+ tutors:
Zhuo (Zoey) Chen
Raphael Douglas Giles
Charran Kethees

Johannes Åman Pohjola
We’ll learn to
model systems in a way that’s unambiguous and mathematically precise.

We’ll be able to
say what it means for a system to satisfy its specification,
and prove that it does so.

We’ll need
a substantial toolbox of discrete math and formal logic.

Don’t worry; we’ll teach it, not assume it.
Non-examples

Transmission Control Protocol
Functional Specification

3.9. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls

OPEN
SEND
RECEIVE
CLOSE
ABORT
STATUS

Arriving Segments

SEGMENT ARRIVES

Timeouts

USER TIMEOUT
RETRANSMISSION TIMEOUT
TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term “signal” means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive “error: connection not open”.

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, etc., is modulo $2^{32}$ the size of the sequence number space. Also note that “<” means less than or equal to (modulo $2^{32}$).
Non-examples

This RFC is a specification in English.

Natural language specs tend to have:
• Ambiguities
• Room for interpretation
• Important details in the writer’s head absent from actual text.

Transmission Control Protocol
Functional Specification

3.9. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls
  OPEN
  SEND
  RECEIVE
  CLOSE
  ABORT
  STATUS

Arriving Segments
  SEGMENT ARRIVES

Timeouts
  USER TIMEOUT
  RETRANSMISSION TIMEOUT
  TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, etc., is modulo $2^{32}$ the size of the sequence number space. Also note that "$<=$" means less than or equal to (modulo $2^{32}$).
Non-examples

DrawPolygons
- polygonsArray(Polygon)
+ paint(g:Graphics):void
+ main(args: String[]):void

Polygon
# center: Point
- Polygon(p: Point)
+ Polygon(x: int, y: int)
+ paint(g: Graphics): void

Point

Rectangle
+ Rectangle(x: int, y: int)

Square
+ Square(x: int, y: int)

Circle
+ Circle(x: int, y: int)
Non-examples

This UML diagram describes the structure of the system, not its behaviour.
Resources

Course website: http://www.cse.unsw.edu.au/~cs2111
• General announcements
• Lecture slides
• Assignment instructions
• ...

Ed forum: https://edstem.org/au/courses/7621
• Class discussion, announcements
• E-mail cs2111@cse.unsw.edu.au if you haven’t been invited!

Moodle: https://moodle.telt.unsw.edu.au/
• Lecture recordings
• Weekly quizzes
Examination

- Weekly quizzes: 15 credits total
  - After the lectures of every week (except W6 and W10).
  - Will appear on Moodle.
  - Deadline: Monday 10AM (before start of next week’s lectures)

- Three assignments (individual or pair, written): $10 \times 3 = 30$ credits

- Final exam (online, format TBA): 55 credits
Introduction to “Formal” Logic

Start here

This is a (draft) textbook for COMP6721 (In-)Formal Methods by Carroll Morgan

It’s on the course website.

D The arithmetic of conditions
D.1 Introduction and rationale ..................................................... 207
D.2 Why is my program correct? ...................................................... 207
D.3 How do I write my program in the first place? ............................ 209
D.4 Calculating with conditions .................................................... 210
D.5 Simple calculations in logic ..................................................... 212
D.6 Terms ............................................................................ 213
D.7 Simple formulae ................................................................. 214
D.8 Propositions, and propositional formulae .................................... 215
D.9 Operator precedence ............................................................. 215
D.10 Calculation with logical formulae .......................................... 217
D.11 Exercises on propositions .................................................... 218
D.12 Quantifiers .................................................................... 221
D.13 Exercises on quantifiers ....................................................... 222
D.14 (General) formulae ............................................................... 223

E Some helpful logical identities
E.1 Some basic propositional rules .................................................... 225
E.2 Some basic predicate rules ....................................................... 228
E.3 Exercises on rules for logic ...................................................... 232
E.4 Epilogue on notation and terminology ...................................... 232
Introduction to “Formal” Logic

- element
- set
- membership
- intersection

$x \in S$
$x \notin S$
$x \in A \cap B$

defined?
Introduction to “Formal” Logic

$x \in A \cap B$ if and only if $x \in A$ and $x \in B$
Introduction to “Formal” Logic

intersection  \( A \cap B \)

union  \( A \cup B \)

subset  \( A \subseteq B \)

Q: Subset is not like others in an important way. How?

\[ x \in A \cup B \text{ if and only if } x \in A \quad \text{or} \quad x \in B \]

\[ A \subseteq B \text{ if and only if } x \in A \quad \text{implies} \quad x \in B \]
Introduction to “Formal” Logic

Where is $A \cap B$ now?
Introduction to “Formal” Logic

Let’s prove $A \subseteq A \cup B$
Why so pedantic?

\[ \{ y \mid y \subseteq x \} \]

_____________________

\[ \{ y \mid y \in x \} \]

_____________________

\[ \{ y \mid y \subseteq x \} \]

_____________________

\[ \{ y \mid y \in x \} \]

_____________________

\[ \{ y \mid y \subseteq x \} \]

_____________________

\[ \{ y \mid y \in x \} \]

_____________________

\[ \{ y \mid y \subseteq x \} \]

_____________________

\[ \{ y \mid y \in x \} \]

_____________________

\[ \{ y \mid y \subseteq x \} \]

_____________________

\[ \{ y \mid y \in x \} \]
Why so pedantic?

\[ x \in x \]

Does it make sense to write?

Is it ever true?

\[ \{ x \mid x \in x \} \]
Why so pedantic?

\[ \{ x \mid x \not\in x \} \]

\[ y = \{ x \mid x \not\in x \} \]

\[ y \in y \Rightarrow y \not\in y \]

\[ y \not\in y \Rightarrow y \in y \]

“There is just one point where I have encountered a difficulty”
- Bertrand Russell

Q: Why does this matter?
The language of logic

- Propositional formulae
- Simple formulae
- Terms

Layers!
The language of logic

A *term* is either
(a) a variable, or
(b) a constant symbol, or
(c) a function symbol applied to the correct number of other terms.

A function’s number of arguments is its *arity*. 

The language of logic

A *term* is either
(a) a variable, or
(b) a constant symbol, or
(c) a function symbol applied to the correct number of other terms.

| variables | x, y, ... |
| constants | ... |
| functions | ... |

A function's number of arguments is its *arity*. 
The language of logic

- Propositional formulae
- Simple formulae
- Terms

Terms have values

Layers!

Terms

- $0$
- $x + 1$
- $x$
- $\sin\left(\frac{x}{2}\right)$
- $x +$

Not terms

- $x y$
- $x +$
- $x +$

Terms have values
The language of logic

A *simple formula* is a predicate symbol applied to the correct number of (term) arguments.

Layers!

- $t < u$
- $t = u$
- $t \geq u$
- $\text{even}(t)$
- $t \in u$
- $false$

$t, u$ are terms

These can be True or False
The language of logic

- Propositional formulae
- Simple formulae
- Terms

Simple formulae

Layers!

false
$x \in y$
$\pi \in \mathbb{IR}$
$1 \leq a/2$
$x + 5$
$\ x \in \left( y \in z \right)$

Not formulae
A *propositional formula* is either
(a) a simple formula
(b) a *propositional connective* applied to
the right number of arguments.

\[
\begin{array}{l}
\wedge \text{ and} \\
\lor \text{ or} \\
\neg \text{ not} \\
\rightarrow \text{ implies} \\
\leftrightarrow \text{ iff}
\end{array}
\]
The language of logic

- Propositional formulae
- Simple formulae
- Terms

Layers!

\[ x \in y \land x \geq 2 \]
\[ n! = n \leftrightarrow (n = 1 \lor n = 3) \]

Q: Is this formula True?

*false*
## Truth tables

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ∧ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ∨ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ↔ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
The language of logic: summary

<table>
<thead>
<tr>
<th>Propositional formulae</th>
<th>Propositional connectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple formulae</td>
<td>Predicate symbols</td>
</tr>
<tr>
<td>Terms</td>
<td>Constants, functions, variables</td>
</tr>
</tbody>
</table>

Truth tables define what the propositional connectives mean.

Q: Did the layered, systematic approach help against Russell's paradox?
Calculating with logic

D The arithmetic of conditions
D.1 Introduction and rationale ........................................... 207
D.2 Why is my program correct? ........................................... 207
D.3 How do I write my program in the first place? .................. 209
D.4 Calculating with conditions ........................................... 210
D.5 Simple calculations in logic ........................................... 212
D.6 Terms ................................................................. 213
D.7 Simple formulae ......................................................... 214
D.8 Propositions, and propositional formulae ......................... 215
D.9 Operator precedence ................................................... 215
D.10 Calculation with logical formulae ................................... 217
D.11 Exercises on propositions ............................................. 218
D.12 Quantifiers ............................................................. 221
D.13 Exercises on quantifiers .............................................. 222
D.14 (General) formulae ..................................................... 223

E Some helpful logical identities
E.1 Some basic propositional rules ....................................... 225
E.2 Some basic predicate rules ............................................ 228
E.3 Exercises on rules for logic ........................................... 232
E.4 Epilogue on notation and terminology ............................... 232
Calculating with logic

Propositional formulae

Are like the conditions in if-then-else, while

Terms

are like the RHS of assignment statements

We can \textit{calculate} with logic as a thinking tool for programming,

...just as we can use mathematical calculation as a thinking tool for physics.
Calculating with logic

```python
if l <= m < h:
    ...
else:
    ... #what’s true here?
```

(let’s calculate)
Calculating with logic

Are these programs the same?

(let’s calculate)