Lecture 1
Why (formal) Verification?

- Safety-critical applications:
  - Air-traffic controllers
  - Medical equipment
  - Cars

  Bugs are unacceptable!

- Bugs found in later stages of design are expensive, e.g. Intel’s Pentium bug in floating-point division

- Testing does not provide full coverage
What are we doing about it?

Testing - build it, try it on a some cases, hope it works all cases
What should we be doing?

Formal analysis and verification

\[
\begin{align*}
\mathbf{r}_{\text{com}} &= \frac{\sum m_i \mathbf{r}_i}{\sum m_i} = (\bar{x}, \bar{y}, \bar{z}) \\
\bar{x} &= \frac{\sum m_i x_i}{\sum m_i} \\
\bar{y} &= \frac{\sum m_i y_i}{\sum m_i} \\
\bar{z} &= \frac{\sum m_i z_i}{\sum m_i} \\
\mathbf{r}_{\text{com}} &= \frac{1}{m} \int r dm = \frac{1}{V} \int r dV \\
\mathbf{r}_{\text{mg}} &= \frac{\sum w_i \mathbf{r}_i}{\sum w_i} = (\bar{x}, \bar{y}, \bar{z}) \\
\bar{x} &= \frac{\sum w_i x_i}{\sum w_i} \\
\bar{y} &= \frac{\sum w_i y_i}{\sum w_i} \\
\bar{z} &= \frac{\sum w_i z_i}{\sum w_i} \\
\mathbf{r}_{\text{mg}} &= \frac{1}{W} \int r dw = \frac{1}{V} \int r dV \\
\sum \mathbf{F} = 0 &\iff \begin{cases} \\
\sum F_x = \sum F_x \\
\sum F_y = \sum F_y \\
\sum F_z = \sum F_z \\
\end{cases} \\
\sum \tau = 0 &\iff \begin{cases} \\
\sum \tau_x = \sum \tau_x \\
\sum \tau_y = \sum \tau_y \\
\sum \tau_z = \sum \tau_z \\
\end{cases}
\end{align*}
\]
The goal of the course: Formal Verification

Given

- A (model of) hardware or software system and
- a formal specification

does the system satisfy the specification?

Not decidable!
Formal Verification

Solutions:

• “Program correctness”: Provide non-automated verification methods

• “Automatic verification / Model Checking”: restrict the problem to a decidable one:
  - Finite-state reactive systems
  - Propositional temporal logics
Specifications

• Should be given for a system by the designer, developer, programmer, user

• Examples:
  - Does the program always terminate?
  - Does the program compute correctly multiplication of its inputs?
Specifications

• Additional examples:
  - When we press a sequence of buttons on the control panel of an airplane / microwave - do we get the desired result?
  - When we deposit money - does it get to our account?
  - Can a user access data only if he has the appropriate authorization?
Verification tools

Are developed and used in

• **Hardware industry:** Intel, IBM, Cadence, Mellanox, ...

• **Software industry:** Microsoft, NASA, Amazon, Facebook...

• **Universities**
Part 1 of the course

Program Correctness

- Non-automated
- Verifies program with possibly infinite number of states
- Refers to the programs as input-output transformation
Ingredients for Formal Verification

1. Specification language
   • With formal semantics

2. Programming language
   • with formal semantics

3. Proof rules
   • For proving “Program P has the property \( \varphi \)”
Requirements from the proof rules

• **Soundness of the rules:** if we were able to prove correctness of program $P$ w.r.t. specification $\varphi$ using the proof rules, then $P$ is correct w.r.t. $\varphi$

• **Completeness of the rules:** if $P$ is correct w.r.t. specification $\varphi$, then our proof rules can prove it
We handle:

- **Deterministic programs**
  - Exactly one computation for every input
  - At most one output for each input

- **Properties**
  - Partial correctness
  - Termination
  - Total Correctness
Some notations

• Program variables: \( \bar{x} = (x_1, \ldots, x_n) \)

• A state of the program \( \sigma \) is a function from program variables to their domains

• The set of program states is defined by:
  \[ D_1 \times \ldots \times D_n \cup \{ \perp \} \]
  Where \( D_i \) is the domain of variable \( x_i \)
Program states: Examples

• A program with integer variable $x$, Boolean variable $b$
  – States: $(5, F), (-17, T)$

• Elevator on 3 floors:
  $\text{elev\_at } \in \{1, 2, 3\}$
  $\text{on\_floor1, on\_floor2, on\_floor3}$: Boolean
  $\text{in\_elev1, in\_elev2, in\_elev3}$: Boolean
  $\text{direction } \in \{\text{up, down}\}$, $\text{door } \in \{\text{open, close}\}$
  – State: $(2, F, T, T, T, T, F, \text{up}, \text{close})$
Defining the Specification

Specification is a pair \(<q_1(\overline{x}), q_2(\overline{x})>\) where:

- \(q_1(\overline{x})\), \(q_2(\overline{x})\) are first order formulas over program variables

- \(q_1(\overline{x})\) describes a condition holding before the execution of the program

- \(q_2(\overline{x})\) describes a condition holding at the end of the execution of the program
Examples

Specification example

• \( < (x \geq 0 \land y > 0) \land (z = x/y \land z \geq 0) > \)

A program with \( x \in \mathbb{N}, y \in \mathbb{R}, b \in \{T,F\} \)

States: \((5, 5.0, T), (7, 3.111, F)\)

\( q_1(x, y, b) = x > 0 \land b \)

\( q_2(x, y, b) = x+y > 0 \land \neg b \)
Computations of Programs

- $\pi(P, \sigma)$ denotes a computation of program $P$ from state $\sigma$
- $\pi(P, \sigma)$ is a finite $(\sigma_1, ..., \sigma_k)$ or infinite $(\sigma_1, \sigma_2, ...) \text{ sequence of states where:}$
  - $\sigma_1 = \sigma$
  - $\sigma_{i+1}$ is a result of applying an action from the program on $\sigma_i$
- This definition is not a full definition
More notations

• $\bot$ - bottom: the undefined value

• $\text{val}(\pi)$ denotes the final state of computation $\pi$ (if exists)
  
  - $\text{val}(\pi) = \sigma_k$ if $\pi = (\sigma_1, \ldots, \sigma_k)$
  
  - $\text{val}(\pi) = \bot$ if $\pi = (\sigma_1, \sigma_2, \ldots)$
    
    • $\pi$ is an infinite computation

• $\sigma \models q(\overline{x})$ if $q(\overline{x})$ is true when free variables in $q$ are replaced with matching values in $\sigma$
• Important remark:
  \( \bot \neq q(\bar{x}) \) for every \( q(\bar{x}) \) (even \( \bot \neq \text{true} \))

• Example of formulas and their meaning:
  \( q(y) = \forall x (y | x \lor 2 | x) \)  where \( x, y \) are naturals
  – For a state \( \sigma (x) = 1, \sigma (y) = 2, \sigma (z) = 1 \)
    \( \sigma \models q(y) \) since \( \forall x (2 | x \lor 2 | x) \) is true
Partial Correctness

• A program $P$ is partially correct with respect to specification $<q_1(x), q_2(x)> \iff$ for every computation $\pi$ of $P$ from an initial point of $P$, and for every state $\sigma_0$:

  if

  – the computation starts from state $\sigma_0$ which satisfies $q_1(x)$ and
  – the computation terminates

  then

  – $q_2(x)$ holds at the end of the computation
Partial Correctness

• For every computation $\pi$ and every state $\sigma_0$:

$$(\sigma_0 \models q_1(x) \text{ and } \text{val}(\pi(P, \sigma_0)) \neq \bot) \Rightarrow \text{val}(\pi(P, \sigma_0)) \models q_2(x)$$

• Notation: $\{q_1\}P\{q_2\}$
Total Correctness

- A program $P$ is **totally correct** with respect to specification $<q_1(\bar{x}), q_2(\bar{x})>$ iff for every computation $\pi$ of $P$ from an initial point of $P$, and for every state $\sigma_0$:

  - the computation starts from state $\sigma_0$ which satisfies $q_1(\bar{x})$
  - the computation terminates, and
  - $q_2(\bar{x})$ holds at the end of the computation
Total Correctness

• For every computation $\pi$ and every state $\sigma_0$:

\[ \sigma_0 \models q_1(\bar{x}) \Rightarrow \text{val}(\pi(P, \sigma_0)) \neq \bot \text{ and } \text{val}(\pi(P, \sigma_0)) \models q_2(\bar{x}) \]

• Notation: $<q_1>P<q_2>$
How do we write the specification:

“P terminates if the initial state satisfies $q_1$”
Separation Lemma

- For every program $P$ and specification $<q_1,q_2>$:

$$\models <q_1>P<q_2>$$

if and only if

$$\models \{q_1\}P\{q_2\} \text{ and } \models <q_1>P<true>$$
Examples

• Which programs satisfy \{true\}P\{false\}?

• Which programs satisfy \langle true \rangle P \langle false \rangle ?
Logical Variables in Specifications

Example 1:
Specify a program with a single variable $x$ whose value at the end of the computation is twice its value at the beginning
Logical Variables in Specifications

Solution: add fresh variables which are
– not part of the program and therefore
– their value does not change during the execution of the program

These variables are called logical variables

Convention: We use logical variable $X$ to preserve the value of variable $x$
Logical Variables in Specifications

Example 2:
Program which returns in variable $z$ the multiplication of variables $x$ and $y$

Convention:
Assertions $q_1, q_2$ are now defined over $\bar{x}$ that includes program variables as well as logical variables
End of lecture 1