Introduction to Software Specifications

Stefan D. Bruda

CS 310, Winter 2025

CS 310: Intro to Software Specifications

- Coordinates:
  - **Course Web page: http://cs.ubishops.ca/home/cs310**
  - Instructor: Stefan Bruda (http://bruda.ca, stefan@bruda.ca, Johnson 114B, ext. 2374)
  - Office hours?
- Textbook (required): R. D. Tennent, Specifying Software: A Hands-On Introduction, Cambridge University Press, 2002
- Two subjects:
  - Formal languages
  - Program specification and reasoning about program correctness

Formal Languages

- Describe problems in a formal way so that we can reason about them (syntax)
- Describe how a problem can be solved computationally (semantics)
- Example of the impact of theory to practice
  - Lexical and parsing stages of compiler construction
  - Use of regular expressions in text editors
  - State-charts in object-oriented modeling
  - Circuit design
  - DNA and protein sequence matching
  - Behaviour of reactive systems
- Also useful in answering fundamental questions such as whether there exist tasks/problems that cannot be solved algorithmically and, if yes, which tasks are algorithmically solvable and which are not

Formal Languages (cont'd)

- Formal languages = the mathematics of strings of symbols
- Alphabet = a finite, nonempty set of elements (symbols, tokens, characters)
- String (word) over an alphabet Σ = finite sequence of symbols of Σ (a string in Σ^*)
  - ε = the empty string
  - Examples: cat, dog, mouse, Fluffy, xzrbstuph, 0011011100, -36.557
- Language over Σ = set of strings over Σ
  - Examples:
    - The English language
    - $\{w : w \text{ is the name of a cat}\}$
    - $\{n \in \{0,1\}^* : \exists x, y, z \in \mathbb{N} : x^n + y^n = z^n\}$
    - $\{g \in (\mathbb{N} \times \mathbb{N})^* : \text{the graph } g \text{ has a Hamiltonian path}\}$
  - Finite (or not), finite representation (or not), mathematical representation
  - We want to analyze how problems are solved computationally, so we study formal (systematic, computational) descriptions

Program Specification

- A program transforms input values to output values in a particular way, so a specification describes transformations from input values to output values
- Therefore a specification consists in the following parts:
  - What the input will be
  - What the output should be
  - What is the environment in which the specification/program should work
- Input and output refer to things that can be observed: input and output variables and constraints on the variables
- The formalization of a specification consists in the following:
  - Declarative interface: static properties of the identifiers
  - Pre-conditions: assertion on input values that the program will be given
  - Post-condition: assertion on output values, possibly in relation to input values

Program Specification (cont'd)

- A specification is a contract: the software designer agrees to establish the post-condition if the program is started in a way that satisfies the pre-condition
  - If the program is run in a context not covered by the pre-condition, it can run in any way without "breaking" the contract
- In this course we will develop logic-based techniques to verify correctness of small programs (algorithms)
  - That is, the goal is to prove that a program does what a specification says it should do
  - We use logical formulas (assertions) as comments in programs
    - We assert that the formula should be true when flow of control reaches it
- Such techniques are generally too time consuming to be used directly with large software systems
  - In these cases formal methods are used instead
  - Combination of program specification and formal languages!