Introduction

Lecture Notes by Dhananjay Raju
draju@cs.utexas.edu

Overview to the course

The syntax of a program is a sequence of characters that adhere to some grammer. At the highest level, we can consider the english language. The basic parts of a sentence are the subject and the verb. The subject is usually a noun–a word (or phrase) that names a person, place, or thing. The predicate (or verb) usually follows the subject and identifies an action or a state of being. This is a rule. Similarly programming languages can be thought of combining legal words (example... rule for variables in c) with suitable connectives. Eg. $y = p + q$ is a legal sentence. We can interpret y, p, q over $\mathbb{Z}$. Programs can be expressed as mathematical objects. Programs have syntax and semantics. Every program has a finite set of instructions(instruction set... remember computer architecture or complier design or microprocessors) this is the syntax. The semantics is the meaning. We define it mathematically, because it makes "meaning" unambiguous, concise, and most importantly it makes it possible to develop mathematical proofs about properties of interest.

Historically, there are 3 ways of representing the meaning

- Operational Semantics defines meaning in terms of execution on an abstract machine.
- Denotational Semantics defines meaning in terms of mathematical objects such as functions.
- Axiomatics Semantics defines meaning in terms of logical formulas satisified before and after execution.

Hoare logic is an example of an axiomatic semantic. Central to Hoare logic is the Hoare triple $\{P\}c\{Q\}$. Here P and Q are conditions and c is a command. $\{P\}$ is the precondition. $\{Q\}$ is the post-condition. Very simply it means when condtion P is met before executing command c then condition Q is met after executing the command c.For example $\{x + 1 = 43\}y := x + 1\{y = 43\}$ is a valid triple. P and Q are formulas over prepositional logic.

Pnueli introduced temporal logic to deal with checking of correctness or properties in concurrent programs. It is a very convinient way of representing specification of programs. Over the usual atomic propositions and boolean connectives present in prepositional logic, it also has temporal modalities. We can use temporal logic to reason about prepositions qualified in time. Example we can represent statements like "I am always hungry" G, "I will be hungry sometime in future" F and "I will be hungry in the next hour" X. Here G, F, X

are the temporal modalities. This is very useful in representing properties of computer programs. $G\alpha$, where α is a formula which says that some variable is in bounds, is a kind of property that we want to verify. This is a typical safety property. Mutual exclusion is another such property. The Future modality F is useful in expressing properties which will eventually come true e.g. The program will eventually terminate another example is if a program keeps requesting for a resource, it eventually gets it.

These logics are useful because they are sound and complete. Verification normally is generating a verification condition ϕ from a source code and annotated loop invariants, then checking the validiy of this condition in the specification. Typically programs always have bugs. Debugging can prove the presence of bugs, but cannot prove their absence. Hence the interest in verification of programs.