CS571

- Notes 08
- Axiomatic semantics

Problems with operational semantics

- Too coarse (big-step)
- Syntax is part of the rules of inference, not separated
- Non-uniform language – logic (for rules) and functions (for model of program state)
- Denotational uses pure sets and functions
- Axiomatic use pure logic

Program state in axiomatic semantics

- Implicit model through constraints on values of variables
- Constraint = assertion
- E.g. $x = 1$ instead of $[x \mapsto 1]$
- Also inequalities: $x > 1$
- Assertions can annotate a program by putting constraints on program state

Annotations

- $\{a_{\text{initial}}\} P \{a_{\text{final}}\}$
- $P$ is the program code
- $a$’s are assertions
- Often $a_{\text{initial}}$ is empty (always true)
- $a_{\text{final}}$ is the specification of the program, i.e. what the program is intended to do
Example final assertion
- Imagine $x_0, x_1, \ldots, x_{n-1}$ are elements of an array, and total is a variable that starts out at 0.
- The assertion $\text{total} = \sum_i x_i$ is the specification.
- The initial assertion might be $\{x_0 = 5 \land x_1 = 12 \land \cdots\}$.
- Can we relate these with the actual program code?

Joining syntax and semantics
- We can say that if $a_{\text{initial}}$ is true before the program starts, and $a_{\text{final}}$ is true afterwards, then P must connect them.
- In logic: $a_{\text{initial}} \land \text{terminates}(P) \Rightarrow a_{\text{final}}$.
- How do we know that the implication is true?
- We can try to prove it, using the rules of logic.

Proofs and problems
- To prove the implication, we need to break down the program into its constituent commands and write assertions about each intermediate program state.
- Axioms will help us deal with the simple command types (assignment, sequence, conditional, loop).
- Even if we know the implication to be true, we may not be able to find a proof.
- If we make the wrong assertions, or write the code badly, then it may be unprovable.

Requirements for the method
- A language for assertions (logic + arithmetic).
- A systematic way to break down the program (abstract syntax).
- A proof technique using axioms and rules of inference (sometimes called natural deduction).