CS 6110 Lecture 18 Friday March 2, 2012
CS 6116 Lecture 6

Programs as proofs and Programming Logics

What is a key lesson from Prof Gries' lecture?
What is a lesson from the history of programming languages?

Programming languages are the key interface to computers when we want to engage them to help us accomplish algorithmic tasks.

The technology behind implementations them, compilation and semantic definition, is autocatalytic, e.g. it can be applied to itself. This is a powerful technology paradigm.

This technology can be used to provide computer support for the problem solving process that Prof. Gries so elegantly illustrated in Lecture 17.

He showed us that careful specifications, even informal ones and partial ones, can help us think. This is a very important lesson.*

A key step in providing computer assistance to writing good and correct code is to design programming logics and implement them.

* I'll tell a story about Lamport's lecture (at Lada 2012) on specifying programming tasks – he writes 10 times as much to himself as he publishes. (I recall 30K lines to 300K lines.)

CS6110 Lecture 18 Friday March 2, 2012
CS6116 Lecture 6

IMP Logic Rules for Asserted Programs

Loop Termination

Let $\bar{x}$ be the n-tuple of variables in the body of the loop.

$\{ \exists y{:}\mathbb{N}.\ T(y, \bar{x}) \}$ suppose n_0 is the value

while bexp **do**

$\{$ arbitrary $y{:}\mathbb{N}.\ T(y, \bar{x}) \}$

$\leftarrow$ alternative $\{ \neg T(0, \bar{x}) \}$

body

$\{ T(y-1/y) \}$

$\{ T(0, \bar{x}) \Rightarrow \neg \text{bexp} \}$

od

$\{ \neg \text{bexp} \}$

This is equivalent to requiring $T(0, \bar{x}) \Rightarrow \neg \text{bexp}$ at the beginning which is equivalent to $\text{bexp} \Rightarrow \neg T(0, \bar{x})$.

If the body terminates, then small step semantics provides a sequence of states $s_0, s_1, \ldots, s_n$ in which the value y decreases. So the loop executes for at most n_0 steps.

Theorem If the body of the loop terminates on any state s_0 satisfying the predicate $T(n_0, \bar{x})$ and bexp and the assertions in the loop are true, then the loop terminates in at most n_0 steps.

CS 6110 Lecture 18
CS 6116 Lecture 6

Turing Completeness of Programming Languages and Kleene Normal Form

Turing machines have been accepted as the "ground definition" of mechanically computable functions, and the Church/Turing thesis is that every effectively computable function (humanly computable) is Turing Computable. Results in the 1930's, 40's, 50's and 60's established a wide range of equivalent definitions of the Turing computable partial functions. The basic data type was typically strings of symbols, Σ^*, for Σ a finite alphabet. This is equivalent to using natural numbers $\mathbb{N}$ which can code Σ^* into $\mathbb{N}$.

Kleene provided one of the most commonly used Turing complete formalisms based on $\mathbb{N}$, the μ-recursive functions. These are the primitive recursively defined functions augmented by the least number operator μx. We write

$$f(x) = \mu y.(h(y,x) = 0)$$

where μy is the least natural such that $h(y,x) = 0$. His famous normal form theorem is that every partial recursive function can be defined as

$$\varphi_i(x) \equiv U(\mu y\, T(i,x,y))$$

where y codes a terminating computation of $\varphi_i(x)$ and U picks out the value of the computation. The functions T and U are primitive recursive. (Kleene 1M, 1952 p. 288)