Lecture 13

Midterm reminder Friday March 6 – 2 weeks from today

Topics

1. The PL models so far:
   \( \lambda \)-calculus (Universal, i.e Turing complete)
   Combinators (Universal)
   Primitive recursion and CoqPL (Subrecursive languages)
   Partial recursive functions (Turing complete)
   \( \mu \)-operator
   Herbrand/Gödel recursive

2. Functional PL mechanisms so far:
   Howe’s equality
   Environments, closures
   Continuations (tail recursions)
   (Path to compilation)
   Yet to come – defunctionalization, Abstract State Machines
   “Compiler” Theory – the logic of compilers

3. Relating models.
   All can be compiled to Turing machines or other universal machine models: RAM, RASP, \( G_3 \) machine, etc. If they can simulate a Turing machine, they are universal. We should also require oracles. This has become key in Nuprl’s event logic. We also call these models Turing complete.

   Primitive and Partial Recursive vs. \( \lambda \)-terms
   Recursive functions in mathematics, refining this idea led to general recursive functions or Herbrand/Gödel, also captured in CoqPL. Primitive recursion is natural, but see PS2 on Fibonacci function.
   Primitive recursive functions are tail recursive.

5. The lambda calculus theory requires a meta language as we see in Vincent’s lectures.

---

\(^1\) Turing was careful to allow oracles, that is part of some models explicitly. What about for the \( \lambda \)-calculus? Add a function term \( f \) with oracular reductions. Have non-deterministic combinator been studied? Yes! \( \alpha + \beta \).
6. Steps toward the BRICS evaluators (in SML)

The style of recursive function classes.
The first glimpse of the notion of a universal class of computable functions came from the study of recursive functions in number theory.

Recursive Functions by Rozsa Peter, 1967 from her work starting in 1932.

Herbrand 1932
Herbrand Gödel 1934 \{ General recursive functions

Kleene 1936, two key papers relating general recursive functions to his partial recursive functions defined in lecture.

The \( \lambda \)-calculus ideas came in 1932 as well. Revisited in 1940.

Turing 1937 – Turing machines [Gödel said: this definition is \textit{absolute}.]
Turing 1937 – PhD compilation of \( \lambda \)-calculus to Turing machines.

Markov algorithms 1947 – other late models of computersl RAMS, RASP, etc.

The recursive function style is mathematical or \textit{denotational}, based on classes of functions, methods of recursion.

- Primitive (singly recursive)
- Double recursive, course-of-values, simultaneous
- \( n \)-recursive \( h(p + 1, n + 1, a) = h(p, h(p + 1, n, a), a) \) See Ackermann example of a doubly recursive function.
- Higher-type recursion on \textit{functionals} will be discarded later. They are used in Gödel’s system \( T \) – 1958
- Recursive Number theory – Goodstein 1957 shows how to formulate results in number theory, such as the fundamental theorem of arithmetic, using recursive functions. He later did elements of analysis this way (Recursive Analysis).

What is the essence of this method?
Closing a class of functions under operations. No “syntax”, just math operations as functions and an inductive class of functions using \textit{functionals}, i.e. functions taking functions as inputs and outputs.

\{ Examine the definition of primitive recursion.
Imagine how to extend to get Ackermann

Let’s show that primitive recursive functions are \textit{tail recursive}.
Consider the pattern of primitive recursion.

\[
\begin{align*}
  f(0, y) &= a(y) \\
  f(S(n), y) &= h(n, y, f(n, y))
\end{align*}
\]
Consider this computation:

\[
\begin{align*}
  f(3, 5) &= h(2, 5, f(2, 5)) \\
  f(2, 5) &= h(1, 5, f(1, 5)) \\
  f(1, 5) &= h(0, 5, f(0, 5)) \\
  f(0, 5) &= a(5)
\end{align*}
\]

These are stack frames

We can see how destructive updates can collapse the stack frames – using “mutable variables”, i.e. imperative programs. We study their semantics later in the course.

\[
\begin{align*}
  i := 0; & \quad f := a(5) \\
  \text{while } (i < n) \text{ do} & \quad f := h(i, 5, f) \\
  & \quad i := i + 1 \\
  \text{od} & \quad \{ f(i, 5) = h(i - 1, 5, f(i - 1, 5)) \} \\
  \{ i = n \} & \quad \{ f(n, 5) = h(n - 1, y, f(n - 1, y)) \}
\end{align*}
\]

The while loop does tail recursion.

Exercise: Write the while loop as a recursive function. (To be assigned in PS3.)

Steps toward understanding the BRICS (Basic Research in CS, Aarhus, Denmark) paper on the course web page: *A Functional Correspondence Between Evaluators and Abstract Machines*. Ager, Bienacki, Danvy, Midtgaard, 2003.

They write a sequence of evaluators in SML. Eval0 produces Krivine’s Abstract State machine. It uses \(\lambda\)-terms coded with deBruijn \(\lambda\)-terms. The bound variables are given by their off-set numbers from the binding \(\lambda\). For example:

\[
\begin{align*}
  \lambda x.\lambda y. x &\text{ is } \lambda \lambda 2 \\
  \text{The } S\text{-combinator } \lambda x\lambda y\lambda z. xz(yz) &\text{ is} \\
  &\quad \lambda \lambda \lambda 31(21) \\
  \lambda z. (\lambda y. y(\lambda x. x))(\lambda x. zx) &\text{ is} \\
  &\quad \lambda (\lambda 1)(\lambda 1)(\lambda 21)
\end{align*}
\]

You only need to know that bound variables are numbers. We won’t use this notation. Substitution is nasty, need to renumber.