Advanced Progamming Languages	**Lecture 18**
CS 6110 Spring 2015	**Wed. March 4, 2015**

Lecture 18

Topics

1. The typed λ-calculus continued

   Curry vs. Church typing
   Strong Normalization
   A wild conjecture

2. Statman's Theorem – ties to Theory A

3. Partial Types

   The conjecture: λ-terms with partial types are SN.

4. Midterm exam review, Krvine ASM, and higher-order primitive recursion

---

1. **The typed λ-calculus continued**

   Curry style typing – based on untyped terms (Nuprl) $\lambda(x.x) \in \alpha \to \alpha$

   Church style typing – terms come with types attached (Coq) $\lambda x^{\alpha}.x \in \alpha \to \alpha$

   Comparisons:

   Lisp is Curry style, can reason about untyped computation.

   The ML family of languages is Church style, but they allow polymorphism, e.g. $\lambda x^{\alpha}.\lambda y^{\beta}.x$

   Key Theorem for either style: *Strong Normalization*

   Thompson section 2.7, p.45

   Induction on type structures Def. 2.22

   We might discuss the *Tait method* for a deeper theorem. Another approach to this topic is to use logical relations. We will not cover this, but see Harper's textbook.

   Also see Thompson's account of primitive recursion in section 2.9.

   Note: Thompson's account of general recursion, section 2.10, does not agree with Kleene's. We stick with Kleene's account.

2. **Statman's Theorem**

   We will not explore this topic, but Statman proved that β-equality on the typed lambda terms is decidable *but not elementary*! So the decision method is of exponential complexity.

3. **Partial types**

   We have mentioned the *partial types*, written $\bar{\alpha}$. It is curious that SN might hold for these since self-application and thus the **Y** combinator is prohibited.

4. **Midterm review continued**

   The Abstract State Machine for $eval_c$ – see the Lecture 17 supplement by Dr. Rahli, section 6, page 3.

   ASM loop(term,env,stack)

(i)	$loop(x, e, lst)$	$=$	let $< t, e' >= e(x)$
			in $loop(t, e', lst)$
(ii)	$loop(\underline{\lambda}(x.b), e, lst)$	$=$	match lst with
	remove closure from		$[\,] \Rightarrow < \underline{\lambda}(x.b), e >$
	stack and evaluate		$\mid\ < a, e' > :: tl \Rightarrow$
	b in new env		$loop(b, e[x \rightarrow < a, e' >], tl)$
(iii)	$loop(\underline{ap}(f; a), e, lst)$	$=$	$loop(f, e, < a, e > :: lst)$
	add closure $<a, env>$		
	to stack, evaluate f		

   Optional exercise: Write a Lisp style dynamic scoping evaluator.

   Note, Dr. Rahli lets $loop(t, e, [\,])$ evaluate to (t, e) to simplify the typing, e.g. to return a V closure.

   Exercise: Evaluate $ap(\lambda(x.\lambda(y.x)); \mathbf{I})$ using loop.

   Observation. We can make the type of V closure more informative by writing
   V closure $= v : Value \times Env(v)$, in detail $v : Val \times \{e : Env \mid domain(e) = freevars(v)\}$

   This is called a *dependent type*.