Advanced Progamming Languages **Lecture 28**
CS 6110 Spring 2015 **Mon. April 6, 2015**

Lecture 28

Topics

1. Remainder of the course – a dozen standard lectures left because of Charter Day on Monday April 27, and a lecture for final exam review from Abhishek and Mark Bickford.
2. Some lectures are to suggest to you that there is an opportunity for enduring deep contributions to programming languages and type theory, steps toward the "next big thing". What might it be? It's good to look at history and ties between PL and other areas – Theory A, AI (automated reasoning, NLP, machine learning (MUSE), cyber physical systems (HACMS), distributed computing (CRASH)).

   Historical Notes – already mentioned and yet to come:

   Lisp – 1962 higher-order functions as objects, reflection and self modification
   Algol 60 – types
   Algol W, Pascal – types
   PL1
   Java, C^{++}, $C^{\#}$
   The ML Family:
   Classic ML
   Standard ML
   OCaml
   Haskell
   $F^{\#}$
   ——Speculation——
   Coq PL, F*, dependent types
   EventML (functional processes)
3. The Blum Size Theorem
4. Start imperative programming languages – Winskel

The Blum Size Theorem Continued

Recall that we frame these results in terms of *acceptable indexings* of the partial recursive functions on $\mathbb{N}$. We let φ_i be such an indexing.

Rogers shows that two such indexings, φ_i and ψ_i are recursively isomorphic, i.e. there is a *1 - 1 onto* recursive function $f : \mathbb{N} \to \mathbb{N}$ such that $\varphi_{f(i)} = \psi_i$.

Blum uses this result to show that two size measures are "closely related" – his Proposition 1 and Propostion 2.

His main theorem on size has already been stated. We now look more closely at the proof.

Blum Size Theorem (Theorem 1 p.259)

0. Let $| \, |$ be any size measure.
1. Let g be any (*total*) recursive function with *unbounded range* which enumerates indexes in an acceptable indexing – think of enumerating the smallest of the primitive recursive programs, but it could be a subset of the partial recursive functions.
2. Let f be any total recursive function used to measure the difference in magnitude, think of fast growing monotonic functions such as $k \cdot x$ or $k_1 \cdot x^{x^2}$, etc.

   Then we can effectively find numbers i, j uniformly in f, g (i.e. constructively, given the programs f and g), such that:
3. $\varphi_i = \varphi_{g(j)}$ and
4. $f|i| < |g(j)|$ (φ_i is "smaller than" $g(j)$)

Exercise: Show that we can enumerate the smallest primitive recursive programs, for length measure. Can it be done for any measure?

Blum Size Theorem Proof

We define a procedure for effectively finding the two integers i and j give the indexes (programs) for total recursive functions f and g.

Define a partial recursive function by the following program:

```
begin
With natural number inputs $x$, $y$ and given programs $f$ and $g$
   1. compute $f|y|$.
   2. then while $|g(j)| \leq f|y|$ starting at $j = 0$, increase $j$.
      Eventually we will find the least $j$ such that $f|y| < |g(j)|$, call it $j_0$.
      output $\varphi_{g(j_0)}(x)$.
end
```

These instructions define $\varphi_z(x, y)$, and we can determine the index z.

The recursion theoreom of Kleene in the strong form guarantees that we can find an integer i such that $\varphi_i(x) = \varphi_z(x, i)$

First we find i constructively (uniformly), as above. Then compute $f|i|$ and compute $|g(0)|,\ |g(1)|, ...$ until $f|y| < |g(j)|$ at j_0, then we have $\varphi_i(x) = \varphi_z(x, i) = \varphi_{g(j_0)}(x)$. ■

Consider the following imperative code for a bst function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ with size measuere $|\ |$.

```
bst($x : \mathbb{N},\ y : \mathbb{N}) : \mathbb{N} =$
**begin**
    $j : \mathbb{N}$
    $j := 0$
    **while** $|g(j)| \leq f(|y|)$ **do**
        $j := j + 1$
    **od**
        $f(|y|) < |g(j)|$
    $j_0 := j$
    **output** $(\varphi_{g(j_0)}(x))$
**end** bst
```

Call this program $\varphi_z(x, y)$.

Why is there a z index for this program?

By the recursion theorem, we can find a natural number i_0 such that $\varphi_{i_0}(x) = \varphi_z(x, i_0)$

We have: $\varphi_{i_0}(x) = \varphi_z(x, i_0) = \varphi_{g(j_0)}(x)$.