Computer Science 513

A Computable Total Function That is Not Primitive Recursive

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #13

Goals for Today

Goal for Today:

- Use of ***diagonalization*** to provide a computable total function that is not primitive recursive.
- This is tricky because it is not clear how to ***enumerate*** the primitive recursive functions (in one variable) — and this will be a significant part of the argument.

Reference: *Computability, Complexity and Languages*, Section 4.9.

Expectations

- Students will not be asked to recall or explain the details of the long, complicated proof that is provided in this lecture.
- The result, itself, was surprising when it was first discovered that there are computable total functions are not primitive recursive — and students will be expected to remember (and understand) this.

Outline of Proof

1. An ***alternative definition*** of the set of primitive recursive functions will be given and proved to be equivalent to the original one — that is, it defines the same set of functions.
2. This will be used to provide another ***alternative definition*** of the set of ***unary primitive recursive functions***.
3. This will provide an ***enumeration*** of the set of unary primitive recursive functions — such the value of the i^{th} of these functions at value n can be shown to be a ***computable*** function of i and n, by an application of the ***Recursion Theorem***.
4. This will make it possible for the ***diagonalization method*** to demonstrate the existence of a total computable function that is, provably, not primitive recursive.

Another Version of Primitive Recursion

Consider the following ***new version of primitive recursion***. For functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, the function h obtained from f and g using this new version of primitive recursion is the function $h : \mathbb{N}^2 \to N$ satisfying the following properties.

- $h(x, 0) = f(x)$ for all $x \in \mathbb{N}$.
- $h(x, t + 1) = g(h(x, t))$ for all $x, t \in \mathbb{N}$.

These conditions allow the value of $h(x, y)$ to be determined, using the functions f and g, for all $x, y \in \mathbb{N}$.

Another Version of Primitive Recursion

Example: Suppose f is the projection function $u_1^1 : \mathbb{N} \to \mathbb{N}$, and that g is the successor function $s : \mathbb{N} \to \mathbb{N}$. Then the function $h : \mathbb{N}^2 \to \mathbb{N}$ obtained from f and g is the function $h : \mathbb{N}^2 \to \mathbb{N}$ such that

- $h(x, 0) = f(x) = u_1^1(x) = x$ for all $x \in \mathbb{N}$, and
- $h(x, t + 1)) = g(h(x, t)) = s(h(x, t)) = h(x, t) + 1$ for all $x, t \in \mathbb{N}$.

It is easy to show, by induction on y, that $h(x, y) = x + y$ for all $x, y \in \mathbb{N}$. Thus h is the ***addition*** function in this case.

Alternative Characterization

Theorem #1: Let $f : \mathbb{N}^n \to \mathbb{N}$. Then f is primitive recursive if and only if it can be obtained from the (enriched) set of initial functions —

(a) The *successor function* $s : \mathbb{N} \to \mathbb{N}$ such that $s(x) = x + 1$ for all $x \in \mathbb{N}$.

(b) The *nullify function* $n : \mathbb{N} \to \mathbb{N}$ such that $n(x) = 0$ for all $x \in \mathbb{N}$.

(c) The *left and right inverses* $\ell, r : \mathbb{N} \to \mathbb{N}$ of the pairing function.

(d) The *pairing function* mapping $x, y \in \mathbb{N}$ to $\langle x, y \rangle$.

Alternative Characterization

(e) The *projection functions* $u_i^n : \mathbb{N}^n \to \mathbb{N}$ such that, for $1 \leq i \leq n$ and all $x_1, x_2, \ldots, x_n \in \mathbb{N}$,

$$u_i^n(x_1, x_2, \ldots, x_n) = x_i.$$

— using the operations ***composition*** and the ***new version of primitive recursion*** that has now been defined.

What This Says: By slightly expanding the set of "initial functions" we may replace the original, more general, form of "primitive recursion" with the new, more restrictive form of "primitive recursion" that has now been introduced — without changing the set of "primitive recursive functions" that are defined.

Alternative Characterization

The following is reasonably easy to prove and is used in the proof of Theorem #1.

Lemma #2). Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ be primitive recursive functions, and let $h : \mathbb{N}^2 \to \mathbb{N}$ be the function obtained from f and g using the new version of primitive recursion. Then h is primitive recursive.

How This is Proved: The function h is constructed from f, g, and some of the (standard) initial functions using composition and the original form of primitive recursion. The proof is included in the supplemental document for this lecture.

Alternative Characterization

Lemma #3: Let $f : \mathbb{N}^k \to \mathbb{N}$ for a positive integer k. If f can be constructed from the "expanded set" of initial functions, listed in parts (a)–(e) of Theorem #1, above, using a finite number of applications of composition and the new version of primitive recursion, then f is primitive recursive.

How This is Proved: By induction on the length of a "derivation" of f that includes introduction of functions in the expanded set of initial functions and applications of composition and the new form of primitive recursion — applying Lemma #2, above, when functions obtained using the new version of primitive recursion are considered.

Alternative Characterization

Claims similar to Lemma #2 can be established and used to prove a converse for Lemma #3.

Lemma #4: Let $c \in \mathbb{N}$, let $g : \mathbb{N}^2 \to \mathbb{N}$, and let $h : N \to \mathbb{N}$ such that h can be constructed from c and g using the first form of primitive recursion. Then h can be constructed from g and the extended set of initial functions in parts (a)–(e) of Theorem #1 using a finite number of applications of composition and the new version of primitive recursion.

Reference to Proof: The proof of this claim (which uses the pair function, and its inverses, along with the new version of primitive recursion to replace a use of the first (standard) form of primitive recursion) is included in the supplemental document for this lecture.

Alternative Characterization

Lemma #5: Let k be a positive integer, let $f : \mathbb{N}^k \to N$, let $g : \mathbb{N}^{k+2} \to \mathbb{N}$, and let $h : \mathbb{N}^{k+1} \to \mathbb{N}$ be the function obtained from f and g using the second (standard) form of primitive recursion. Then h can be obtained from f, g, and the extended set of initial functions (given in parts (a)–(e) of Theorem #1) using a finite number of applications of composition and the new version of primitive recursion.

How This is Proved: The result — whose proof is also included in the supplemental document — is established using techniques used to establish the previous one, in an induction on k.

Alternative Characterization

Lemma #6: Let $f : \mathbb{N}^k \to \mathbb{N}$. If f is primitive recursive then f can be constructed from the "expanded set" of initial functions, listed in parts (a)–(e), of Theorem #1, above, using a finite number of applications of composition and the new version of primitive recursion.

How To Prove This: This can be proved by induction on the length of a derivation of f (from the initial functions, using composition and both (standard) forms of primitive recursion), so that it is structurally similar to the proof of Lemma #3, with Lemmas #4 and #5 applied when primitive recursion is being used. The proof is left as an ***exercise.***

Theorem #1 is now a straightforward consequence of Lemma #3 and Lemma #6.

Unary Primitive Recursive Functions

- Recall that — for functions whose inputs and outputs are natural numbers (with only one number returned as output) — a function is a ***unary*** function if it only has one input — so that it is a partial or total function $f : \mathbb{N} \to \mathbb{N}$.
- The next goal is is to establish an ***alternative definition*** for the set of unary primitive recursive functions.
- This will allow us to obtain a ***enumeration*** of these functions that can be used to establish the result that we want.
- The "alternative definition" is given by the following definition. The theorem that follows it (and its proof) establishes that this is another — equivalent — way to define the set of unary primitive recursive functions.

Unary Primitive Recursive Functions

Definition: Let PR be the smallest family of functions $f : \mathbb{N} \to \mathbb{N}$ that includes

(a) the successor function $s : \mathbb{N} \to \mathbb{N}$ such that $s(x) = x + 1$ for all $x \in \mathbb{N}$,

(b) the nullify function $n : \mathbb{N} \to \mathbb{N}$ such that $n(x) = 0$ for all $x \in \mathbb{N}$,

(c) the inverses $\ell, r : \mathbb{N} \to \mathbb{N}$, of the pairing function,

and such that the following functions are also in PR whenever $f, g \in$ PR:

Unary Primitive Recursive Functions

(d) The function $f \circ g : \mathbb{N} \to \mathbb{N}$ such that $f \circ g(x) = f(g(x))$ for all $x \in \mathbb{N}$;

(e) The function $h : \mathbb{N} \to \mathbb{N}$ such that $h(x) = \langle f(x), g(x) \rangle$ for all $x \in \mathbb{N}$,

(f) The function $h : \mathbb{N} \to \mathbb{N}$ such that $h(0) = 0$ and, for $t \in \mathbb{N}$,

$$h(t+1) = \begin{cases} f(t/2) & \text{if } t \text{ is even,} \\ g(h((t+1)/2)) & \text{if } t \text{ is odd.} \end{cases}$$

Unary Primitive Recursive Functions

Theorem #7: PR is the set of unary primitive recursive functions.

About the Proof: A variety of more minor results (”lemmas”) must be stated and proved before a proof of Theorem 7 can be given — but the next major goal is to prove this theorem.

The supplemental document for this lecture includes proofs of all of the lemmas that follow.

Unary Primitive Recursive Functions

Lemma #8: Suppose that $f, g : \mathbb{N} \to \mathbb{N}$ are primitive recursive. If $h : \mathbb{N} \to \mathbb{N}$ is the function such that

$$h(0) = 0$$

and, for all $t \in \mathbb{N}$,

$$h(t+1) = \begin{cases} f(t/2) & \text{if } t \text{ is even,} \\ g(h((t+1)/2)) & \text{if } t \text{ is odd,} \end{cases}$$

then h is primitive recursive as well.

Reference To Proof: See the supplemental document for a proof of this result.

Unary Primitive Recursive Functions

Lemma #9: Every function in PR is a unary primitive recursive function.

How This is Proved: This is proved by induction on the number of times that application of parts (d), (e) and (f) of the above definition of PR are used, when showing that a function is in PR — using Lemma #8 to establish what is needed when the operation given in part (f) of the definition is applied.

Unary Primitive Recursive Functions

Lemma #10: The function $u_1^1 : \mathbb{N} \to \mathbb{N}$ is in PR.

About the Proof: While a proof of this lemma is included the supplemental document, it is simple enough that it could also have been left as a straightforward exercise.

Unary Primitive Recursive Functions

Definition: A function $g : \mathbb{N}^n \to \mathbb{N}$ is ***satisfactory*** if the function $\widehat{g} : \mathbb{N} \to \mathbb{N}$ such that, for all $t \in \mathbb{N}$,

$$\widehat{g}(t) = g(h_1(t), h_2(t), \ldots, h_n(t))$$

is an element of PR whenever $h_1, h_2, \ldots, h_n \in$ PR.

Lemma #11 Let $f : \mathbb{N} \to \mathbb{N}$ be a *unary* primitive recursive function. Then f is satisfactory *if and only if* $f \in$ PR.

About the Proof: The proof — which is included in the supplemental document — is also simple enough that it could be left as an exercise (so you should try to prove it yourself, if you have time for this).

Unary Primitive Recursive Functions

Lemma #12: Let $h : \mathbb{N} \to \mathbb{N}$ be primitive recursive. Then h is satisfactory.

About the Proof: This can be established by application of the "alternative characterization" of the primitive recursive functions, using induction on the number of applications of ***composition*** and the ***special form of primitive recursion*** used to obtain h. As for the above results, a proof of this can be found in the supplemental document for this lecture.

Unary Primitive Recursive Functions

Lemma #13: If f is a unary primitive recursive function then $f \in \mathrm{PR}$.

Proof. This follows immediately from Lemmas #11 and #12. □

Theorem 7 now follows directly from Lemmas #9 and #13. □

Enumerating the Unary Primitive Recursive Functions

- We are now ready to produce an ***enumeration*** for the unary primitive recursive functions.
- This is actually an enumeration of the ways to obtain such functions using Theorem #7 — so each unary primitive recursive function will appear more than once in this listing. Indeed (since composition with the identity function will produce the same function, farther along in the list) each function will appear infinitely often.

Enumerating the Unary Primitive Recursive Functions

Consider a function $\psi : \mathbb{N} \to$ PR that is defined as follows.

- $\psi(0)$ is the successor function $s : \mathbb{N} \to \mathbb{N}$
- $\psi(1)$ is the nullify function $n : \mathbb{N} \to \mathbb{N}$
- $\psi(2)$ is the left inverse of the pairing function $\ell : \mathbb{N} \to \mathbb{N}$
- $\psi(3)$ is the right inverse of the pairing function $r : \mathbb{N} \to \mathbb{N}$
- If $n = 3k + 4$ for $k \in \mathbb{N}$ then $\psi(n) = \psi(\ell(k)) \circ \psi(r(k))$
- If $n = 3k + 5$ for $k \in \mathbb{N}$ then $\psi(n) = \langle \psi(\ell(k)), \psi(r(k)) \rangle$
- If $n = 3k + 6$ for $k \in \mathbb{N}$ then $\psi(n)$ is the function $h : \mathbb{N} \to \mathbb{N}$ such that $h(0) = 0$ and, for $t \in \mathbb{N}$,

$$h(t+1) = \begin{cases} \psi(\ell(k))(t/2) & \text{if } t \text{ is even,} \\ \psi(r(k))(h((t+1)/2)) & \text{if } t \text{ is odd.} \end{cases}$$

Enumerating the Unary Primitive Recursive Functions

- Theorem #7 implies that every unary primitive recursive function appears at least once in the listing

$$\psi(0), \psi(1), \psi(2), \psi(3), \ldots$$

- Indeed, as previously noted, every unary primitive recursive function appears infinitely often in the above list.

A Total Function That is Computable But Not Primitive Recursive

- Next consider the function $\xi : \mathbb{N}^2 \to \mathbb{N}$ such that, for all $x, n \in \mathbb{N}$

$$\xi(x, n) = \psi(n)(x)$$

for $\psi : \mathbb{N} \to$ PR as above, as well as the function $\rho : \mathbb{N}^2 \to \mathbb{N}$ such that, for all $x, n \in \mathbb{N}$,

$$\rho(x, n) = \xi(x, n) + 1 = \psi(n)(x) + 1.$$

A Total Function That is Computable But Not Primitive Recursive

Lemma #14: The above function $\rho : \mathbb{N}^2 \to \mathbb{N}$ is a computable function.

Proof: Consider the function $g : \mathbb{N}^3 \to \mathbb{N}$ such that, for all $z, x \in \mathbb{N}$,

- $g(z, x, 0) = \psi(0)(x) = s(x) = x + 1$,
- $g(z, x, 1) = \psi(1)(x) = n(x) = 0$,
- $g(z, x, 2) = \psi(2)(x) = \ell(x)$,
- $g(z, x, 3) = \psi(3)(x) = r(x)$,
- If $n = 3k + 4$ for $k \in \mathbb{N}$ then

$$g(z, x, n) = \Phi_z^{(2)}(\phi_z^{(2)}(x, r(k)), \ell(k)),$$

A Total Function That is Computable But Not Primitive Recursive

- If $n = 3k + 5$ for $k \in \mathbb{N}$ then

$$g(z, x, n) = \langle \Phi_z^{(2)}(x, \ell(k)), \Phi_z^{(2)}(x, r(k)) \rangle,$$

- If $n = 3k + 6$ for $k \in \mathbb{N}$ then $g(z, 0, n) = 0$ and, for all $t \in \mathbb{N}$,

$$g(z, t+1, n) = \begin{cases} \Phi_z^{(2)}(t/2, \ell(k)) & \text{if } t \text{ is even,} \\ \Phi_z^{(2)}(\Phi_z^{(2)}((t+1)/2, n), r(k)) & \text{if } t \text{ is odd.} \end{cases}$$

The ***Universality Theorem*** (and a little bit of work) can be used to show that g is a computable partial function.

A Total Function That is Computable But Not Primitive Recursive

- Recall that, by the ***Recursion Theorem***, there exists a number $e \in \mathbb{N}$ such that, for all $x, n \in \mathbb{N}$,

$$\Phi_e^{(2)}(x, n) = \Phi^{(2)}(x, n, e) = g(e, x, n).$$

  Let $h : \mathbb{N}^2 \to \mathbb{N} = \Phi_e^{(2)}(x, n) = g(e, x, n)$ — a computable partial or total function of x and n, since e is a constant.

- For all $x \in \mathbb{N}$, $h(x, 0) = g(e, x, 0) = s(x) = \xi(x, 0)$.
- For all $x \in \mathbb{N}$, $h(x, 1) = g(e, x, 1) = n(x) = \xi(x, 1)$.
- For all $x \in \mathbb{N}$, $h(x, 2) = g(e, x, 2) = \ell(x) = \xi(x, 2)$.
- For all $x \in \mathbb{N}$, $h(x, 3) = g(e, x, 3) = r(x) = \xi(x, 3)$.

A Total Function That is Computable But Not Primitive Recursive

- If $n = 3k + 4$ for $k \in \mathbb{N}$ then

$$\begin{aligned}\xi(x, n) &= \psi(n)(x) \\ &= \psi(\ell(k))(\psi(r(k))(x)) \\ &= \psi(\ell(k))(\xi(x, r(k)) \\ &= \xi(\xi(x, r(k)), \ell(k))\end{aligned}$$

while

$$\begin{aligned}h(x, n) &= \Phi_e^{(2)}(x, n) = g(e, x, n) \\ &= \Phi_e^{(2)}(\Phi_e^{(2)}(x, r(k)), \ell(k)) \\ &= \Phi_e^{(2)}(h(x, r(k)), \ell(k)) \\ &= h(h(x, r(k)), \ell(k)).\end{aligned}$$

A Total Function That is Computable But Not Primitive Recursive

- If $n = 3k + 5$ for $k \in \mathbb{N}$ then

$$\begin{aligned}\xi(x, n) &= \psi(n)(x) \\ &= \langle \psi(\ell(k))(x), \psi(r(k))(x) \rangle \\ &= \langle \xi(x, \ell(k)), \xi(x, r(k)) \rangle\end{aligned}$$

while

$$\begin{aligned}h(x, n) &= \Phi_e^{(2)}(x, n) = g(e, x, n) \\ &= \langle \Phi_e^{(2)}(x, \ell(k)), \Phi_e^{(2)}(x, r(k)) \rangle \\ &= \langle h(x, \ell(k)), h(x, r(k)) \rangle.\end{aligned}$$

A Total Function That is Computable But Not Primitive Recursive

- If $n = 3k + 6$ for $k \in \mathbb{N}$ then $\xi(x, n) = \psi(n)(x)$, so that $\xi(0, n) = \psi(n)(0) = 0$, while if $t \in \mathbb{N}$ then

$$\begin{aligned}
\xi(t+1, n) &= \psi(n)(t+1) \\
&= \begin{cases} \psi(\ell(k))(t/2) & \text{if } t \text{ is even,} \\ \psi(r(k))(\psi(n)((t+1)/2)) & \text{if } t \text{ is odd,} \end{cases} \\
&= \begin{cases} \xi(t/2, \ell(k)) & \text{if } t \text{ is even,} \\ \xi(\xi((t+1)/2, n), r(k)) & \text{if } t \text{ is odd.} \end{cases}
\end{aligned}$$

A Total Function That is Computable But Not Primitive Recursive

- On the other hand, $h(0,n) = \Phi_e^{(2)}(0,n) = g(e,0,n) = 0$ and, for $t \in \mathbb{N}$,

$$h(t+1,n) = \Phi_e^{(2)}(t+1,n) = g(e,t+1,n)$$

$$= \begin{cases} \Phi_e^{(2)}(t/2, \ell(k)) & \text{if } t \text{ is even,} \\ \Phi_e^{(2)}(\Phi_e^{(2)}((t+1)/2,n), r(k)) & \text{if } t \text{ is odd,} \end{cases}$$

$$= \begin{cases} h(t/2, \ell(k)) & \text{if } t \text{ is even,} \\ h(h((t+1)/2,n), r(k)) & \text{if } t \text{ is odd.} \end{cases}$$

A Total Function That is Computable But Not Primitive Recursive

- It is now easy to prove — by induction on $x + n$ (using the strong form of mathematical induction) that

$$\xi(x, n) = h(x, n) \quad \text{for all } x, n \in \mathbb{N},$$

  so that ξ is a computable total function, since h is computable.

- Finally, since

$$\rho(x, n) = \xi(x, n) + 1$$

  for all $x, n \in \mathbb{N}$, ϕ is a computable total function as well, as claimed. □

A Total Function That is Computable But Not Primitive Recursive

Theorem #15: ρ is a computable total function that is not primitive recursive.

Proof: Since ρ is a computable total function, by Lemma #14, it suffices to prove that ρ is not primitive recursive.

This will be proved by contradiction. Suppose that ρ *is* primitive recursive.

- The function $\tau : \mathbb{N} \to \mathbb{N}$ such that, for all $x \in \mathbb{N}$,

$$\tau(x) = \rho(x, x)$$

  is certainly primitive recursive too. Since τ is also a unary function it follows (again, by Theorem #7) that $\tau \in$ PR and, therefore, $\tau = \psi(n)$ for some number $n \in \mathbb{N}$.

A Total Function That is Computable But Not Primitive Recursive

- However, it now follows that

$$\begin{aligned}
\tau(n) &= \rho(n,n) \\
&= \xi(n,n)+1 && \text{(by the definition of } \rho\text{)} \\
&= \psi(n)(n)+1 && \text{(by the definition of } \xi\text{)} \\
&= \tau(n)+1 && \text{(since } \tau = \psi(n)\text{).}
\end{aligned}$$

- This ***contradiction*** establishes that ρ is not primitive recursive, after all.

Thus ρ is a computable total function that is not primitive recursive. □

Ackerman Function

The ***Ackerman function*** is the function $A : \mathbb{N}^2 \to \mathbb{N}$ such that, for $m, n \in \mathbb{N}$,

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

- One can prove, by a "nested induction" (first on m, then on $m+n$) that $A(m,n)$ is defined for all $m, n \in \mathbb{N}$.
- This is actually a later, *simplified* version of the function considered by Ackerman — which had three arguments, instead of two.

Ackerman Function

- The ***Recursion Theorem*** can be used to prove that the Ackerman function is a ***computable*** total function.
- While the proof is rather long, it can be shown that Ackerman's function is not primitive recursive — essentially, because it ***grows too quickly***!
- Thus this is another example of a computable total function that is not primitive recursive.