Computer Science 513

Oracle Computations

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #14

Goals for Today

Goals for Today:

- Introduction to ***oracles***
- Definition of the class of ***G-computable functions*** when $G : \mathbb{N} \to \mathbb{N}$ is a ***total function***, and investigation of properties of this class of functions
- Definition of the class of ***f-computable functions*** when $f : \mathbb{N}^n \to \mathbb{N}$ is a ***total function*** such that $n \geq 2$, and investigation of properties of this class of functions
- "Relativization of Universality:" Results and proofs about universal computation relative to oracles

Reference: *Computability, Complexity and Languages*, Chapter 8, Sections 1 and 2. Notation from that reference is (at least, sometimes) being followed, here.

Oracles: An Informal Introduction

Several recent textbooks, that could be used for CPSC 351 (or 313) or 413, begin a description of ***reductions*** with a highly informal discussion that seem to describe reductions as ***algorithms*** or ***computer programs*** that can call a subroutine or *black box* deciding membership of a number (or string) in a give subset of $\mathbb{N}$ (or language) any number of times — and use the results to form output in a variety of ways.

- The kind of reduction that is *informally* introduced, before that, can be formally defined as an ***oracle reduction*** — the main subject of this lecture.
- ***Many-one*** reductions are different, and are generally only formally defined later on in these textbooks.

$\mathcal{S}$-Programs with an Oracle

Definition: An ***$\mathcal{S}$ program with an oracle*** is a sequence of the following types of instructions:

(a) $V \leftarrow V + 1$ where V is a variable

(b) $V \leftarrow V - 1$ where V is a variable

(c) IF $V \neq 0$ GOTO L, where V is a variable and L is a label, and

(d) $V \leftarrow O(V)$ where V is a variable.

$\mathcal{S}$-Programs with an Oracle

- The first three types of instructions are the same as for regular $\mathcal{S}$-programs — and they have the same effects as before. The sets of labels and variables that can be included in a program (and their roles) are the same as for regular $\mathcal{S}$-programs.
- The last kind of statement is called an ***oracle statement***. These replace the "dummy statements" $V \leftarrow V$ that can also be included in regular $\mathcal{S}$-programs.

$\mathcal{S}$-Programs with an Oracle

- The ***length*** of the program is the number of instructions in this sequence (just as for regular $\mathcal{S}$-programs).
- As for regular $\mathcal{S}$-programs, if an $\mathcal{S}$-program with an oracle has length n then the statements in the program should be assigned numbers $1, 2, \ldots, n$ by order of appearance in the program.

$\mathcal{S}$-Programs with an Oracle: States

- A ***state*** of a program $\mathcal{P}$ is a sequence of equations $V = m$ such that the following properties are satisfied:
  - This includes ***exactly one*** equation $V = m$ for each variable V appearing in $\mathcal{P}$ (and no equations for variables that *do not* appear in $\mathcal{P}$)
  - If the state includes an equation $V = m$ then $m \in \mathbb{N}$ — and one should think of m as being "the current value" of the variable V.
  - The first can be somewhat relaxed: If the equation for a variable V, is not included then it is understood that the equation "$V = 0$" should be added — and equations for input variables. local variables or the output variable that *do not* appear are permissible, as long as they indicate that the input variables have the expected input values, and that the local variables and output variable have value 0.
- This definition is *also* the same as for regular $\mathcal{S}$-programs.

$\mathcal{S}$-Programs with an Oracle: Snapshots

- A ***snapshot*** or ***instantaneous description***, for a program $\mathcal{P}$ with length n, is an ordered pair $\mathcal{S} = (i, \sigma)$ such that
  - $1 \leq i \leq n + 1$. One should think of i as being the number of the statement in $\mathcal{P}$ that is about to be executed *next* — and where $i = n + 1$ if and only if the computation has already ended.
  - σ is a ***state*** of $\mathcal{P}$, as defined on the previous slide.
- Once again, this definition is the same as for regular $\mathcal{S}$-programs.

$\mathcal{S}$-Programs with an Oracle: G-Computations

Now let $G : \mathbb{N} \to \mathbb{N}$ be some partial or total function.

- A ***G-computation*** of a program $\mathcal{P}$ is a sequence $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ of snapshots, where
  - $\mathcal{S}_{i+1}$ is the ***G-successor*** of $\mathcal{S}_i$, for every integer i such that $1 \leq i \leq k - 1$ — as defined more formally on the next slide.

    That is, if $\mathcal{S}_i = (j, \sigma_1)$ and $\mathcal{S}_{i+1} = (k, \sigma_2)$, then $1 \leq j \leq n$, if the program was in state σ_1 and instruction j was executed, then program would be in state σ_2, and k is the number of the instruction that would be executed after this.
  - $\mathcal{S}_k$ is ***terminal***. That is, $\mathcal{S}_k = (n + 1, \sigma)$ for some state σ.

$\mathcal{S}$-Programs with an Oracle: G-Computations

- If $1 \leq i \leq k-1$, $\mathcal{S}_i = (j, \sigma)$, and the j^{th} instruction in the program is one of
  (a) $V \leftarrow V + 1$ for a variable V,
  (b) $V \leftarrow V - 1$ for a variable V, or
  (c) IF V GOTO L for a variable V and label L,

  then the ***G-successor*** of $\mathcal{S}_i$ is the same as the ***successor*** of $\mathcal{S}_i$, as this was defined for regular $\mathcal{S}$-programs.

$\mathcal{S}$-Programs with an Oracle: G-Computations

- On the other hand, if $1 \leq i \leq k$, $\mathcal{S}_i = (j, \sigma_1)$, and the j^{th} instruction in the program is

  $$V \leftarrow O(V)$$

  for a variable V whose current value is m, then
  - If $G(m)$ is defined then $i < k$ and the G-successor of $\mathcal{S}_i$ is $\mathcal{S}_{i+1} = (j+1, \sigma_2)$, where the equation $V = m$ in σ_1 is replaced by $V = G(m)$ in σ_2 and are no other changes are made.
  - If $G(m)$ is undefined then $i = k$. That is, $\mathcal{S}_i$ has no G-successor — but it is considered to be ***nonterminal*** since it represents a computation that will never halt.

    Note that *this* is new: When this was defined for regular computations, all snapshots that did not have successors were terminal because they signified that a computation would end.

$\mathcal{S}$-Programs with an Oracle: G-Computations

- Thus, if $G(m)$ is defined when V has value m then execution of the instruction "$V \leftarrow O(V)$" changes the value of V from m to $G(m)$. If $G(m)$ is undefined then execution of the instruction "$V \leftarrow O(V)$" when V has value m causes the computation to freeze, or ***crash*** — that is, the computation will not really continue but no output will ever be returned.
- A number m that is replaced by $G(m)$ as the result of an execution of the above instruction is called an ***oracle query*** of this G-computation.

$\mathcal{S}$-Programs with an Oracle: G-Computations

- If $\mathcal{P}$ is an $\mathcal{S}$ program with an oracle, n is a positive integer, and $x_1, x_2, \ldots, x_n \in \mathbb{N}$ then — as for regular $\mathcal{S}$-programs, the ***initial snapshot*** for $\mathcal{P}$ and the inputs $x_1, x_2, \ldots, x_n$ is a snapshot $(1, \sigma)$ where σ state including the following equations.
  - $X_i = x_i$ for $1 \leq i \leq n$,
  - $X_i = 0$ for any integer i such that $i \geq n + 1$ and the variable X_i appears in $\mathcal{P}$,
  - $Z_i = 0$ for any integer i such that $i \geq 1$ and the variable Z_i appears in $\mathcal{P}$, and
  - $Y = 0$.
- The ***G-computation*** of program $\mathcal{P}$ and inputs $x_1, x_2, \ldots, x_n$ is the G-computation $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$, of $\mathcal{P}$, that begins with the initial snapshot for $\mathcal{P}$ and the inputs $x_1, x_2, \ldots, x_n$. This is either unique or does not exist, at all.

$\mathcal{S}$-Programs with Oracle G-Computations

- ***Definition:*** For an $\mathcal{S}$-program $\mathcal{P}$ with an oracle for G, and for a positive integer n, the partial or total function

$$\Phi^{(n)}_{\mathcal{P},G} : \mathbb{N}^n \to \mathbb{N}$$

is defined, as follows, for $x_1, x_2, \ldots, x_n \in \mathbb{N}$:

  - If there exists a G-computation of $\mathcal{P}$ and inputs $x_1, x_2, \ldots, x_n$,

$$\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$$

and $\mathcal{S}_k$ is a terminal snapshot, then $\Phi^{(n)}_{\mathcal{P},G}(x_1, x_2, \ldots, x_n)$ is the number ℓ such that the equation "$Y = \ell$" is included in $\mathcal{S}_k$ (or 0 if there is no equation for Y included in $\mathcal{S}_k$). On the other hand, $\Phi^{(n)}_{\mathcal{P},G}(x_1, x_2, \ldots, x_n)$ is undefined if $\mathcal{S}$ is nonterminal.
  - $\Phi^{(n)}_{\mathcal{P},G}(x_1, x_2, \ldots, x_n)$ is undefined if no G-computation for $\mathcal{P}$ and inputs $x_1, x_2, \ldots, x_n$ exists.

G-Computable Functions: Definition

Suppose, now, that $G : \mathbb{N} \to \mathbb{N}$ is a ***total function***.

- ***Definition:*** An $\mathcal{S}$-program $\mathcal{P}$, with an oracle for a total function $G : \mathbb{N} \to \mathbb{N}$, ***G-computes*** a partial or total function $f : \mathbb{N}^n \to \mathbb{N}$ if $f = \Phi^{(n)}_{\mathcal{P},G}$.
- A (partial or total) function $f : \mathbb{N}^k \to \mathbb{N}$ is said to be ***G-computable*** if there is a program $\mathcal{P}$ that G-computes f.

G-Computable Functions and Computable Functions

Theorem #1: If a (partial or total) function $f : \mathbb{N}^n \to \mathbb{N}$ is computable then f is G-computable for ***every*** total function G.

Proof:

- If $f : \mathbb{N}^n \to \mathbb{N}$ is computable then there exists a regular $\mathcal{S}$-program that computes f.
- If unlabelled dummy statements $V \leftarrow V$ are deleted and each labelled dummy statement

$$[L] \quad V \leftarrow V$$

is replaced by a pair of statements

$$[L] \quad V \leftarrow V + 1$$
$$V \leftarrow V - 1$$

then the result is an "$\mathcal{S}$-program with an oracle" that also computes f — as needed to prove the claim. □

G-Computable Functions and Computable Functions

Let $I : \mathbb{N} \to \mathbb{N}$ be the identity function — that is, $I(n) = n$ for all $n \in \mathbb{N}$.

Theorem #2: A computable (partial or total) function $f : \mathbb{N}^n \to \mathbb{N}$ is computable if and only if f is I-computable.

Proof:

- Theorem 1 implies that if f is computable then (since I is a total function) f is I-computable — so it is necessary and sufficient to prove that if f is I-computable then f is computable in order to establish the claim.

G-Computable Functions and Computable Functions

- Let $\mathcal{P}$ be an $\mathcal{S}$-program with an oracle that I-computes f.
- Notice that, since $I(m) = m$ for all $m \in \mathbb{N}$, it suffices to replace each statement

$$V \leftarrow O(V)$$

with the dummy statement

$$V \leftarrow V$$

in order to produce a regular $\mathcal{S}$-program that also computes f.

- It follows that f is computable, as needed to establish the claim. □

G-Computable Functions: *G* is *G*-Computable

Theorem #3: Let G be a total function. Then G is G-computable.

Proof:

- It suffices to notice that G is G-computed by the $\mathcal{S}$-program with an oracle obtained from the following after macro expansion:

$$X_1 \leftarrow O(X_1)$$
$$Y \leftarrow X_1$$

□

A Closure Property for the Set of *G*-Computable Functions

Theorem #4: Let $G : \mathbb{N} \to \mathbb{N}$ be a total function.

(a) Each of the ***initial functions*** (included in the definition of a primitive recursive function) is *G*-computable.

(b) The set of *G*-computable functions is ***closed under composition***. That is, if n and k are positive integers, and $f : \mathbb{N}^k \to \mathbb{N}$ is a *G*-computable partial or total function, and $g_i : \mathbb{N}^n \to \mathbb{N}$ is a *G*-computable partial or total function, for every integer i such that $1 \leq i \leq k$, then the function $h : \mathbb{N}^n \to \mathbb{N}$ such that

$$h(x_1, x_2, \ldots, x_n) = f(g_1(x_2, x_2, \ldots, x_n), \\ g_2(x_1, x_2, \ldots, x_n), \ldots g_k(x_1, x_2, \ldots, x_n)),$$

is also a *G*-computable partial or total function.

A Closure Property for the Set of *G*-Computable Functions

(c) The set of *G*-computable functions is ***closed under the first form of primitive recursion.*** That is, if k is a constant and $g : \mathbb{N}^2 \to \mathbb{N}$ is a *G*-computable partial or total function, then the partial or total function $h : \mathbb{N} \to \mathbb{N}$ such that

$$h(0) = k,$$

and

$$h(t+1) = g(t, h(t))$$

for every integer $t \geq 0$, is also a *G*-computable partial or total function.

A Closure Property for the Set of *G*-Computable Functions

(d) The set of *G*-computable functions is ***closed under the second form of primitive recursion***. That is, if n is a positive integer, $f : \mathbb{N}^n \to \mathbb{N}$ is a *G*-computable partial or total function, and $g : \mathbb{N}^{n+2} \to \mathbb{N}$ is a *G*-computable partial or total function, then the function $h : \mathbb{N}^{n+1} \to \mathbb{N}$ such that, for all $x_1, x_2, \ldots, x_n \in \mathbb{N}$,

$$h(x_1, x_2, \dot{,} x_n, 0) = f(x_1, x_2, \ldots, x_n),$$

and

$$h(x_1, x_2, \ldots, x_n, t+1) = g(t, h(x_1, x_2, \ldots, x_n, t), x_1, x_2, \ldots, x_n)$$

for all $t \in \mathbb{N}$, is also a *G*-computable partial or total function.

A Closure Property for the Set of *G*-Computable Functions

Sketch of Proof: The proof of this is (essentially) identical to the proof that the set of all *computable* partial and total functions satisfies these properties:

- It was shown in Lecture #4 that each of the initial functions is $\mathcal{S}$-computable. Furthermore, each of these functions is computed by an $\mathcal{S}$-program without dummy statements — See "Claim #4", in the preparatory reading for that lecture, and its proof.

  These programs can be easily converted into $\mathcal{S}$-programs with oracles that compute the same (initial) functions: Nothing, at all, about these programs must be changed.

A Closure Property for the Set of *G*-Computable Functions

- Similarly, it is straightforward to modify the proofs of Claims #5 and #6 in the notes for Lecture #4 to establish closure of the set of $\mathcal{S}$-computable partial and total functions under composition and both forms of primitive recursion — without modifying the $\mathcal{S}$-programs in these proofs (which do not include dummy statements) at all.

  These $\mathcal{S}$-programs also easily converted into $\mathcal{S}$-programs with oracles that establish closure of the set of *G*-computable partial and total functions under these operations. Once again, nothing about these programs must be changed. □

G-Computable Functions: Transitivity

Theorem #5: Let $G, H : \mathbb{N} \to \mathbb{N}$ be total functions. If f is G-computable and G is H-computable then f is H-computable.

Sketch of Proof:

- Let $\mathcal{P}$ be an $\mathcal{S}$-program with an oracle that G-computes f.
- Let $\mathcal{Q}$ be an $\mathcal{S}$-program with an oracle that H-computes G.
- If each oracle statement "$V \leftarrow O(V)$" in $\mathcal{P}$ is replaced by a (suitably modified) copy of $\mathcal{Q}$, by macro expansion[1], then the result is an $\mathcal{S}$-program with an oracle that H-computes f — as needed to establish the claim. □

[1] That is, the inserted program does not modify inputs or the output unless it is supposed to, and it does not use local variables or labels that are used anywhere else.

G-Computable Functions when *G* is Computable

Theorem #6: Let *G* be a ***computable total function***. Then a (partial or total) function *f* is computable if and only if *f* is *G*-computable.

Sketch of Proof:

- It follows by Theorem #1 that if *f* is computable then *f* is *G*-computable for every total function $G : \mathbb{N} \to \mathbb{N}$. It is therefore necessary and sufficient to show that if *G* is a computable total function and *f* is *G*-computable then *f* is computable as well.
- Since *f* is *G*-computable there exists an $\mathcal{S}$-program with an oracle, $\mathcal{P}$, such that $\mathcal{P}$ *G*-computes *f*.

G-Computable Functions when *G* is Computable

- Since G is computable, there exists a regular $\mathcal{S}$-program $\mathcal{Q}$ such that $\mathcal{Q}$ computes G.
- If each oracle statement "$V \leftarrow O(V)$" in $\mathcal{P}$ is replaced by a (suitably modified) copy of $\mathcal{Q}$, by macro expansion, then the result is a regular $\mathcal{S}$-program that computes f — as needed to show that f is computable and complete the proof. □

f-Computable Functions: Definition

Definition: Let $f : \mathbb{N}^n \to \mathbb{N}$ be a ***total function*** where $n \geq 2$. Then we say that a function g is ***f-computable*** if g is G-computable, where

$$G(x) = f((x)_1, ((x)_2, \ldots, (x)_n)$$

for all $x \in \mathbb{N}$.

f-Computable Functions: f is f-Computable

Theorem #7: Let $f : \mathbb{N}^n \to \mathbb{N}$ be a total function where $n \geq 2$. Then f is f-computable.

Proof:

- Let $G(x) = f((x)_1, (x)_2, \ldots, (x)_n)$ as above.
- Consider an execution of the $\mathcal{S}$-program with an oracle obtained from the following, by macro expansion, on inputs $x_1, x_2, \ldots, x_n$.

$$Z_1 \leftarrow [X_1, X_2, \ldots, X_n]$$
$$Z_1 \leftarrow O(Z_1)$$
$$Y \leftarrow Z_1$$

f-Computable Functions: f is f-Computable

- After the execution of the first statement, the local variable Z_1 has value

$$[x_1, x_2, \ldots, x_n].$$

- After the execution of the second instruction, the local variable Z_1 has value

$$G([x_1, x_2, \ldots, x_n]) = f(x_1, x_2, \ldots, x_n).$$

f-Computable Functions: f is f-Computable

- After the execution of the final instruction, the output variable, Y, also has value

$$f(x_1, x_2, \ldots, x_n).$$

- Thus this program G-computes f — as needed (along with the previous definition) to conclude that the total function f is f-computable. □

Relativization

- If a result continues to hold when an oracle is added, and the proof is the same as well, then we generally speak of the result (that continues to hold after computation is changed by adding an oracle) as a ***relativized theorem***, and we called the changed proof (which now proves the result that mentions oracles or oracle programs) to be a ***relativized proof***.
- It is also possible to modify a Turing machine in order to produce an ***oracle Turing machine,*** so that results in ***complexity theory*** that relativize can also be considered.

Relativization of Universality

- The goal of this final section is to state and prove versions of some of the results that we have seen already, that apply to *G*-computation instead of (regular) computation.
- In particular, the following will be stated and proved:
  - Relativized Universality Theorem
  - Relativized Step Counter Theorem — A result establishing that a useful kind of "halt" predicate is *G*-Computable
  - Relativized and Strengthened Parameter Theorem

Relativization of Universality

- A “Finiteness Theorem” is also presented and proved. This is only needed later on — but its proof is related to the proof of the “Relativized Universality Theorem,” so it makes sense to state and prove it here.
- Understanding these ***results*** will be more important than understanding their proofs.

Encoding $\mathcal{S}$-Programs with an Oracle

As with regular $\mathcal{S}$-programs, we will encode an instruction $\mathcal{I}$ in an $\mathcal{S}$-program with a macro by

$$\#(\mathcal{I}) = \langle a, \langle b, c \rangle \rangle$$

where a, b and c are as follows.

(a) If $\mathcal{I}$ has no label then $a = 0$. Otherwise, if this statement has a label L, then $a = \#(L)$.

(b) If $\mathcal{I}$ is a statement "$V \leftarrow O(V)$" for some variable V then $b = 0$. If $\mathcal{I}$ is a statement "$V \leftarrow V + 1$" then $b = 1$. If $\mathcal{I}$ is a statement "$V \leftarrow V - 1$" then $b = 2$. Finally, if $\mathcal{I}$ is a statement "IF $V \neq 0$ GOTO L" for some variable V and label L then $b = 2 + \#(L) \geq 3$.

Encoding $\mathcal{S}$-Programs with an Oracle

(c) If the variable V is mentioned in the instruction $\mathcal{I}$ then $c = \#(V) - 1$.

Note: It is important that the length of a program be computable from its encoding because $\mathcal{S}$-programs with oracles do not have any "dummy statements."

- We will ensure this by requiring that the final statement in any $\mathcal{S}$-program with an oracle must have a label (so that its encoding is positive) if it is a statement with the form "$Y \leftarrow O(Y)$."

Relativized Universality Theorem

Definition: For $n \geq 1$ $x_1, x_2, \ldots, x_n, x_{n+1} \in \mathbb{N}$, and a total function $G : \mathbb{N} \to \mathbb{N}$, $\Phi_G : \mathbb{N}^{n+1} \to \mathbb{N}$ is the partial or total function such that, for all $x_1, x_2, \ldots, x_{n+1}$,

$$\Phi_G^{(n)}(x_1, x_2, \ldots, x_n, x_{n+1})$$

is the value produced as output when the $\mathcal{S}$-program with an oracle $\mathcal{P}$ such that $\#(\mathcal{P}) = x_{n+1}$ is executed, using G as an oracle, on inputs $x_1, x_2, \ldots, x_n$. This value is undefined if this G-computation does not terminate.

Relativized Universality Theorem

- Thus

$$\Phi_G^{(n)}(x_1, x_2, \ldots, x_n, x_{n+1}) = \Phi_{\mathcal{P}_{x_{n+1}}, G}(x_1, x_2, \ldots, x_n)$$

for the function $\Phi_{\mathcal{P}_{x_{n+1}}, G} : \mathbb{N}^n \to \mathbb{N}$ defined earlier in these notes, where $\mathcal{P}_{x_{n+1}}$ is the $\mathcal{S}$-program, with an oracle, such that $\#(\mathcal{P}_{x_{n+1}}) = x_{n+1}$.

Relativized Universality Theorem

Theorem #8 (Relativized Universality Theorem): Let $G : \mathbb{N} \to \mathbb{N}$ be a total function and let $n \in \mathbb{N}$ such that $n \geq 1$. Then the program

$$\Phi_G^{(n)} : \mathbb{N}^{n+1} \to \mathbb{N}$$

is G-computable.

How This is Proved:

- A "universal $\mathcal{S}$-program with an oracle" is presented and argued to G-compute the function in the claim.

  Details are provided in the supplemental document for this lecture. □

Relativized Step Counter Theorem

Now let $G : \mathbb{N} \to \mathbb{N}$ be a *partial* or total function. Let $n \geq 1$ and consider the total predicate

$$\text{halted}_G^{(n)} : \mathbb{N}^{n+2} \to \{0, 1\}$$

such that, for all $x_1, x_2, \ldots, x_n, y, t \in \mathbb{N}$,

$$\text{halted}_G^{(n)}(x_1, x_2, \ldots, x_n, y, t) = 1$$

if and only if there is a G-computation of the program with number y, beginning with inputs $x_1, x_2, \ldots, x_n$, whose length is at most $t + 1$, which ends with a ***terminal*** snapsot.

Relativized Step Counter Theorem

- The above condition is ***not*** satisfied — so the predicate has value 0 — if the program with number y *crashes* when executed on inputs $x_1, x_2, \ldots, x_n$ after taking at most $t + 1$ steps (so that the final snapshot in the G-computation for the program encoded by y, and inputs $x_1, x_2, \ldots, x_n$, is *not* a terminal snapshot).

Relativized Step Counter Theorem

Theorem #9 (Relativized Step Counter Theorem): For any ***total function*** $G : \mathbb{N} \to \mathbb{N}$, and every number $n \geq 1$, the above predicate $\text{halted}_G^{(n)}$ is G-computable.

How This is Proved:

- The universal $\mathcal{S}$-program with an oracle, used in the proof of the previous result, is modified to keep track of the number of steps used before either the emulated execution terminates or the input bound on the number of steps is exceeded, returning the required output in each case.

  Details are provided in the supplemental document for this lecture. □

Finiteness Theorem

- The next (seemingly minor) result will be needed several lectures later on.
- It is only being included *here* because its proof is also based on the "universal $\mathcal{S}$-program with an oracle" that has now been presented.

Finiteness Theorem

- For each number $u \in \mathbb{N}$ consider the partial function

$$\{u\} : \mathbb{N} \to \mathbb{N}$$

such that, for all $i \in \mathbb{N}$,

  - if $i < \ell(u)$ then $\{u\}(i) = (r(u))_{i+1}$, and
  - if $i \geq \ell(u)$ then $\{u\}(i)$ is undefined.

- This definition implies the following.

  - If $\ell(u) = 0$ then $\{u\}(i)$ is undefined for all $i \in \mathbb{N}$.
  - If k is positive and

$$u = \langle k, [a_0, a_1, \ldots, a_{k-1}] \rangle$$

then $\{u\}(i) = a_i$ for $0 \leq i \leq k - 1$ and $\{u\}(i)$ is undefined for all $i \in \mathbb{N}$ such that $i \geq k$.

Finiteness Theorem

Theorem #10: The total predicate $P^{n+3} \to \{0,1\}$ such that, for all $x_1, x_2, \ldots, x_n, y, t, u \in \mathbb{N}$, $P(x_1, x_2, \ldots, x_n, y, t, u) = 1$ if and only if

$$\text{halted}^{(n)}_{\{u\}}(x_1, x_2, \ldots, x_n, y, t) = 1$$

is computable.

How This is Proved:

- This uses a straightforward modification of the program used in the proof of Theorem #9 (and its analysis) to replace an oracle call with an evaluation of the function $\{u\}$ included in the input.

  Details are in the supplemental document, once again.

  □

Finiteness Theorem

- Once again let $G : \mathbb{N} \to \mathbb{N}$ be a total function. Let us write

$$u \prec G$$

to mean that $\{u\}(i) = G(i)$ for every integer i such that $0 \leq i < \ell(u)$.

Finiteness Theorem

Theorem #11 (Finiteness Theorem): Let G be a total function. Then, for every integer $n \geq 1$ and $x_1, x_2, \ldots, x_n, y, t \in \mathbb{N}$,

$$\text{halted}_G^{(n)}(x_1, x_2, \ldots, x_n, y, t) = 1$$

if and only if there exists a number $u \in \mathbb{N}$ such that

$$u \prec G \quad \text{and} \quad \text{halted}_{\{u\}}^{(n)}(x_1, x_2, \ldots, x_n, y, t) = 1.$$

- See the supplemental document of a proof of this result.

Relativized and Strengthened Parameter Theorem

The following extends the *Parameter Theorem* that was stated and proved in Lecture #10.

Theorem #12 (Strengthened Parameter Theorem): For all positive numbers $m, n \in \mathbb{N}$, there is a primitive recursive function $S_m^n : \mathbb{N}^{n+1} \to \mathbb{N}$ such that, for all $x_1, x_2, \ldots, x_m$. $u_1, u_2, \ldots, u_n, y \in \mathbb{N}$,

$$\Phi^{(m+n)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_n, y) = \Phi^{(m)}(x_1, x_2, \ldots, x_m, S_m^n(u_1, u_2, \ldots, u_n, y)).$$

Furthermore, if $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, y \in \mathbb{N}$ such that

$$S_m^n(u_1, u_2, \ldots, u_n, y) = S_m^n(v_1, v_2, \ldots, v_n, y)$$

then $u_i = v_i$ for $1 \leq i \leq n$.

Relativized and Strengthened Parameter Theorem

Proof:

- It follows by the proof of the original *Parameter Theorem* that the above function $S_m^n : \mathbb{N}^{n+1} \to \mathbb{N}$ exists and is primitive recursive.
- Now let $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, y \in \mathbb{N}$ such that

$$S_m^n(u_1, u_2, \ldots, u_n, y) = S_m^n(v_1, v_2, \ldots, v_n, y). \qquad (1)$$

- Recall that $S_m^n(u_1, u_2, \ldots, u_n, y)$ is the encoding of a program that sets X_{m+i} to have value u_i for $1 \leq i \leq n$ and that continues with the program encoded by y.
- It follows by this observation (since X_{m+i} cannot be set to have two different values, at the same time) that if the equation at line (1) is satisfied then $u_i = v_i$ for $1 \leq i \leq n$, as claimed. □

Relativized and Strengthened Parameter Theorem

Theorem #13 (Relativized and Strengthened Parameter Theorem): For all positive $n, m \in \mathbb{N}$, there exists a primitive recursive function $S_m^n : \mathbb{N}^{n+1} \to \mathbb{N}$ such that for all $x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_n, y \in \mathbb{N}$,

$$\Phi_G^{(m+n)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_n, y) = \Phi_G^{(m)}(x_1, x_2, \ldots, x_m, S_m^n(u_1, u_2, \ldots, u_n, y)).$$

Furthermore, if $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, y \in \mathbb{N}$ such that

$$S_m^n(u_1, u_2, \ldots, u_n, y) = S_m^n(v_1, v_2, \ldots, v_n, y)$$

then $u_i = v_i$ for $1 \leq i \leq n$.

Relativized and Strengthened Parameter Theorem

Proof:

- This is an example of a result (and proof) that relativize: Because the encodings of programs are so similar, the function S_m^n mentioned in this claim is identical to the function S_m^n mentioned in the "Strengthened Parameter Theorem," given above. □