Investigating the Generalization of Rice's Theorem

Enrico Bragastini

Abstract

In Computability, Rice's theory in its classical form hides more information of what is usually expressed in their respective statements. More recent work made this information explicit, allowing to state stronger complexity theoretic-versions of the theorem; this was done by replacing the notion of extensional set of indices of programs, with a set of indices of programs having not only the same extensional behavior, but also similar complexity (Complexity Clique), proving, under very weak complexity assumptions, that any recursive Complexity Clique is trivial.

In this thesis we modify the definition of complexity measure, on which Complexity Cliques are based, to give a measure of complexity even in the case of programs that do not converge. We observe how this affects the behavior of Complexity Cliques and the generalization of Rice's theorem.

1 Introduction

Most classical results in Computability Theory focus on the so-called *extensional properties* of programs, i.e. properties of the partial functions they compute. Among intensional properties of programs, a major role is played by *complexity*. The theoretical research on effective properties of recursive functions under complexity assumptions focused, since the very beginning, on complexity classes, that is classes of recursive functions computable (almost everywhere) within a given bound of complexity t. It is important to understand that a *complexity class* is a set of functions and not of indices; regarded as a set of indices, it is thus extensional by definition.

Asperti, in his 2008 paper on *The Intensional Content of Rice's Theorem*, proposed the notion of *Complexity Cliques*, as an instrument to study decidability properties of program complexity. A *Complexity Clique* is a set of indices for recursive functions closed w.r.t. indices of functions with same extension and similar complexity (defined in the only sensible way, namely up to constants). Asperti based this definition on the framework proposed by Blum (Blum 1967), in particular the definition of *computational complexity measure*, with the intention of sticking to the generality of Blum's axiomatic approach, although that was not the focus of the paper.

In this thesis we redefine the definition of *Computational Complexity Measure*, changing the first of Blum's axioms so that we may have complexity measures even if the program does not converge. This may only work identifying thresholds beyond which complexity is considered infinite of maximum. Then we study how the modified definition affects Asperti's approach up the generalization of Rice's Theorem.

2 Similarity and Complexity Cliques

We shall work with n-ary partial recursive functions over natural numbers. If f is such a partial function, $dom(f)$ and $cod(f)$ respectively denote the domain and range of f. We write $f(n)\downarrow$ if $n \in dom(f)$. We use $\perp$ for the everywhere divergent function, that is $dom(\perp) = \emptyset$. Given two function f and g, we write $f \cong g$ when f and g have the same graph, i.e. $dom(f) = dom(g)$ and for any $n \in dom(f)$, $f(n) = g(n)$. We say a function is finite if it has a finite graph (i.e. $dom(f)$ is a finite set). Partial functions are ordered w.r.t. the set-teoretic inclusion of their graphs: $f \leq g$ if and only if $graph(f) \subseteq graph(g)$. ω is the set of all natural numbers.

Definition 1. *Let f and g be partial functions on natural numbers:*

1. *we say $f \in O(g)$ if and only if there exist n and c such that for any $m \geq n$, if $g(m)\downarrow$ then $f(m) \leq cg(m)$*
2. *$f \in \Theta(g)$ if and only if $f \in O(g)$ and $g \in O(f)$*

Note that if $f \in \Theta(g)$ the domains of f and g may only differ for a finite number of points.

Observation 1.1. *This is true because $f \in \Theta(g) \Rightarrow f \in O(g) \wedge g \in O(f)$, then:*

- $f \in O(g)$*:* $\exists n_1, c_1 \in \mathbb{N} \,.\, \forall m \geq n_1 \,.\, g(m)\downarrow \Rightarrow f(m) \leq c_1 g(m)$

- $g \in O(f)$: $\exists n_2, c_2 \in \mathbb{N} \;.\; \forall m \geq n_2 \;.\; f(m)\downarrow \Rightarrow g(m) \leq c_2 f(m)$

Let $n = max\{n_1, n_2\}$,

- *for all* $m \geq n$, *if* $g(m)\downarrow$, *then* $f(m)$ *must be defined because* $g(m)\downarrow \Rightarrow f(m) \leq c_1 g(m)$;
- *similarly, for all* $m \geq n$, *if* $f(m)\downarrow$, *then* $g(m)$ *must be defined because* $f(m)\downarrow \Rightarrow g(m) \leq c_2 f(m)$.

Therefore, beyond $max\{n_1, n_2\}$, $f(m)\downarrow \iff g(m)$. *With* $m < max\{n_1, n_2\}$, $f(m)$ *and* $g(m)$ *can differ in their domains. However, since* n_1 *and* n_2 *are finite, the domains can only differ for a finite number of points.*

Let us also remark that, $f \in \Theta(g)$ if and only if $g \in \Theta(f)$; in particular $f \in \Theta(g)$ is an *equivalence relation.*

Definition 2. *A pair* $\langle \phi, \Phi \rangle$ *is a computational complexity measure if* ϕ *is a principal effective enumeration of partial recursive functions and* Φ *satisfies the following axioms:*

(a) $\phi_i(\vec{n})\downarrow \Rightarrow \; \Phi_i(\vec{n})\downarrow$

(b) the predicate $\Phi_i(\vec{n}) = m$ *is decidable*

Observation 2.1. *This definition differs from the one on Asperti's paper. On Asperti's paper the definition is based on Blum's axioms, where axiom (a) is defined as* $\phi_i(\vec{n})\downarrow \iff \Phi_i(\vec{n})\downarrow$. *The rest is unchanged.*

This new condition is weaker, since it only says that if $\phi_i(\vec{n})$ *converges then* $\Phi_i(\vec{n})$ *must also converge. This allows for situations where* $\phi_i(\vec{n})\uparrow$ *but* $\Phi_i(\vec{n})$ *is still defined. It could, for example, give an infinite complexity measure beyond a certain threshold. This means that* $dom(\phi_i) \subseteq dom(\Phi_i)$, *thus broading the set of pairs* $\langle \phi, \Phi \rangle$.

We say two programs are *similar* when they compute the same function and have the same complexity, up to constants:

Definition 3. *Two programs* i *and* j *are similar (write* $i \approx j$*) if and only if*

$$\phi_j \cong \phi_i \;\wedge\; \Phi_j \in \Theta(\Phi_i)$$

Observation 3.1. *Similarity is still possible but note that it has slightly different implications. Previously, because* $\phi_i(\vec{n})\downarrow \iff \Phi_i(\vec{n})\downarrow$, *we had*

$$dom(\Phi_i) = dom(\phi_i) = dom(\phi_j) = dom(\Phi_j)$$

Now, with $\phi_i(\vec{n})\downarrow \Rightarrow \Phi_i(\vec{n})\downarrow$, *we have*

$$dom(\Phi_i) \supseteq dom(\phi_i) = dom(\phi_j) \subseteq dom(\Phi_j)$$

This does not create contradictions with the definitions similarity is based on, since if $\Phi_j \in \Theta(\Phi_i)$, *the domains of* Φ_j *and* Φ_i ***may*** *only differ for a finite number of points, thus no contradiction is created. When we say that two programs are similar in this context, we're making a slightly stronger claim abount their complexity as well. Specifically we are saying that their complexity is equivalent not only on the common domain of their computed function* ϕ *but also on a broader domain where the complexity* Φ *is defined even if* ϕ *is not.*

Theorem 4. *Similarity is an equivalence relation.*

Proof. Obvious, since similarity is defined as intersection of two equivalence relations. □

We shall write $[i]_{\approx}$ for the equivalence class of i w.r.t. the similarity relation $\approx$.

Definition 5. *Let $\langle \phi, \Phi \rangle$ be an abstract complexity measure. A set P of natural numbers is a* Complexity Clique*, if and only if for all i and j*

$$i \in P \wedge j \approx i \Rightarrow j \in P$$

Observation 5.1. *This definition is based on the definition of computational complexity masure we have changed and on the definition of similarity build on top of it. We may now find new Complexity Cliques that fall within this definition that were not included in the original definition.*

Despite this, the concept of Complexity Clique remains valid: if i is program already in a Compexity Clique P, every program j similar to i (according to our modified definition), is in P, whether or not their Φ-functions are defined on inputs where their ϕ-functions are not.

Example 6. *The following are examples of Complexity Cliques:*

1. *$\emptyset$ and ω;*
2. *for any index i, $[i]_{\approx}$;*
3. *for any index i, $\{j | \Phi_j \in O(\Phi_i)\}$*

On the other side, the set $\{i | \Phi_i(a) = b\}$ (i.e. the programs whose complexity on input a is b) is not a Complexity Clique, in general.

Observation 6.1. *All of these examples are still valid. Intuitively,*

1. *Sets $\emptyset$ and ω are trivial Complexity Cliques;*
2. *The set $[i]_{\approx}$, for any index i, contains all the indices of programs similar (according to our definition of similarity) to i, therefore it is a Complexity Clique;*
3. *The set $\{j | \Phi_j \in O(\Phi_i)\}$, for any index i, is the set of all programs with complexity bounded above by the complexity of program i. The program i is in the set, since $\Phi_i \in O(\Phi_i)$. Let $j \approx i$, then by our definition of similarity $\Phi_j \in \Theta(\Phi_i)$, which also implies $\Phi_j \in O(\Phi_i)$. Therefore j is in $\{j | \Phi_j \in O(\Phi_i)\}$, making it a Complexity Clique;*
4. *The set $\{i | \Phi_i(a) = b\}$ in general is not a Complexity Clique because it only gathers programs based on their complexities on a specific input a. Let i be in this set and $j \approx i$. There's no gaurantee that j will also be in the set since it might not have the same complexity as i on input a.*

3 Simple properties of Complexity Cliques

A set $A \subset \omega$ of indices is *extensional* if, for all i and j, $i \in A \wedge \phi_i \cong \phi_j \Rightarrow j \in A$; This is equivalent to say that A is the counter-image (index-set) under ϕ of some subset of partial recursive functions.

The notion of Complexity Clique is a strict generalization of the notion of *extensional set* well known in computability theory:

Theorem 7. *Every extensional set is a Complexity Clique*

Proof. Trivial. □

Observation 7.1. *This remains valid as we expanded the definition of Complexity Cliques. Therefore extensional sets are still included in the weaker Complexity Clique definition.*

Of course, the converse is not true: for instance, the class of programs with polynomial complexity is not extensional.

Observation 7.2. *This remains the same, as we may still have Complexity Cliques containing programs that are similar in complexity but do not compute the same functions, making them non-extensional. The set* $\{j|\Phi_j \in O(\Phi_i)\}$ *is a clear example.*

Any Complexity Clique is a union of similarity classes, in particular:

Theorem 8. *C is a Complexity Clique if and only if*

$$C = \bigcup_{i \in C} [i]_{\approx}$$

Proof. ($\Rightarrow$)

1. Let C be a Complexity Clique.
2. For each $i \in C$, the equivalence class of i w.r.t. the similarity relation $\approx$, $[i]_{\approx}$, consists of all j such that $j \approx i$, i.e. i and j are similar.
3. By the definition of Complexity Clique, if $i \in C$ and $j \approx i$, then $j \in C$. This implies that $[i]_{\approx} \subseteq C$.
4. Therefore, $\bigcup_{i \in C} [i]_{\approx} \subseteq C$

□

Proof. ($\Leftarrow$)

1. Let $C = \bigcup_{i \in C} [i]_{\approx}$ and let $j \in C$.
2. Since $C = \bigcup_{i \in C} [i]_{\approx}$, then j must be in some of those $[i]_{\approx}$.
3. Being $j \in [i]_{approx}$ means that $j \approx i$.
4. Since $i \in C$, $j \approx i$ and $j \in C$, this satisfies the condition for C to be a Complexity Clique

□

Observation 8.1. *This theorem uses the concept of equivalence class w.r.t. the similarity relation $\approx$ found in the Complexity Clique definition. As long as we accept the altered axiom for computational complexity measure, and the similarly relation, this theorem remains valid.*

As a consequence, Complexity Cliques have very good algebraic properties; in particular:

Theorem 9. *Complexity Cliques, equipped with the subset relation, are a Complete Boolean Lattice*

Proof. Just note that $\overline{C} = \bigcup_{i \in \overline{C}} [i]_{\approx}$. The rest is easy. □

Observation 9.1. *As before, as long as we accept the altered axiom for the complexity measure definition, the theorem remains valid.*

The good set-theoretic properties of Complexity Cliques are to be compared with the very bad properties of Complexity Classes (stressing again the strong difference between the two notions). In particular:

1. for any complexity measure, Complexity Classes are not closed under union (Hartmanis and Stearns 1965; McCreight and Meyer 1969);
2. there are complexity measures whose Complexity Classes are not closed under intersection [1] (Landweber and Robertson 1972)

Let us finally remark an important property about finite functions.

Theorem 10. *Let C be a Complexity Class and let $i \in C$. If ϕ_i is finite, then for any j such that $\phi_i \cong \phi_j$, $j \in C$*

Proof. For finite functions, extensional equivalence implies similarity. □

4 Generalized Rice's Theorem

Definition 11. *A pair $\langle \phi, \Phi \rangle$ has the* s-m-n property *if for all positive natural numbers m and n there exists a recursive function s_m^n such that, for all i and all $x_1, \ldots, x_m$*

(a) $\phi_{s_m^n(i,x_1,\ldots,x_m)} \cong \lambda y_1, \ldots, y_n . \phi_i(x_1, \ldots, x_m, y_1, \ldots, y_n)$

(b) $\Phi_{s_m^n(i,x_1,\ldots,x_m)} \in \Theta(\lambda y_1, \ldots, y_n . \Phi_i(x_1, \ldots, x_m, y_1, \ldots, y_n))$

Equation (a) is the standard *s-m-n* theorem, while equation (b) states that the overhead introduced by the function s_m^n is at most a constant factor: in fact, in standard computational models, it amounts to the cost of copying the (fixed) parameters $x_1, \ldots, x_m$ and then calling ϕ_i - see e.g. Cutland (1986); Börger (1986); Odifreddi (1997)

Observation 11.1. *The s-m-n property takes $\phi_{s_m^n}$ and $\Phi_{s_m^n}$ into account separately and therefore it does not require $\phi_{s_m^n}(\vec{n}) \downarrow \iff \Phi_{s_m^n}(\vec{n}) \downarrow$.*

[1] An important exception are measures satisfying the parallel computation property (Landweber and Robertson 1972)

Since the set of recursive functions is still closed under composition, by the *s-m-n theorem* we may conclude that there exists a total computable function h such that $\phi_{h(i,j)} \cong \phi_i \circ \phi_j$. Axioms relating the complexity of $h(i,j)$ to the complexity of its components have been considered by Lischke (1975, 1976, 1977). For the purposes of our analysis, we need a tight bound:

Definition 12. *A pair $\langle \phi, \Phi \rangle$ has the (linear) time-composition property if there exists a total computable function h such that*

(a) $\phi_{h(i,j)} \cong \phi_i \circ \phi_j$

(b) $\Phi_{h(i,j)} \in \Theta(\Phi_i \circ \phi_j + \Phi_j)$

We say that it has the (linear) space-composition property if equation (b) is replaced by

(c) $\Phi_{h(i,j)} \in \Theta(max\{\Phi_i \circ \phi_j, \Phi_j\})$

Equation (b) says that the cost for computing $\phi_i(\phi_j(x))$ is, up to a constant factor, no more than the cost for computing $\phi_j(x)$ plus the cost for computing ϕ_i on $\phi_j(x)$. Similarly, the space required for the same computation is the maximum among the space required for computing $\phi_j(x)$ and the space required for computing ϕ_i on input $\phi_j(x)$.

Observation 12.1. *Equation (a) is about computability and not complexity, so it remains unchanged. In equations (b) and (c), an impact of the weakening of the first axiom could be assessed if $\Phi_i(\vec{n})$ or $\Phi_j(\vec{n})$ give values where $\phi_i(\vec{n})$ or $\phi_j(\vec{n})$ are undefined. However if $\phi_i(\vec{n})$ or $\phi_j(\vec{n})$ are undefined, $\Phi_i(\vec{n})$ or $\Phi_j(\vec{n})$ would result in some kind of infinity complexity, thus making $\Phi_{h(i,j)}$ return an infinite complexity value.*

This means that the properties of (linear) time and space composition can be applied to pairs $\langle \phi, \Phi \rangle$ just as before, including pairs that fall within the new definiton of complexity measure.

The previous equations may be extended to n-ary composition, provided that input variables are linearly partitioned among the components; if duplication of some input is required, we should eventually take this cost into account, adding a linear overhead.

A particular case of (multi-variable) linear composition of frequent use in computability theory is sequencing with absence of communication, i.e. the case of a function

$$\phi_{g(i,j)}(x,y) = \phi_j(x); \phi_i(y) = \begin{cases} \phi_i(y) & \text{if } \phi_j(x) \downarrow \\ \perp & \text{otherwise} \end{cases}$$

The aim is to merely force the evaluation of $\phi_j(x)$ before computing $\phi_i(y)$. According to our definitions

$$\Phi_{g(i,j)} \in \Theta(\lambda xy.(\Phi_j(x) + \Phi_i(y)))$$

for time and

$$\Phi_{g(i,j)} \in \Theta(\lambda xy.max\{\Phi_j(x), \Phi_i(y)\})$$

for space.

In combination with the *s-m-n* property, fixing the parameter x, we have, for both time and space:

Lemma 13. *Let m be the index of a program computing the everywhere divergent function. There exists a total computable function c such that:*

(a) $\begin{cases} \phi_{c(i,j,x)}(y) = \phi_i(y) & \text{if } \phi_j(x)\downarrow \\ \phi_{c(i,j,x)}(y) = \perp & \text{otherwise} \end{cases}$

(b) $\begin{cases} \Phi_{c(i,j,x)}(y) \in \Theta(\Phi_i) & \text{if } \phi_j(x)\downarrow \\ \Phi_{c(i,j,x)}(y) = \Phi_m(y) & \text{otherwise} \end{cases}$

Observation 13.1. *This lemma needed a slight modification from the original one in Asperti's paper. In particular, the complexity $\Phi_{c(i,j,k)}(y)$, with the modified complexity measure definition, can't be equal to $\perp$. It is actually equal to the complexity of a program computing the everywhere divergent function.*

Observation 13.2. *This lemma is compatible with our modified definition of computational complexity measure. In particular, if $\phi_j(x)$ terminates, nothing changes for ϕ and Φ from the standard complexity measure definition.*

If $\phi_j(x)$ does not terminate, $\phi_{c(i,j,k)}(y)$ is equal the everywhere divergent function and $\Phi_{c(i,j,k)}(y)$ is equal to the complexity assigned to divergent programs.

In fact, the previous Lemma is all we need for the generalization of Rice's Theorem to Complexity Cliques.

Theorem 14. *Under the s-m-n (Def. 11) and the linear-composition (Def. 12) assumptions (either in time or space), a Complexity Clique P is recursive if and only if it is trivial, i.e. $P = \emptyset \vee P = \omega$.*

Proof. Assume P is recursive and let ϕ_p be its characteristic function. Suppose P is not trivial: $P \in REC \wedge (P \neq \emptyset \wedge P \neq \omega)$

So there exist a and b such that $\phi_p(a) = 1$ and $\phi_p(b) = 0$. Let m be the index of a program computing the everywhere divergent function: $\forall n \in \mathbb{N} : \phi_m(n)\uparrow$. Either $\phi_p(m) = 1$ or $\phi_p(m) = 0$. Let us consider the latter case, the other one being analogous. Let K be a r.e. non recursive set, and let ϕ_k be its semi-decision function, i.e. $dom(\phi_k) = \mathcal{K}$.

Consider the following function

$$f(x, y) = \phi_k(x); \phi_a(y)$$

By Lemma 13 there exists a total computable function c such that

(a) $\begin{cases} \phi_{c(a,k,x)}(y) = \phi_a(y) & \text{if } x \in \mathcal{K} \\ \phi_{c(a,k,x)}(y) = \perp = \phi_m(y) & \text{otherwise} \end{cases}$

(b) $\begin{cases} \Phi_{c(a,k,x)}(y) \in \Theta(\Phi_a) & \text{if } x \in \mathcal{K} \\ \Phi_{c(a,k,x)}(y) = \Phi_m(y) & \text{otherwise} \end{cases}$

Since P is a Complexity Clique,

$$\phi_p(c(a, k, x)) = \begin{cases} \phi_p(a) = 1 & \text{if } x \in \mathcal{K} \\ \phi_p(m) = 0 & \text{otherwise} \end{cases}$$

So, $\lambda x.\phi_p(c(a, k, x))$ is a characteristic function for $\mathcal{K}$; c is computable, hence ϕ_p cannot be. The case $\phi_p(m) = 1$ is similar.

□