Type-two Iteration with Bounded Query Revision

Bruce M. Kapron*
University of Victoria
Victoria, BC, Canada
bmkapron@uvic.ca

Florian Steinberg
INRIA
Saclay, Île-de-France
florian.steinberg@inria.fr

Motivated by recent results of Kapron and Steinberg (LICS 2018) we introduce new forms of iteration on length in the setting of applied lambda-calculi for higher-type poly-time computability. In particular, in a type-two setting, we consider functionals which capture iteration on input length which bound interaction with the type-one input parameter, by restricting to a constant either the number of times the function parameter may return a value of increasing size, or the number of times the function parameter may be applied to an argument of increasing size. We prove that for any constant bound, the iterators obtained are equivalent, with respect to lambda-definability over type-one poly-time functions, to the recursor of Cook and Urquhart which captures Cobham's notion of limited recursion on notation in this setting.

1 Introduction

Recursion on notation is a fundamental tool for syntactic characterizations of feasible computation, in particular capturing the notion of bounding the number of steps of a computation in terms of input size. However, as a constraint, it is too weak on its own to capture feasibility as characterized by polynomial time computability. A well-known example is the following: consider a function φ, mapping binary strings to binary strings, which for any input string $\mathbf{a}$ returns the string concatenated with itself: $\varphi(\mathbf{a}) = \mathbf{aa}$. The function φ should clearly be accepted as feasible, but recursion on notation allows the definition of a new function which, on input $\mathbf{b}$ of length n returns $\varphi^n(0)$, which is a string of length 2^n. To capture feasibility through a recursion scheme, further restrictions are required to prevent this kind of exponential blowup. Indeed, Cobham, in perhaps one of the earliest works mentioning polynomial-time computability, gives a characterization using a scheme of *limited* recursion on notation [3]. Here, definition of a new function through recursion on functions already known to be from the class is allowed only in case the length of the resulting function may be *a priori* bounded by the length of a function already known to be in the class. Cobham's approach is a canonical example of *explicit* bounding. It is also possible to formulate forms of limited recursion with *implicit* bounding and recover the same class of polynomial time functions [12, 1].

It is possible to consider feasibility in the *type-two* setting, which allows computation with respect to an arbitrary function oracle. The original definition of type-two polynomial time was given by Mehlhorn using a straightforward generalization of Cobham's scheme [13]. Just like the polynomial time functions, this class of functionals allows for a number of different characterizations and is accepted as capturing feasibility at type level two appropriately: Cook and Urquhart gave a formulation of Mehlhorn's class, and in fact generalized it to all finite types by use of an applied typed lambda calculus with constant symbols for type-one poly-time functions, as well as a *recursor* $\mathscr{R}$, which captures Mehlhorn's scheme as a type-two functional [6]. Kapron and Cook showed that Mehlhorn's class may be characterized in

*Research supported in part by an NSERC Discovery Grant

To appear in EPTCS.

terms of oracle Turing machines (OTMs) whose run-time is bounded by a *second-order polynomial* [9]. Both of these characterizations have lead to a multitude of applications and further characterizations.

The content of this paper is inspired by a recent description of Mehlhorn's class given by Kapron and Steinberg [10]. For this it is instructive to think of an analogue of unrestricted recursion on notation in the OTM setting. Informally, this corresponds to Cook's notion of *oracle polynomial time* (OPT) [4], which bounds the running time of OTMs by a polynomial in the size of the input and the largest answer returned by any call to the oracle. Here, a higher time consumption can be justified by an increasing chain of oracle return values and in particular it is possible to recover the example above within OPT. To force feasibility, Kapron and Steinberg use restrictions of OPT based on *query-size revisions*. They considered two forms of revision: a *length revision* occurs when a query to the oracle returns an answer with size larger than the size of the input or the answer to any previous query, a *lookahead revision* occurs when the size of a query provided to the oracle is larger than the size of any previous such query. *Strong polynomial time* (SPT) allows only a constant number of length revisions, while *moderate polynomial time* (MPT) allows only a constant number of lookahead revisions. Kapron and Steinberg prove that both of the classes SPT and MPT give proper subsets of Mehlhorn's class even when restricted to the functionals of the type that they are meant to capture, but that Mehlhorn's class can be recovered from each of the classes by closing under λ-abstraction and application. It should be noted that length revisions and SPT make an earlier apparent in a somewhat different setting in work of Kawamura and Steinberg [11].

The outline of this paper is as follows: In the first section we describe the setting. Namely we work in a simply typed lambda-calculus with constant symbols for all type-1 polynomial-time computable functions. This is identical to the setting Cook and Urquhart chose for their characterization of Mehlhorn's class through the recursor $\mathscr{R}$ and means that we reason about higher-order complexity modulo the availability of the full strength of a first-order bounded recursion scheme. The paper starts from the observation that the bounded recursor $\mathscr{R}$ is meant to model Mehlhorn's scheme, which is strictly more expressive than the first order scheme that is already available through the constants. Clearly $\mathscr{R}$ adds something, as the class of functionals expressible without its presence has been classified by Seth and is considerably restricted in its access of the oracle [14]. Thus, one may ask for functionals that are less expressive and still generate the same class given the context. Section 2 weakens $\mathscr{R}$ in two steps by first simplifying the way in which the bounding is done and afterwards by restricting the data that is available to the step-function. This leaves us with a functional $\mathscr{I}$ that no longer captures bounded recursion but is better understood as doing bounded iteration.

Section 3 starts involving the ideas of length revisions: Inspired by the definitions of the classes SPT and MPT we change the way in which iteration is bounded. The new conditions intuitively provide more freedom than the direct bounding the iterator $\mathscr{I}$ uses and do so in a way that is somewhat orthogonal to how Cook and Urquhart's original recursor $\mathscr{R}$ did more complicated bounding. We are lead to consider a family of operators $\mathscr{I}_k$ where the condition that is imposed becomes less restrictive as k grows. Over the chosen background theory, all of the operators $\mathscr{I}_k$ as well as $\mathscr{R}$ and $\mathscr{I}$ are of equal expressive power. However, the parameter k is tightly connected to runtime-bounds for $\mathscr{I}_k$ in the OTM setting, and the use of higher values should allow expressing some functionals that feature more complicated interaction with the oracle more concisely. The proof that all considered operators are equivalent additionally covers a similarly defined family of iterators based on the idea of lookahead revisions that is introduced in Section 4. The final section specifies an efficient generation scheme for the values of the new iterators.

Kapron and Steinberg define the classes SPT and MPT using the OTM framework which is bound to a specific machine model. This paper transfers the notions of length and lookahead revisions to the machine independent setting of iteration schemes, where the number of iterations is determined by the

length of a specified input parameter (which is a string over some finite alphabet). Our proofs introduce some interesting and useful idioms for programming in this setting.

1.1 Preliminaries

Let Σ denote a finite alphabet that contains symbols `0` and `1`, and Σ^* the set of finite strings over Σ. The empty string is denoted ε, and arbitrary elements of Σ^* are denoted $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$. We attempt to bind names of string variables to their meanings as far as possible: $\mathbf{a}$ is associated with initial values, $\mathbf{b}$ with size-bounds $\mathbf{c}$ with values that a recursion or iteration is carried out over and $\mathbf{t}$ the previous values in a recursion or iteration. For $\mathbf{a} \in \Sigma^*$ let $|\mathbf{a}|$ to denote the length of $\mathbf{a}$ and a_i its digits, i.e., $\mathbf{a} = a_1 \ldots a_{|\mathbf{a}|}$. We write $\mathbf{b} \subseteq \mathbf{a}$ to indicate that $\mathbf{b}$ is an initial segment of $\mathbf{a}$. We assume that we have symbols for all type-1 poly-time functions, for instance:

- *Truncation*: The 2-ary function sending $\mathbf{b}$ and $\mathbf{c} = c_1 c_2 \ldots c_{|\mathbf{c}|}$, to $\mathbf{c}^{\leq |\mathbf{b}|} := c_1 \ldots c_{|\mathbf{b}|}$, if $|\mathbf{b}| \leq |\mathbf{c}|$ and $\mathbf{c}$ otherwise. Note that always $\mathbf{c}^{\leq |\mathbf{b}|} \subseteq \mathbf{c}$ and $\mathbf{c}^{\leq |\varepsilon|} = \varepsilon$. For $\mathbf{c}^{\leq |\mathbf{c}|-1}$ we use the shorthand $\mathbf{c} \gg 1$.
- *Tupling* and *projection* functions $\langle \cdot, \ldots, \cdot \rangle$ and π_i, such that tupling is monotone with respect to length in each argument. Namely if $|\mathbf{a}_i| = |\mathbf{b}_i|$ for all i apart from k, then $|\mathbf{a}_k| \leq |\mathbf{b}_k|$ if and only if $|\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle| \leq |\langle \mathbf{b}_1, \ldots, \mathbf{b}_n \rangle|$.
- *Length minimum*: We adopt the convention used by Cook and Urquart, i.e.

$$\mathsf{lmin}(\mathbf{c}, \mathbf{b}) := \begin{cases} \mathbf{c} & \text{if } |\mathbf{c}| < |\mathbf{b}| \\ \mathbf{b} & \text{otherwise.} \end{cases}$$

We also use definition by cases extensively, relying on the fact that there is a polynomial-time conditional and avoid over-use of λ-abstractions via explicit function definition. Tupling functions that satisfy the demands above exist and are 1-1, but not bijective. In spite of this we still write $\lambda \langle \mathbf{a}_1, \ldots, \mathbf{a}_k \rangle . t[\mathbf{a}_1, \ldots, \mathbf{a}_k]$ as short hand for $\lambda \mathbf{b} . t[\pi_1 \mathbf{b}, \ldots, \pi_k \mathbf{b}]$. This is all done for the sake of readability.

1.2 λ-Definability

The treatment of the typed λ-calculus here follows that used by Cook and Urquart for their characterization of Mehlhorn's class [6]. The set of *types* is defined inductively as follows:

- 0 is a type
- $(\sigma \to \tau)$ is a type, if σ and τ are types.

The set $Fn(\tau)$ of *functionals of type* τ is defined by induction on τ:

- $Fn(0) = \Sigma^*$
- $Fn(\sigma \to \tau) = \{F | F : Fn(\sigma) \to Fn(\tau)\}$.

It is not hard to show that each type τ has a unique normal form

$$\tau = \tau_1 \to \tau_2 \to \cdots \to \tau_k \to 0$$

where the missing parentheses are put in with association to the right. Hence a functional F of type τ is considered in a natural way as a function of variables $X_1, \ldots, X_k$, with X_i ranging over $Fn(\tau_i)$, and returning a natural number value:

$$F(X_1)(X_2)\ldots(X_k) = F(X_1, \ldots, X_k).$$

The *level* of a type is defined inductively: The level of type 0 is 0, and the level of the type τ written in the above normal form is 1 + the maximum of the levels of $\tau_1,\ldots,\tau_k$. This paper is mostly only concerned with functionals of type level smaller or equal two.

Let $\mathbf{X}$ be a class of functionals. The set of λ*-terms* over $\mathbf{X}$, denoted $\lambda(\mathbf{X})$ is defined as follows:

- For each type σ there are infinitely many variables $X^\sigma, Y^\sigma, Z^\sigma, \ldots$ of type σ, and each such variable is a term of type σ.
- For each functional F (of type σ) in $\mathbf{X}$ there is a term F^σ of type σ.
- If T is a term of type τ and X is a variable of type σ, then $(\lambda X.T)$ is a term of type $(\sigma\to\tau)$ (an abstraction).
- If S is a term of type $(\sigma\to\tau)$ and T is a term of type σ, then (ST) is a term of type τ (an application).

For readability, we write $S(T)$ for (ST); we also write $S(T_1,\ldots,T_k)$ for $(\ldots((ST_1)T_2)\ldots T_k)$, and $\lambda X_1\ldots\lambda X_k.T$ for $(\lambda X_1.(\lambda X_2.(\ldots(\lambda X_k.T)\ldots)))$.

The set of free variables of a lambda term can be defined inductively and are those that are not bound by a lambda abstraction. A term is called closed, if it has no free variables. In a natural way each closed λ-term T of type τ represents a functional in $Fn(\tau)$. This correspondence is demonstrated in the standard way, by showing that a mapping of variables to functionals with corresponding type can be extended to a mapping of terms to functionals with corresponding type.

An *assignment* is a mapping φ taking variables to functionals with corresponding type. Suppose φ is an assignment and T a λ-term over $\mathbf{X}$. The value $\mathscr{V}_\varphi(T)$ of T with respect to φ is defined by induction on T as follows.

When T is a variable, $\mathscr{V}_\varphi(T)$ is $\varphi(T)$. If $T=F^\sigma$ is a constant symbol for some $F\in\mathbf{X}$, then $\mathscr{V}_\varphi(T)=F$.

Suppose that $\tau=\tau_1\to\ldots\to\tau_k\to 0$. When T has the form $\lambda X^\sigma.S^\tau$, F is a type σ functional and F_i are type τ_i functionals, then

$$\mathscr{V}_\varphi(T)(F,F_1,\ldots,F_k) := \mathscr{V}_{\varphi'}(S)(F_1,\ldots,F_k),$$

where $\varphi'(X^\sigma)=F$, but φ' is otherwise identical to φ. When T has the form $S^{\sigma\to\tau}R^\sigma$,

$$\mathscr{V}_\varphi(T)(F_1,\ldots,F_k)=\mathscr{V}_\varphi(S)(\mathscr{V}_\varphi(R)(F_1,\ldots,F_k)).\quad\square$$

It is not hard to show that if T,S are terms such that T is a β or η redex and S is its contractum, then for all φ, $\mathscr{V}_\varphi(T)=\mathscr{V}_\varphi(S)$. A functional F is *represented* by a term T relative to an assignment φ if $F=\mathscr{V}_\varphi(T)$.

Our goal in this paper is to prove the equivalence, with respect to λ-representability in the presence of poly-time type-1 functions, of type-2 functionals capturing different forms of recursion on notation. To this end we have the following definitions.

Definition 1.1. Let P be the class of (type-1) poly-time functions, and F,G be functionals. We say that F is P-*reducible* to G, denoted $F\preceq_{\lambda\mathsf{P}}G$ if F is representable by a term of $\lambda(\mathsf{P}\cup\{G\})$, and that F is P-*equivalent* to G, denoted $F\equiv_{\lambda\mathsf{P}}G$, if $F\preceq_{\lambda\mathsf{P}}G$ and $G\preceq_{\lambda\mathsf{P}}F$.

We regularly use that P-reducibility is a transitive relation, which is easily verified. We refer to the class of functionals representable by a term from $\lambda(\mathsf{P}\cup\{G\})$ as the class of functionals *generated by* G. Clearly two functionals are P-equivalent if and only if they generate the same classes of functionals.

2 The Cook-Urquhart recursor and bounded iteration

Our starting point is the recursor that Cook and Urquart use to characterize Mehlhorn's class [6]. This recursor is patterned on the scheme of limited recursion on notation introduced by Cobham [3] and its second-order variant, introduced by Mehlhorn [13]. It is recursively defined by

$$\mathscr{R}(\varphi,\psi,\mathbf{a},\varepsilon) := \mathbf{a} \quad \text{and} \quad \mathscr{R}(\varphi,\psi,\mathbf{a},\mathbf{c}i) := \mathsf{lmin}(\varphi(\mathbf{c}i,\mathbf{t}),\psi(\mathbf{c}i)), \quad \text{where} \quad \mathbf{t} = \mathscr{R}(\varphi,\psi,\mathbf{a},\mathbf{c}).$$

Here, the length minimum lmin returns its left argument if it has strictly smaller length and the right argument otherwise as defined in the preliminaries. The schemes used by Cobham and Mehlhorn feature explicit external bounding that captures almost directly the notion of bounding by a polynomial (in the first-order setting we could easily use a scheme with explicit bounding by polynomials in the argument size, and as shown by [8] this may be extended to the second-order setting as well). In the Cook-Urquart recursor, this limiting is realized via an additional type-1 input ψ. Our first observation is that the limiting may instead be realized through a type-0 input.

Lemma 2.1 ($\mathscr{R} \equiv_{\lambda\mathsf{P}} \mathscr{R}_0$)**.** *The Cook-Urquart recursor and its restriction to constant bounding functions generate the same class of functionals. More specifically $\mathscr{R}$ is P-equivalent to the functional*

$$\mathscr{R}_0(\varphi,\mathbf{b},\mathbf{a},\mathbf{c}) := \mathscr{R}(\varphi,\lambda\mathbf{d}.\mathbf{b},\mathbf{a},\mathbf{c}).$$

Proof. From the definition of $\mathscr{R}_0$ it is immediate that $\mathscr{R}_0 \preceq_{\lambda\mathsf{P}} \mathscr{R}$. To see that also $\mathscr{R} \preceq_{\lambda\mathsf{P}} \mathscr{R}_0$ argue that it suffices to show that $\mathsf{max} \preceq_{\lambda\mathsf{P}} \mathscr{R}_0$, where max is the functional that maximizes return values of a function over the initial segments of a string, i.e. is recursively defined via $\mathsf{max}(\psi,\varepsilon) := \psi(\varepsilon)$ and

$$\mathsf{max}(\psi,\mathbf{c}i) := \begin{cases} \psi(\mathbf{c}i) & \text{if } \mathsf{lmin}(\mathsf{max}(\psi,\mathbf{c}),\psi(\mathbf{c}i)) = \mathsf{max}(\psi,\mathbf{c}) \\ \mathsf{max}(\psi,\mathbf{c}) & \text{otherwise.} \end{cases}$$

Indeed, once this is proven $\mathscr{R} \preceq_{\lambda\mathsf{P}} \mathscr{R}_0$ follows from the equality

$$\mathscr{R}(\varphi,\psi,\mathbf{a},\mathbf{c}) = \mathscr{R}_0(\lambda\mathbf{d}.\lambda\mathbf{t}.\mathsf{lmin}(\varphi(\mathbf{d},\mathbf{t}),\psi(\mathbf{d})),0\mathsf{max}(\psi,\mathbf{c}),\mathbf{a},\mathbf{c}),$$

where the second argument is the maximum with an additional digit added to make sure it is always strictly bigger than any value of ψ on an initial segement of $\mathbf{c}$. This equality can be proven by an induction where the crucial point in the induction step is that the outer of the nested minima always chooses its left argument as value.

To see that the length maximization functional is definable using $\mathscr{R}_0$, note that the argmax functional, which returns the smallest initial segment where a given input-function assumes its maximum, can be defined from $\mathscr{R}_0$ via

$$\mathsf{argmax}(\psi,\mathbf{c}) = \mathscr{R}_0(\lambda\mathbf{d}.\lambda\mathbf{t}.A(\psi,\mathbf{d},\mathbf{t}),\mathbf{c},\varepsilon,\mathbf{c})$$

where $A(\psi,\mathbf{d},\mathbf{t}) = \mathbf{d}$ if $\mathsf{lmin}(\psi(\mathbf{t}),\psi(\mathbf{d})) \neq \psi(\mathbf{t})$ and $\mathbf{t}$ otherwise. Since $\mathsf{max}(\psi,\mathbf{c}) = \psi(\mathsf{argmax}(\psi,\mathbf{c}))$, it follows that max can be expressed and thus that $\mathscr{R} \preceq_{\lambda\mathsf{P}} \mathscr{R}_0$. □

As a further simplification of $\mathscr{R}$, it is possible to eliminate the reference to the current value of the recursion parameter at each step, that is, to replace a functional capturing a form primitive recursion on notation with one that captures functional iteration. This is known as a folklore result, but to the best of our knowledge does not appear explicitly in any previous work in this setting. The most similar characterization we are aware of is one based on typed loop-programs and appeared in [5].

For a function $\varphi\colon \Sigma^* \to \Sigma^*$ let the n-fold iteration φ^n be inductively defined by $\varphi^0(\mathbf{a}) := \mathbf{a}$ and $\varphi^{n+1}(\mathbf{a}) := \varphi(\varphi^n(\mathbf{a}))$. An unbounded iterator would be a functional that takes $n, \mathbf{a}$ and φ as inputs and returns $\varphi^n(\mathbf{a})$. Recall from the introduction, that there are polynomial-time computable φ such that the function $\lambda\mathbf{a}.\lambda\mathbf{b}.\varphi^{|\mathbf{b}|}(\mathbf{a})$ exhibits exponential growth and is in particular not polynomial time computable. Thus, to capture the class of feasible functionals, the considered iterator needs to be bounded. We define the *bounded iterator* $\mathscr{I}$ by

$$\mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) := (\lambda\mathbf{t}.\mathsf{lmin}(\varphi(\mathbf{t}), \mathbf{b}))^{|\mathbf{c}|}(\mathsf{lmin}(\mathbf{a}, \mathbf{b})).$$

That is: $\mathscr{I}$ iterates the input function φ on starting value $\mathsf{lmin}(\mathbf{a}, \mathbf{b})$ for as many times as the input $\mathbf{c}$ is long, while after each iteration truncating the resulting value to be no longer than a bound $\mathbf{b}$.

Before we go on to prove the bounded iterator equivalent to the Cook-Urquart recursor, let us briefly discuss the choices we have taken in bounding. First off, it is easy to see that whether or not the starting value is truncated is irrelevant up to P-equivalence. Furthermore, the definition of $\mathscr{I}$ is such that the bounding is done after φ is applied. Another possibility would be to consider an iterator where the bounding is done on the argument side of φ, i.e. before its application. We give a short proof that the resulting iterator is equivalent.

Lemma 2.2 ($\mathscr{I} \equiv_{\lambda\mathsf{P}} \mathscr{I}'$)**.** *The bounded iterator generates the same class of functionals as its variant that bounds on the argument side. More specifically $\mathscr{I}$ is P-equivalent to the functional*

$$\mathscr{I}'(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) := (\lambda\mathbf{t}.\varphi(\mathsf{lmin}(\mathbf{t}, \mathbf{b})))^{|\mathbf{c}|}(\mathbf{a}).$$

Proof. We prove that for all $\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}$,

$$\mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) = \mathsf{lmin}(\mathscr{I}'(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}), \mathbf{b}) \tag{*}$$

$$\mathscr{I}'(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) = \begin{cases} \mathbf{b} & \text{if } \mathbf{c} = \varepsilon; \\ \varphi(\mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}')) & \text{if } \mathbf{c} = \mathbf{c}'i. \end{cases} \tag{**}$$

We prove (*) and (**) simultaneously by induction on $|\mathbf{c}|$. The case when $|\mathbf{c}| = 0$ is clear, so suppose (*) and (**) hold for all $\mathbf{c}'$ with $|\mathbf{c}'| = k \geq 1$. Consider $\mathbf{c}$ with $|\mathbf{c}| = k+1$, say $\mathbf{c} = \mathbf{c}'i$ where $|\mathbf{c}'| = k$. Then

$$\begin{aligned} \mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) &= \mathsf{lmin}(\varphi(\mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}')), \mathbf{b}) \\ &= \mathsf{lmin}(\mathscr{I}'(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}), \mathbf{b}) && \text{(By IH (**))} \end{aligned}$$

and

$$\begin{aligned} \mathscr{I}'(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) &= \varphi(\mathsf{lmin}(\mathscr{I}'(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}'), \mathbf{b})) \\ &= \varphi(\mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}')). && \text{(By IH (*))} \end{aligned}$$

□

We end the section with the promised proof that the bounded iterator, and thus also its modification from the last lemma, generate the basic feasible functionals. That is, that they generate the same class of functionals as the Cook-Urquart recursor.

Lemma 2.3 ($\mathscr{R} \equiv_{\lambda\mathsf{P}} \mathscr{I}$)**.** *The Cook-Urquart recursor and the bounded iterator are P-equivalent.*

Proof. The first implication, namely that $\mathscr{I} \preceq_{\lambda \mathsf{P}} \mathscr{R}$, follows from the equality

$$\mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) = \mathscr{R}(\lambda \mathbf{d}.\lambda \mathbf{t}.\varphi(\mathbf{t}), \lambda \mathbf{d}.\mathbf{b}, \mathsf{lmin}(\mathbf{a}, \mathbf{b}), \mathbf{c}),$$

that can be proven through a simple induction.

For the converse note that, by Lemma 2.1 the recursor is equivalent to its version $\mathscr{R}_0$ where the bounding is done via a constant instead of a function. Thus it suffices to prove that $\mathscr{R}_0 \preceq_{\lambda \mathsf{P}} \mathscr{I}$. Furthermore note how close the expanded definition of the iterator is to the definition of $\mathscr{R}_0$:

$$\mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \varepsilon) := \mathsf{lmin}(\mathbf{a}, \mathbf{b}) \quad \text{and} \quad \mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}i) := \mathsf{lmin}(\varphi(\mathbf{t}), \mathbf{b}), \quad \text{where} \quad t = \mathscr{I}(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c})$$

The main difference is that for the recursor the function that is recursed on is additionally given access to the value of the string $\mathbf{c}$ that the recursion is done over. We postpone the discussion of how to mend the additional bounding of the initial value to the end of the proof and show that the operator $\mathscr{R}'_0$ defined by

$$\mathscr{R}'_0(\varphi, \mathbf{a}, \mathbf{b}, \varepsilon) := \mathsf{lmin}(\mathbf{a}, \mathbf{b}) \quad \text{and} \quad \mathscr{R}'_0(\varphi, \mathbf{a}, \mathbf{b}, \mathbf{c}i) := \mathsf{lmin}(\varphi(\mathbf{c}i, \mathbf{t}), \mathbf{b}), \quad \text{where} \quad \mathbf{t} = \mathscr{R}'_0(\varphi, \mathbf{a}, \mathbf{b}, \mathbf{c}).$$

can be expressed by using the bounded iterator. To achieve this define

$$\Phi(\varphi, \mathbf{b}, \mathbf{c}) := \lambda \langle \mathbf{u}, \mathbf{v} \rangle . \langle \mathbf{u}0, \mathsf{lmin}(\varphi(\mathbf{c}^{\leq |\mathbf{u}|+1}, \mathbf{v}), \mathbf{b}) \rangle.$$

In the above φ has the type of a functional input of the recursor $\mathscr{R}'_0$ and $\Phi(\varphi, \mathbf{b}, \mathbf{c})$ has the type of a functional input for the bounded iterator for fixed φ, $\mathbf{b}$ and $\mathbf{c}$. We claim that for any $\mathbf{c}$, and $\mathbf{c}' \subseteq \mathbf{c}$,

$$\mathscr{I}(\Phi(\varphi, \mathbf{b}, \mathbf{c}), \langle 0^{|\mathbf{c}|}, \mathbf{b} \rangle, \langle \varepsilon, \mathbf{a} \rangle, \mathbf{c}') = \langle 0^{|\mathbf{c}'|}, \mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}') \rangle \qquad (*)$$

and so, in particular

$$\mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}) = \pi_2(\mathscr{I}(\Phi(\varphi, \mathbf{b}, \mathbf{c}), \langle 0^{|\mathbf{c}|}, \mathbf{b} \rangle, \langle \varepsilon, \mathbf{a} \rangle, \mathbf{c})),$$

which proves the P-reducibility of $\mathscr{R}'_0$ to $\mathscr{I}$. The equality (*) can be verified by fixing $\mathbf{c}$, an proving the following statement by induction on $\mathbf{c}'$: if $\mathbf{c}' \subseteq \mathbf{c}$, then (*) holds for $\mathbf{c}'$. The base case of this induction follows from the properties we demanded the pairing functions to have. Next suppose that the assertion is true for $\mathbf{c}'$. If $\mathbf{c}'i \subseteq \mathbf{c}$, then it is also the case that $\mathbf{c}' \subseteq \mathbf{c}$, and the induction hypothesis implies that (*) holds for $\mathbf{c}'$. But then

$$\begin{aligned}
\mathscr{I}(\Phi(\varphi, \mathbf{b}, \mathbf{c}), \langle 0^{|\mathbf{c}|}, \mathbf{b} \rangle, \langle \varepsilon, \mathbf{a} \rangle, \mathbf{c}'i) &= \mathsf{lmin}(\Phi(\varphi, \mathbf{b}, \mathbf{c})(\langle 0^{|\mathbf{c}'|}, \mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}') \rangle), \langle 0^{|\mathbf{c}|}, \mathbf{b} \rangle) \\
&= \mathsf{lmin}(\langle 0^{|\mathbf{c}'|}0, \mathsf{lmin}(\varphi(\mathbf{c}^{\leq |\mathbf{c}'|+1}, \mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}')), \mathbf{b}) \rangle, \langle 0^{|\mathbf{c}|}, \mathbf{b} \rangle) \\
&= \mathsf{lmin}(\langle 0^{|\mathbf{c}'i|}, \mathsf{lmin}(\varphi(\mathbf{c}'i, \mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}')), \mathbf{b}) \rangle, \langle 0^{|\mathbf{c}|}, \mathbf{b} \rangle) \\
&= \mathsf{lmin}(\langle 0^{|\mathbf{c}'i|}, \mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}'i) \rangle, \langle 0^{|\mathbf{c}|}, \mathbf{b} \rangle) \\
&= \langle 0^{|\mathbf{c}'i|}, \mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}'i) \rangle
\end{aligned}$$

Where the last equality uses the properties of the tupling functions again and the fact that $\mathscr{R}'_0(\varphi, \mathbf{b}, \mathbf{a}, \mathbf{c}'i)$ is either strictly shorter than $\mathbf{b}$ or equal to $\mathbf{b}$.

Finally, to change the initial value, define H as follows:

$$H(\varphi, \mathbf{d}, \mathbf{t}, \mathbf{a}) = \begin{cases} \varphi(\mathbf{d}, \mathbf{t}) & \text{if } |\mathbf{t}| > 1; \\ \varphi(\mathbf{d}, \mathbf{a}) & \text{otherwise,} \end{cases}.$$

then

$$\mathscr{R}_0(\varphi, \mathbf{a}, \mathbf{b}, \mathbf{c}) = \begin{cases} \mathbf{a} & \text{if } \mathbf{c} = \varepsilon; \\ \mathscr{R}'_0(\lambda \mathbf{t}.\lambda \mathbf{d}.H(\varphi, \mathbf{d}, \mathbf{t}, \mathbf{a}), \mathbf{a}, \mathbf{b}, \mathbf{c}) & \text{otherwise} \end{cases}$$

and thus we obtain that $\mathscr{R}_0 \preceq_{\lambda \mathsf{P}} \mathscr{R}'_0 \preceq_{\lambda \mathsf{P}} \mathscr{I}$ and with the fact $\mathscr{R} \preceq_{\lambda \mathsf{P}} \mathscr{R}_0$ from Lemma 2.1 also the desired P-reducibility $\mathscr{R} \preceq_{\lambda \mathsf{P}} \mathscr{I}$. □

3 Iteration with Constant Length Revision

Both the Cook-Urquart recursor $\mathscr{R}$ as well as the bounded iterator $\mathscr{I}$ require an absolute bound on the size of intermediate value encountered during a recursion. Specifying such a bound *a priori* can be cumbersome and this section provides an alternative way of bounding an iteration that is inspired by the classes SPT and MPT we considered in earlier work [11, 10]. The elementary notion used in the definition of SPT is that of a length revision. In an OTM computation a length revision is encountered whenever the answer to an oracle query is longer than any previous response. This notion of a length revision can easily be translated to the realm of recursion schemes: in a recursive definition a length revision happens when the return value of the step function is bigger than any of the values returned earlier. In particular, define

$$\varphi^n_{!k}(\mathbf{a}) := \underbrace{\varphi(\varphi(\dots\varphi}_{\ell \text{ times}}(\mathbf{a})\dots))$$

where $\ell \leq n$ is maximum such that the sequence of applications contains no more than k *length revisions*, that is an application $\varphi(\mathbf{t})$ where $|\varphi(\mathbf{t})|$ exceeds $|\mathbf{a}|$ and $|\varphi(\mathbf{t}')|$ for any previous call. In particular, when $k = 0$, this means that no calls return a value that exceeds $|\mathbf{a}|$. For $k \geq 0$, the *k-revision iterator* $\mathscr{I}_k$ is the functional defined by

$$\mathscr{I}_k(\varphi, \mathbf{a}, \mathbf{c}) := \varphi^{|\mathbf{c}|}_{!k}(\mathbf{a})$$

Superficially, this definition is similar to that of the bounded iterator $\mathscr{I}$ from the last section. The functional $\mathscr{I}_k$ iterates a function where the iteration is bounded by k just like the bounded iterator does for each fixed bounding argument $\mathbf{b}$. The essential difference is that k is a statically fixed parameter, i.e., $\mathscr{I}_k$ constitutes a family of iterators. Our goal is to show that for each fixed k the operator $\mathscr{I}_k$ is P-equivalent to the bounded iterator $\mathscr{I}$ (Theorem 4.3 below).

Without restrictions on k, neither of the reducibilities required to prove that $\mathscr{I}_k \equiv_{\lambda\mathsf{P}} \mathscr{I}$ are obvious. However, the claim that $\mathscr{I}_k \preceq_{\lambda\mathsf{P}} \mathscr{I}$ should appear reasonable given the OTM-based characterization of the basic feasible functionals [9]. As proven in the last section, $\mathscr{I}$ is P-equivalent to $\mathscr{R}$ and thus it is enough to check that $\mathscr{I}_k$ is a basic feasible functional, i.e., that $\mathscr{I}_k$ is computable by an OTM whose run-time is bounded by a second-order polynomial. This may be done in a straightforward way, but it is important to note that the complexity of the bounding polynomial (in terms of the depth of calls to the function input, rather than the degree) increases with k. In particular, while $\mathscr{I}_k$ is P-equivalent to $\mathscr{I}$ for every k, the revision parameter k provides a finer delineation of expressive power.

Without an appeal to the OTM-based characterization, showing the P-equivalence of $\mathscr{I}_k$ and $\mathscr{I}$ becomes more of a challenge, although the case for $\mathscr{I}_0$ is relatively straightforward:

Lemma 3.1 ($\mathscr{I}_0 \preceq_{\lambda\mathsf{P}} \mathscr{I}$)**.** *The* 0*-revision iterator is* P*-reducible to the bounded iterator.*

Proof. The main hurdle is to account for the difference in how the violation of the bound is realized: $\mathscr{I}_0$ defaults to the previous value in the iteration while $\mathscr{I}$ defaults to the value it is given as bound. Set

$$G(\varphi, \mathbf{t}1, \mathbf{b}) := \mathbf{t}1 \qquad G(\varphi, \mathbf{t}0, \mathbf{b}) := \begin{cases} \varphi(\mathbf{t})0 & \text{if } |\varphi(\mathbf{t})| \leq |\mathbf{b}|; \\ \mathbf{t}1 & \text{otherwise.} \end{cases}$$

Then $\mathscr{I}_0(\varphi, \mathbf{a}, \mathbf{c})$ can be obtained from $\mathscr{I}(\lambda\mathbf{t}.G(\varphi, \mathbf{t}, \mathbf{a}), \mathbf{a}, \mathbf{a}0, \mathbf{c})$ by simply dropping the last bit. Since the definition of G only uses type-1 polynomial time operations and application, the P-reducibility follows. □

Note that for unrestricted iteration it holds that $\varphi^n(\varphi^{n'}(\mathbf{a})) = \varphi^{n+n'}(\mathbf{a})$. The following observation points out a similar additivity property for φ^n_k and is the starting point for recursively constructing P-reductions of $\mathscr{I}_k$ to $\mathscr{I}$:

Lemma 3.2. *For given φ, $\mathbf{a}$ and numbers k and n set*

$$\ell := \min\{i \mid \forall j, i \leq j \leq n \Rightarrow \varphi^i_{!k}(\mathbf{a}) = \varphi^j_{!k}(\mathbf{a})\},$$

then $\ell \leq n$ and it holds that

$$\varphi^n_{!(k+1)}(\mathbf{a}) = \varphi^{n-\ell}_{!1}(\varphi^\ell_{!k}(\mathbf{a})). \qquad (*)$$

Proof. Since n always fulfills the condition in the minimum, it follows that $\ell \leq n$. To see the equality note that the condition of the minimum may be true for two separate reasons: It may be the case that φ gives the same return value on all of the strings $\varphi^\ell_{!k}(\mathbf{a}), \ldots, \varphi^{n-1}_{!k}(\mathbf{a})$. In this case there will be no further length revisions, and so $\varphi^{n-\ell}_{!1}(\varphi^\ell_{!k}(\mathbf{a})) = \varphi^{n-\ell}(\varphi^\ell_{!k}(\mathbf{a})) = \varphi^n_{!(k+1)}(\mathbf{a})$. Thus suppose that it is not the case that φ is constant on these strings and let j be such that $l \leq j \leq n-1$ and $\varphi(\varphi^j_{!k}(\mathbf{a})) \neq \varphi^{l+1}_{!k}(\mathbf{a})$. By definition of ℓ the strings $\varphi^\ell_{!k}(\mathbf{a}), \ldots, \varphi^n_{!k}(\mathbf{a})$ are still all equal. Then $\varphi(\varphi^j_{!k}(\mathbf{a})) = \varphi(\varphi^l_{!k}(\mathbf{a}))$ and $\varphi(\varphi^l_{!k}(\mathbf{a}))$ and $\varphi^{l+1}_{!k}(\mathbf{a})$ can only be different if the $(\ell+1)$-st call to φ triggers the $(k+1)$-st length revision. Thus, in this case $\varphi^m_{!1}(\varphi^l_k(\mathbf{a})) = \varphi(\varphi^l_{!k}(\mathbf{a})) = \varphi^{l+m}_{!(k+1)}(\mathbf{a})$ for any m and in particular $(*)$ must hold. □

In fact, the above proof proves the following slightly stronger statement.

Corollary 3.3. *The equality in the last Lemma may be replaced by $\varphi^n_{!(k+1)}(\mathbf{a}) = \varphi^{n-\ell-1}_{!0}(\varphi(\varphi^\ell_{!k}(\mathbf{a})))$.*

This allows us to establish the following.

Lemma 3.4 ($\mathscr{I}_k \preceq_{\lambda \mathsf{P}} \mathscr{I}$). *For $k \geq 1$, the k-revision iterator is P-reducible to the bounded iterator.*

Proof. We proceed by induction on k. The case of $k = 0$ has been taken care of in Lemma 3.1. Suppose that the Lemma holds for k. We must now define $\mathscr{I}_{k+1}$ using $\mathscr{I}$. By Lemma 3.2 it is sufficient to show that there exists a function that on inputs φ, $\mathbf{a}$, k and n returns the value ℓ and is P-reducible to $\mathscr{I}$. First note that the condition $\varphi^i_k(\mathbf{a}) = \varphi^j_k(\mathbf{a})$ can be checked by a function from $\lambda(\mathsf{P} \cup \{\mathscr{I}_k\}) \subseteq \lambda(\mathsf{P} \cup \{\mathscr{I}\})$, where the inclusion follows by the induction hypothesis. Now all that remains is to use $\mathscr{I}$ to characterize the bounded quantification and search used to define ℓ in Corollary 3.3. Define the following functionals:

$$U(\psi, \mathbf{a}, \mathbf{c}) := \begin{cases} \varepsilon & \text{if } \forall_{i \leq |\mathbf{c}|} \left(\psi(|0^i, \mathbf{a})| = 0\right); \\ 0 & \text{otherwise.} \end{cases}$$

$$M(\psi, \mathbf{a}, \mathbf{c}) := 0^j \text{ where } j = \mu_{i \leq |\mathbf{c}|} \left(|\psi(1^i, \mathbf{a})| > 0\right) \text{ if such } i \text{ exists, and } i+1 \text{ otherwise.}$$

We first show that $U \preceq_{\lambda \mathsf{P}} \mathscr{R}$ and appeal to Lemma 2.3 to see that it is P-reducible to $\mathscr{I}$. Define

$$V(\psi, \mathbf{a}, \mathbf{t}, \mathbf{d}) := \begin{cases} \varepsilon & \text{if } \mathbf{t} = \varepsilon \text{ and } |\psi(0^{|\mathbf{d}|}, \mathbf{a})| = 0; \\ 0 & \text{otherwise.} \end{cases}$$

Then $U(\psi, \mathbf{a}, \mathbf{c}) = \mathscr{R}(\lambda \mathbf{t}.\lambda \mathbf{d}.V(\psi, \mathbf{a}, \mathbf{t}, \mathbf{d}), \lambda \mathbf{d}.0, \varepsilon, \mathbf{c})$. Since the definition of V only uses polynomial-time computable type-1 functions and application we conclude $U \preceq_{\lambda \mathsf{P}} \mathscr{R}$. To show that $M \preceq_{\lambda \mathsf{P}} \mathscr{I}$, first define

$$N(\psi, \mathbf{a}, \mathbf{t}0) := \mathbf{t}0 \qquad N(\psi, \mathbf{a}, \mathbf{t}1) := \begin{cases} 0\mathbf{t} & \text{if } |\psi(0^{|\mathbf{t}| \dot{-} 1}, \mathbf{a})| \leq 0; \\ \mathbf{t} \gg 1 & \text{otherwise,} \end{cases}$$

and define

$$A(\psi, \mathbf{a}) := \begin{cases} 0 & \text{if } |\psi(\varepsilon, \mathbf{a})| > 0; \\ 01 & \text{otherwise.} \end{cases}$$

Then $M(\psi, \mathbf{a}, \mathbf{c}) = \mathbf{m} = \mathscr{I}(\lambda \mathbf{t}.N(\psi, \mathbf{a}, \mathbf{t}), A(\psi, \mathbf{a}), \mathbf{c}0, \mathbf{c}) \gg 1$. In particular, if $\mathbf{m}$ ends in 0, then $\mathbf{m} = 0^{j+1}$ and if it ends in 1 then $\mathbf{m} = 0^{|\mathbf{c}|+1}1$. □

4 Iteration with Constant Lookahead Revision

Moving to lookahead revision, the definition is similar. Consider the following variant of function iteration

$$\varphi^n_{?k}(\mathbf{a}) := \underbrace{\varphi(\varphi(\ldots\varphi}_{\ell \text{ times}}(\mathbf{a})\ldots))$$

where $\ell \leq n$ is maximum such that the sequence of applications contains no more than k *lookahead revisions*, that is an application $\varphi(\mathbf{t})$ where $|\mathbf{t}|$ exceeds $|\mathbf{t}'|$ for any previous call $|\varphi(\mathbf{t}')|$. Note that we have not included the initial call $\varphi(\mathbf{a})$ as a lookahead revision (choosing to do so would not change any results below.) Then, for $k \geq 0$, the *k-lookahead revision iterator* $\mathscr{I}'_k$ is the functional such that

$$\mathscr{I}'_k(\varphi,\mathbf{a},\mathbf{c}) = \varphi^{|\mathbf{c}|}_{?k}(\mathbf{a})$$

We now consider the relative power of $\mathscr{I}'_k$.

Lemma 4.1. *For any $k \geq 0$, $\mathscr{I}'_k \preceq_{\lambda\mathsf{P}} \mathscr{I}_k$.*

Proof. We claim that $\mathscr{I}'_k(\varphi,\mathbf{a},\mathbf{c}) = \varphi(\mathscr{I}_k(\varphi,\mathbf{a},\mathbf{c} \gg 1))$. This is clear in the case that there are no more than k length revisions in the evaluation of $\mathscr{I}_k(\varphi,\mathbf{a},\mathbf{b},\mathbf{c} \gg 1)$, as any lookahead revision corresponds exactly to a preceding length revision, and so $\mathscr{I}'_k(\varphi,\mathbf{a},\mathbf{c}) = \varphi^{|\mathbf{c}|}(\mathbf{a}) = \varphi(\mathscr{I}_k(\varphi,\mathbf{a},\mathbf{c} \gg 1))$. Otherwise suppose that $\mathscr{I}_k(\varphi,\mathbf{a},\mathbf{b},\mathbf{c} \gg 1) = \varphi^{\ell}(\mathbf{a})$, which means in particular that ℓ is the minimum value less than $|\mathbf{c}|$ such that evaluating $\varphi^{\ell+1}(\mathbf{a})$ results in $k+1$ length revisions. But then ℓ is the minimum value less than $|\mathbf{c}|$ such that $\varphi^{\ell+2}(\mathbf{a})$ results in $k+1$ length revisions. But this means $\mathscr{I}'_k(\varphi,\mathbf{a},\mathbf{c}) = \varphi^{\ell+1}(\mathbf{a}) = \varphi(\mathscr{I}_k(\varphi,\mathbf{a},\mathbf{c} \gg 1))$. □

Lemma 4.2. *For any $k \geq 0$, $\mathscr{I}' \preceq_{\lambda\mathsf{P}} \mathscr{I}'_k$.*

Proof. Unfortunately, the situation is a little less straightforward than we might hope, as $\mathscr{I}'$ and $\mathscr{I}'_k$ differ slightly in the way they do bounding. $\mathscr{I}'_k$ expects queries to be bounded in length by previous queries while $\mathscr{I}'$ uses an explicit bound $\mathbf{b}$. Define ψ as follows:

$$\psi(\mathbf{t}0,\mathbf{a},\mathbf{b}) := \mathsf{lmin}(\mathbf{a},\mathbf{b})1 \qquad \psi(\mathbf{t}1,\mathbf{a},\mathbf{b}) := \mathsf{lmin}(\varphi(\mathbf{t}),\mathbf{b})1$$

Claim. For all $\mathbf{a},\mathbf{b},\mathbf{c}$, $\mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,0\mathbf{c}) = \mathsf{lmin}(\mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}),\mathbf{b})1$

To prove this claim, first note the in the iteration on the left, the first call to ψ is $\mathbf{b}0$, of length $|\mathbf{b}|+1$. All subsequent calls are clearly bounded by $|\mathbf{b}|+1$. So there will be no lookahead revisions in this iteration and it remains to prove equality without consideration of the lookahead bound k. We use induction on $\mathbf{c}$. When $\mathbf{c} = \varepsilon$,

$$\begin{aligned}
\mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,0\mathbf{c}) &= \mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,0) \\
&= \psi(\mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,\varepsilon),\mathbf{a},\mathbf{b}) \\
&= \psi(\mathbf{b}0,\mathbf{a},\mathbf{b}) \\
&= \mathsf{lmin}(\mathbf{a},\mathbf{b})1 \\
&= \mathsf{lmin}(\mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}),\mathbf{b})1.
\end{aligned}$$

Now assume that the claim holds for $\mathbf{c}$. Then

$$\begin{aligned}\mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,0\mathbf{c}i) &= \psi(\mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,0\mathbf{c}),\mathbf{a},\mathbf{b})\\ &= \psi(\mathsf{lmin}(\mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}),\mathbf{b})1,\mathbf{a},\mathbf{b})\\ &= \mathsf{lmin}(\varphi(\mathsf{lmin}(\mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}),\mathbf{b})),\mathbf{b})1\\ &= \mathsf{lmin}(\mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}i),\mathbf{b})1\end{aligned}$$

Now define $\mathscr{I}''$ as follows:

$$\mathscr{I}''(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}) = \begin{cases}\mathbf{a} & \text{if } \mathbf{c}=\varepsilon;\\ \varphi(\mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,0(\mathbf{c}\gg 1))\gg 1) & \text{otherwise.}\end{cases}$$

First note that $\mathscr{I}'' \preceq_{\lambda P} \mathscr{I}'_k$, as definition by cases is a poly-time operation. When $\mathbf{c}=\varepsilon$, $\mathscr{I}''(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}) = \mathbf{a} = \mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c})$. Otherwise, by the claim,

$$\mathscr{I}'_k(\lambda\mathbf{t}.\psi(\mathbf{t},\mathbf{a},\mathbf{b}),\mathbf{b}0,0(\mathbf{c}\gg 1))\gg 1 = \mathsf{lmin}(\mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}\gg 1),\mathbf{b}),$$

so that

$$\mathscr{I}''(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}) = \varphi(\mathsf{lmin}(\mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}\gg 1),\mathbf{b})) = \mathscr{I}'(\varphi,\mathbf{a},\mathbf{b},\mathbf{c}).$$

□

Putting everything together, we have a characterization which is the main result of this paper.

Theorem 4.3. *For every $k\geq 0$, $\mathscr{I} \equiv_{\lambda\mathsf{P}} \mathscr{I}_k \equiv_{\lambda\mathsf{P}} \mathscr{I}'_k \equiv_{\lambda\mathsf{P}} \mathscr{I}'$.*

5 More efficient approaches

The implementation of $\mathscr{I}_k$ by $\mathscr{I}$ given in Lemma 3.4 requires considerable overhead, involving a bounded quanitification and bounded search at each step. An implementation which directly follows this definition is poly-time, but is needlessly complex. The following observation (which in this setting correpsonds to tail-recursion elimination) will simplify things considerably. In particular, we note an alternate characterization of φ^n: $\varphi^0(\mathbf{a}) = \mathbf{a}$ and $\varphi^{n+1}(\mathbf{a}) = \varphi^n(\varphi(\mathbf{a}))$. This leads to the following characterization of $\varphi^n_{!k}$.

Lemma 5.1. *For all $n,k\geq 0$ we have*

$$\begin{aligned}\varphi^0_{!k}(\mathbf{a}) &= \mathbf{a}\\ \varphi^{n+1}_{!0}(\mathbf{a}) &= \begin{cases}\mathbf{a} & \textit{if } |\varphi(\mathbf{a})|>|\mathbf{a}|;\\ \varphi^n_{!0}(\varphi(\mathbf{a})) & \textit{otherwise.}\end{cases}\\ \varphi^{n+1}_{!(k+1)}(\mathbf{a}) &= \begin{cases}\varphi^n_{!k}(\varphi(\mathbf{a})) & \textit{if } |\varphi(\mathbf{a})|>|\mathbf{a}|;\\ \varphi^n_{!(k+1)}(\varphi(\mathbf{a})) & \textit{otherwise.}\end{cases}\end{aligned}$$

Proof. We prove by induction on k that the claim holds for all n. When $k=0$, iteration stops (absolutely) if $|\varphi(\mathbf{a})|>|\mathbf{a}|$, otherwise it proceeds to the next step. Now assume for k that the claim holds for all n. We show that for $k+1$ it holds for all n, by induction on n. When $n=0$ this is immediate. Assume that it holds for n, and consider $\varphi^{n+1}_{!(k+1)}(\mathbf{a})$. Clearly, if $|\varphi(\mathbf{a})|\leq|\mathbf{a}|$, no length revision occurs on the first call, and so $k+1$ are still available for the remaining n calls. Otherwise, only k length revisions are available for the remaining calls. □

We also note that, implicit in the proof of Lemma 3.4, is an implementation which is also efficient – in particular, if we "unwind" the induction, we are eventually left relying only on $\mathscr{I}_0$. As described in [10], §4.3, we can implement the resulting definition using a form of "re-entrant" recursion. We may view the violation of the length-revision bound as triggering an exception, which may then be caught by an exception handler which re-starts the recursion at the point after the offending oracle call has taken place.

6 Conclusions and Future Work

We have provided a new linguistic characterization of the higher-order polynomial time via iteration schemes that restrict the number of times a step function, presented as an oracle, may return an answer or be presented an input which in length exceeds all previous answers (resp. queries). The characterization and the methods used to prove it lead to a number of questions and potential directions for future research.

The characterization provided in this paper could be termed *intrinsic*, in that no external bounding is present in the iteration schemes $\mathscr{I}_k$ and $\mathscr{I}'_k$. The condition itself, however, appears to depend on the dynamics of a particular computation. On its face it is not a structural/syntactic restriction, as is usual in implicit computational complexity. This suggests two directions for further research. The first is to investigate the possibility of statically deriving bounds on query revision. The second is to investigate distinctions on how computational resources are bounded as suggested by this and related work, for example intrinsic versus extrinsic, dynamic versus static, and feasibly constructive versus non-feasibly constructive (an example of non-feasibly constructive bounding would be the second-order polynomials of [9]). A related observation is that iteration with bounded query revision appears to be a generalization of non-size-increasing computation [7]. This apparent connection merits further investigation.

In §5 above, we begin to explore the interplay between familiar programming techiques from the implemenation of functional programming languages (e.g, tail-recursion elimination) with respect to the efficient implementation of our iteration schemes. We have also noted that the introduction on control primitives (e.g., `catch` and `throw`) may be relevant to the characterization of complexity classes in this setting. We note that such control operators have been shown in [2] to be relevant to the general characterization of *sequential* higher-order computation. Here we only scratch the surface. Further investigation of these and related techniques in the context of linguistic characterizations of computational complexity could prove fruitful.

As noted at several points in our development, there are issues of finer-grained complexity that arise from our translations. This gives rise to natural questions on the efficiency, or syntactic complexity, of translations, which bear further investigation.

Finally, while we have drawn an analogy between OTMs with bounded query revision (as introduced in [10]) and certain recursion schemes, we have not investigated just how closely related they are. While the equivalences proved in [10] and in this paper imply an equivalence for all the models, a direct proof would be very interesting in furthering our understanding of poly-time OTMs. It would be very rewarding if a simplified proof of the equivalence of [9] could be obtained in this setting.

References

[1] Stephen Bellantoni & Stephen A. Cook (1992): *A New Recursion-Theoretic Characterization of the Polytime Functions*. *Computational Complexity* 2, pp. 97–110, doi:10.1007/BF01201998. Available at `https://doi.org/10.1007/BF01201998`.

[2] R. Cartwright, P.L. Curien & M. Felleisen (1994): *Fully Abstract Semantics for Observably Sequential Languages*. Information and Computation 111(2), pp. 297 – 401.

[3] A. Cobham (1965): *The intrinsic computational difficulty of functions*. In Yehoshua Bar-Hillel, editor: Logic, Methodology and Philosophy of Science: Proc. 1964 Intl. Congress (Studies in Logic and the Foundations of Mathematics), North-Holland Publishing, pp. 24–30.

[4] S.A. Cook (1992): *Computability and complexity of higher type functions*. In: Logic from computer science (Berkeley, CA, 1989), Math. Sci. Res. Inst. Publ. 21, Springer, New York, pp. 51–72.

[5] S.A. Cook & B.M. Kapron (1990): *Characterizations of the basic feasible functionals of finite type*. In: Feasible mathematics (Ithaca, NY, 1989), Progr. Comput. Sci. Appl. Logic 9, Birkhäuser, pp. 71–96.

[6] S.A. Cook & A. Urquhart (1993): *Functional interpretations of feasibly constructive arithmetic*. Ann. Pure Appl. Logic 63(2), pp. 103–200.

[7] Martin Hofmann (2003): *Linear types and non-size-increasing polynomial time computation*. Inf. Comput. 183(1), pp. 57–85, doi:10.1016/S0890-5401(03)00009-9. Available at `https://doi.org/10.1016/S0890-5401(03)00009-9`.

[8] A. Ignjatovic & A. Sharma (2004): *Some applications of logic to feasibility in higher types*. ACM TOCL 5(2), pp. 332–350.

[9] B.M. Kapron & S.A. Cook (1996): *A new characterization of type-2 feasibility*. SIAM J. Comput. 25(1), pp. 117–132.

[10] B.M. Kapron & F. Steinberg (2018): *Type-two polynomial-time and restricted lookahead*. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (Oxford, UK), 2018, ACM, New York, pp. 579–598.

[11] Akitoshi Kawamura & Florian Steinberg (2017): *Polynomial Running Times for Polynomial-Time Oracle Machines*. In: 2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK, pp. 23:1–23:18, doi:10.4230/LIPIcs.FSCD.2017.23. Available at `https://doi.org/10.4230/LIPIcs.FSCD.2017.23`.

[12] Daniel Leivant (1991): *A Foundational Delineation of Computational Feasiblity*. In: Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science (Amsterdam, The Netherlands), 1991, IEEE Computer Society, pp. 2–11.

[13] K. Mehlhorn (1976): *Polynomial and abstract subrecursive classes*. J. Comp. Sys. Sci. 12(2), pp. 147–178.

[14] Anil Seth (1993): *Some desirable conditions for feasible functionals of type* 2. In: Eighth Annual IEEE Symposium on Logic in Computer Science (Montreal, PQ, 1993), IEEE Comput. Soc. Press, Los Alamitos, CA, pp. 320–331, doi:10.1109/LICS.1993.287576. Available at `https://doi.org/10.1109/LICS.1993.287576`.