Analyzing Innermost Runtime Complexity Through Tuple Interpretations

Liye Guo*
Institute for Computing and Information Sciences
Radboud University, The Netherlands
l.guo@cs.ru.nl

Deivid Vale*
Institute for Computing and Information Sciences
Radboud University, The Netherlands
deividvale@cs.ru.nl

Tuple interpretations are a class of algebraic interpretations that subsume both polynomial and matrix interpretations as they do not impose simple termination and allow non-linearity. They were developed in the context of higher-order rewriting to study derivational complexity of algebraic functional systems. In this paper, we study innermost runtime complexity of first-order applicative rewriting systems by tailoring tuple interpretations to deal with innermost runtime complexity. This simplifies the search for cost interpretations since the strong monotonicity requirement, which is present in full rewriting, is dropped. We prove an innermost version of the compatibility theorem, i.e., if all rules of the system can be oriented, then the complexity relation is contained in the strict (cost) component of the cost–size algebra. We then go on to demonstrate the expressivity of cost–size tuples and how they can be used to bound the runtime complexity of applicative systems.

1 Introduction

In the step-by-step computational model induced by rewriting, time complexity is naturally understood as the number of rewriting steps needed to reach normal forms. It is usually the case that the cost of firing a redex, i.e., performing a computational step, is assumed constant. So the intricacies of a low level rewriting realization (for instance in Turing Machines) are ignored. This assumption does not pose a problem as long as the low level time complexity needed to apply a rule is kept low. Additionally, this abstract approach has the advantages of being independent of the specific hardware platform evaluating the rewriting system at hand.

In this rewriting setting, a complexity function bounds the length of rewrite sequences and is parametrized by the size of the starting term of the derivation. Two distinct complexity notions are commonly considered in the literature: derivational and runtime complexity, and they differ by the restrictions imposed on the initial term of derivations. On the one hand, derivational complexity imposes no restriction on the set of initial terms. Intuitively, it captures the worst-case behavior of reducing a term to normal form. On the other hand, runtime complexity requires basic initial terms which, conceptually, are terms where a single function call is performed on data (e.g., integers, lists, and trees) as arguments.

If programs are expressed by rewriting, their execution time is closely related to the runtime complexity of the associated rewrite system. Similarly related are programs using call-by-value evaluation strategy and innermost rewrite systems. Therefore, by combining these two concepts, we obtain a connection between cost analysis of call-by-value programs and the runtime complexity analysis of innermost term rewriting. More importantly, due to the abstract nature of rewriting, it is feasible to forgo any specific

*The authors are supported by the NWO VIDI project "CHORPE", NWO VI.Vidi.193.075 and the NWO TOP project "ICHOR", NWO 612.001.803/7571.

programming language detail and still derive useful term rewriting results that may carry over to programs. For an overview of the applicability of rewriting to program complexity the reader is referred to [1, 19].

Therefore, a rewriting approach to program complexity allows us to fully concentrate on finding techniques to establish bounds to the derivational or runtime complexity functions. A natural way to determine these bounds is adapting the proof techniques used to show termination to deduce the complexity naturally induced by the method. There is a myriad of works following this program. To mention a few, see [2, 4, 6, 13, 14, 20] for interpretation methods, [5, 12, 23] for lexicographic and path orders, and [11, 21] for dependency pairs. In this paper, we follow the same idea and concentrate on investigating the innermost runtime complexity for applicative systems. The termination method for which we base our complexity analysis framework upon is tuple interpretations [16].

Tuple interpretations are an instance of the interpretation method. Thus, we seek to interpret terms in such a way that the rewrite relation can be embedded in a well-founded ordering. The defining characteristic of tuple interpretations is to allow for a split of the complexity measure into abstract notions of cost and size. When distilled into its essence, the ingredient we need to express the concepts of cost and size is: a product $\mathcal{C} \times \mathcal{S}$ of a well-founded set $\mathcal{C}$ — the cost set — and a quasi-ordered set $\mathcal{S}$ — the size set. Intuitively, the cost tuples in $\mathcal{C}$ bound the number of rewriting steps needed to reach normal forms, which is in line with the aforementioned rewriting cost model. Meanwhile, the size tuples in $\mathcal{S}$ are more general. We can use integers, reals, and terms themselves as size. Following the treatment in [16], the construction of cost–size products is done inductively on the structure of types. So we map each type σ to a cost–size product $\mathcal{C}_\sigma \times \mathcal{S}_\sigma$. Hence, in this paper our first-order term formalism follows a type discipline.

While forging new tools for our complexity framework, we would like to not only exhibit bounds to the runtime complexity function but also determine sufficient conditions for its feasibility, that is, the existence of polynomial upper bounds. In the eighties Huet and Oppen [15] conjectured that polynomial interpretations are sufficient to evince feasibility, which was disproved by Lautemann [17] in the same decade. In fact, polynomial interpretations induce a double exponential upper bound on the derivation length, as shown by the seminal work of Hofbauer and Lautemann [14]. Feasibility can be recovered by imposing additional conditions on interpretations. To the best of our knowledge, Cichon and Lescanne [6] were the first to propose such conditions even though their setting is restricted to number theoretic functions only. Similar results are proved in [4], where the authors provide rewriting characterizations of complexity classes using bounds to the interpretation of data constructors. These same conditions appear in the higher-order setting, see [2, 16]. In the present paper, we follow a similar approach to that in [4] and show that we can recover those classical results by bounding size-tuples in interpretations.

Tuple interpretations do not provide a complete termination proof method: there are terminating systems for which interpretations cannot be found. Consequently, it does not induce a complete complexity analysis framework either. Notwithstanding, it has the potential to be very powerful if we choose the cost–size sets wisely. A second limitation is that the search for interpretations is undecidable in general, which is expected already in the polynomial case [18]. Undecidability never hindered computer scientists' efforts on mechanizing difficult problems, however. Indeed, several proof search methods were developed over the years to find interpretations automatically [3, 7, 8, 13, 24].

Contribution. In Definition 1 we provide a formal definition of cost–size products and interpret types, Definition 3, as cost–size products, which defines the interpretation domain for cost–size algebras defined in Definition 6. In Lemmas 2 and 4 we show the soundness of this approach. In Definition 5 we introduce a type-safe application operator on cost–size products and prove its strong monotonicity, an important ingredient to show the Compatibility Theorem 1. We establish termination of Toyama's system

in Example 3, showing that Theorem 1 correctly captures innermost termination in our setting. We provide sufficient conditions so that feasible bounds on innermost runtime complexity can be achieved in Lemmas 7 and 8.

Outline. In Section 2, we fix notation and recall basic notions of rewriting syntax, basic terminology on complexity of rewriting, and review our notation for sets, orders, and functions. In Section 3, we tailor tuple interpretations to the innermost setting and prove the innermost version of the compatibility theorem. We proceed to establish complexity bounds to innermost runtime complexity in Section 4. In Section 5, we present preliminary work on automation techniques to find cost–size tuple interpretations. We conclude the paper in Section 6.

2 Preliminaries

TRSs and Innermost Rewriting. We consider simply typed first-order term rewriting systems in curried notation. Fix a set $\mathcal{B}$, whose elements are called *sorts*. The set $\mathcal{T}_\mathcal{B}$ of *types* is generated by the grammar $\mathcal{T}_\mathcal{B} ::= \mathcal{B} \mid \mathcal{B} \Rightarrow \mathcal{T}_\mathcal{B}$. Each type is written as $\iota_1 \Rightarrow \cdots \Rightarrow \iota_m \Rightarrow \kappa$ where all ι_i and κ are sorts. A *signature* is a set $\mathcal{F}$ of symbols together with an arity function $\mathtt{ar}$ which associates to each $\mathsf{f} \in \mathcal{F}$ a type $\sigma \in \mathcal{T}_\mathcal{B}$. We call the triple $(\mathcal{B}, \mathcal{F}, \mathtt{ar})$ a *syntax signature*. For each sort ι, we postulate a set $\mathcal{X}_\iota$ of countably many variables and assume that $\mathcal{X}_\iota \cap \mathcal{X}_{\iota'} = \emptyset$ if $\iota \neq \iota'$. Let $\mathcal{X}$ denote $\bigcup_\iota \mathcal{X}_\iota$ and assume that $\mathcal{F} \cap \mathcal{X} = \emptyset$.

The set $\mathbb{T}$ of *pre-terms* is generated by the grammar $\mathbb{T} ::= \mathcal{F} \mid \mathcal{X} \mid (\mathbb{T}\ \mathbb{T})$. The set $T(\mathcal{F}, \mathcal{X})$ of *terms* consists of pre-terms which can be typed as follows: (i) $\mathsf{f} : \sigma$ if $\mathtt{ar}(\mathsf{f}) = \sigma$, (ii) $x : \iota$ if $x \in \mathcal{X}_\iota$, and (iii) $(s\ t) : \tau$ if $s : \iota \Rightarrow \tau$ and $t : \iota$. Application of terms is left-associative, so we write $s\ t\ u$ for $((s\ t)\ u)$. Let $\mathtt{vars}(s)$ be the set of variables occurring in s. A *ground term* is a term s such that $\mathtt{vars}(s) = \emptyset$. A symbol $\mathsf{f} \in \mathcal{F}$ is called the *head symbol* of s if $s = \mathsf{f}\ s_1 \ldots s_k$. A *subterm* of s is a term t (we write $s \trianglerighteq t$) such that (i) $s = t$, or (ii) t is a subterm of s' or s'' when $s = s'\ s''$. A *proper subterm* of s is a subterm of s which is not equal to s. A *substitution* γ is a type-preserving map from variables to terms such that the set $\mathtt{dom}(\gamma) = \{x \in \mathcal{X} \mid \gamma(x) \neq x\}$ is finite. Every substitution γ extends to a type-preserving map from terms to terms, whose image on s is written as $s\gamma$, as follows: (i) $\mathsf{f}\gamma = \mathsf{f}$, (ii) $x\gamma = \gamma(x)$, and (iii) $(s\ t)\gamma = (s\gamma)\ (t\gamma)$.

A relation $\to$ on terms is *monotonic* if $s \to s'$ implies $t\ s \to t\ s'$ and $s\ u \to s'\ u$ for all terms t and u of appropriate types. A *rewrite rule* $\ell \to r$ is a pair of terms of the same type such that $\ell = \mathsf{f}\ \ell_1 \ldots \ell_k$ and $\mathtt{vars}(\ell) \supseteq \mathtt{vars}(r)$. A *term rewriting system* (TRS) $\mathcal{R}$ is a set of rewrite rules. The *rewrite relation* $\to_\mathcal{R}$ induced by $\mathcal{R}$ is the smallest monotonic relation on terms such that $\ell\gamma \to_\mathcal{R} r\gamma$ for all rules $\ell \to r \in \mathcal{R}$ and substitutions γ. A *reducible expression* (redex) is a term of form $\ell\gamma$ for some rule $\ell \to r$ and substitution γ. A term is in *normal form* if none of its subterms is a redex. A TRS $\mathcal{R}$ is *terminating* if no infinite rewrite sequence $s \to_\mathcal{R} s' \to_\mathcal{R} s'' \to_\mathcal{R} \cdots$ exists.

Every rewrite rule $\ell \to r$ *defines* a symbol f, namely, the head symbol of ℓ. For each $\mathsf{f} \in \mathcal{F}$, let $\mathcal{R}_\mathsf{f}$ denote the set of rewrite rules that define f in $\mathcal{R}$. A symbol $\mathsf{f} \in \mathcal{F}$ is a *defined symbol* if $\mathcal{R}_\mathsf{f} \neq \emptyset$; otherwise, f is called a *constructor*. Let $\mathcal{D}$ be the set of defined symbols and $\mathcal{C}$ the set of constructors. So $\mathcal{F} = \mathcal{D} \cup \mathcal{C}$. A *data term* is a term of form $\mathsf{c}\ d_1\ \ldots\ d_k$ where c is a constructor and each d_i is a data term. A *basic term* is a term of type ι and of form $\mathsf{f}\ d_1\ \ldots\ d_m$ where ι is a sort, f is a defined symbol and all $d_1, \ldots, d_m$ are data terms. We let $T_b(\mathcal{F})$ denote the set of all basic terms.

Example 1 We fix nat and list for the sorts of natural numbers and lists of natural numbers, respectively. In the below TRS, $\mathsf{0} : \mathsf{nat}$, $\mathsf{s} : \mathsf{nat} \Rightarrow \mathsf{nat}$, $\mathsf{nil} : \mathsf{list}$ and $\mathsf{cons} : \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}$ are constructors while $\mathtt{add}, \mathtt{minus}, \mathtt{quot} : \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$, $\mathtt{append} : \mathsf{list} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}$, $\mathtt{sum} : \mathsf{list} \Rightarrow \mathsf{nat}$ and $\mathtt{rev} : \mathsf{list} \Rightarrow \mathsf{list}$ are

defined symbols.

$$\begin{array}{rlrl}
\texttt{add}\ x\ \mathsf{0} &\to x & \texttt{sum}\ \mathsf{nil} &\to \mathsf{0}\\
\texttt{add}\ x\ (\mathsf{s}\ y) &\to \mathsf{s}\ (\texttt{add}\ x\ y) & \texttt{sum}\ (\mathsf{cons}\ x\ q) &\to \texttt{add}\ (\texttt{sum}\ q)\ x\\
\texttt{append}\ \mathsf{nil}\ l &\to l & \texttt{rev}\ \mathsf{nil} &\to \mathsf{nil}\\
\texttt{append}\ (\mathsf{cons}\ x\ q)\ l &\to \mathsf{cons}\ x\ (\texttt{append}\ q\ l) & \texttt{rev}\ (\mathsf{cons}\ x\ q) &\to \texttt{append}\ (\texttt{rev}\ q)\ (\mathsf{cons}\ x\ \mathsf{nil})\\
\mathsf{minus}\ x\ \mathsf{0} &\to x & \mathsf{quot}\ \mathsf{0}\ (\mathsf{s}\ y) &\to \mathsf{0}\\
\mathsf{minus}\ \mathsf{0}\ y &\to \mathsf{0} & \mathsf{quot}\ (\mathsf{s}\ x)\ (\mathsf{s}\ y) &\to \mathsf{s}\ (\mathsf{quot}\ (\mathsf{minus}\ x\ y)\ (\mathsf{s}\ y))\\
\mathsf{minus}\ (\mathsf{s}\ x)\ (\mathsf{s}\ y) &\to \mathsf{minus}\ x\ y
\end{array}$$

We restrict our attention to innermost rewriting: only redexes with no reducible proper subterms may be reduced. More precisely, the *innermost rewrite relation* $\to^i_{\mathcal{R}}$ induced by $\mathcal{R}$ is defined as follows:

(i) $\ell\gamma \to^i_{\mathcal{R}} r\gamma$ if $\ell \to r \in \mathcal{R}$ and all proper subterms of $\ell\gamma$ are in normal form,

(ii) $s\ t \to^i_{\mathcal{R}} s'\ t$ if $s \to^i_{\mathcal{R}} s'$, and

(iii) $s\ t \to^i_{\mathcal{R}} s\ t'$ if $t \to^i_{\mathcal{R}} t'$.

Below we only analyze innermost rewriting. So we write $\to$ for $\to^i_{\mathcal{R}}$ whenever no ambiguity arises.

Derivation Height and Complexity. Given a relation $\to$ on terms, we write $s \xrightarrow{n} t$ if there is a sequence $s = s_0 \to \cdots \to s_n = t$ of length n. The *derivation height* $\mathtt{dh}(s,\to)$ of a term s with respect to $\to$ is the length of the longest $\to$-sequence of starting with s, i.e., $\mathtt{dh}(s,\to) = \max\{n \mid \exists t \in T(\mathcal{F},\mathcal{X}) : s \xrightarrow{n} t\}$. The *absolute size* of a term s, denoted by $|s|$, is 1 if s is a symbol in $\mathcal{F}$ or a variable, and $|s_1| + |s_2|$ if $s = s_1\ s_2$. In order to express various complexity notions in the rewriting setting, we define the *complexity function* as follows: $\mathtt{comp}(n,\to,\mathcal{T}) = \max\{\mathtt{dh}(s,\to) \mid s \in \mathcal{T} \text{ and } |s| \le n\}$. Intuitively, $\mathtt{comp}(n,\to,\mathcal{T})$ is the length of the longest $\to$-sequence starting with a term whose absolute size is at most n from $\mathcal{T}$. We summarize four particular instances in the following table:

	derivational	runtime
full	$\mathtt{dc}_{\mathcal{R}}(n) = \mathtt{comp}(n,\to_{\mathcal{R}},T(\mathcal{F},\mathcal{X}))$	$\mathtt{rc}_{\mathcal{R}}(n) = \mathtt{comp}(n,\to_{\mathcal{R}},T_b(\mathcal{F}))$
innermost	$\mathtt{idc}_{\mathcal{R}}(n) = \mathtt{comp}(n,\to^i_{\mathcal{R}},T(\mathcal{F},\mathcal{X}))$	$\mathtt{irc}_{\mathcal{R}}(n) = \mathtt{comp}(n,\to^i_{\mathcal{R}},T_b(\mathcal{F}))$

Ordered Sets and Monotonic Functions. A *quasi-ordered set* $(A,\sqsupseteq)$ consists of a nonempty set A and a quasi-order (reflexive and transitive) $\sqsupseteq$ on A. An *extended well-founded set* $(A,>,\ge)$ is a nonempty set A together with a well-founded order $>$ and a quasi-order $\ge$ on A such that $\ge$ is compatible with $>$, i.e., $x > y$ implies $x \ge y$ and $x > y \ge z$ implies $x > z$. Below we refer to an extended well-founded set simply as a *well-founded set*.

Given quasi-ordered sets $(A,\sqsupseteq)$ and $(B,\sqsupseteq)$, a function $f : A \longrightarrow B$ is said to be *weakly monotonic* if $x \sqsupseteq y$ implies $f(x) \sqsupseteq f(y)$. Let $A \Longrightarrow B$ denote the set of weakly monotonic functions from A to B. The comparison operator $\sqsupseteq$ on B induces pointwise comparison on $A \Longrightarrow B$ as follows: $f \sqsupseteq g$ if $f(x) \sqsupseteq g(x)$ for all $x \in A$. This way $(A \Longrightarrow B,\sqsupseteq)$ is also a quasi-ordered set. Given well-founded sets $(A,>,\ge)$ and $(B,>,\ge)$, a function $f : A \longrightarrow B$ is said to be *strongly monotonic* if $x > y$ implies $f(x) > f(y)$ and $x \ge y$ implies $f(x) \ge f(y)$.

3 Tuple Interpretations

In this section, we introduce the notion of tuple algebras in the context of innermost rewriting. We start by interpreting types as cost–size products, give interpretation of terms as cost–size tuples, and finally prove the innermost version of the compatibility theorem.

3.1 Types as Cost–Size Products

We start with defining cost–size products.

Definition 1 (Cost–Size Products) Given a well-founded set $(\mathcal{C}, >, \geq)$, called the *cost set*, and a quasi-ordered set $(\mathcal{S}, \sqsupseteq)$, called the *size set*, we call $\mathcal{C} \times \mathcal{S}$ the *cost–size product* of $(\mathcal{C}, >, \geq)$ and $(\mathcal{S}, \sqsupseteq)$, and its elements *cost–size tuples*.

Cost–size tuples can be ordered as follows:

Definition 2 (Product Order) For all $\langle x, y\rangle$ and $\langle x', y'\rangle$ in $\mathcal{C} \times \mathcal{S}$,

(i) $\langle x, y\rangle \succ \langle x', y'\rangle$ if $x > x'$ and $y \sqsupseteq y'$, and

(ii) $\langle x, y\rangle \succcurlyeq \langle x', y'\rangle$ if $x \geq x'$ and $y \sqsupseteq y'$.

And we get a well-founded set.

Lemma 1 *$(\mathcal{C} \times \mathcal{S}, \succ, \succcurlyeq)$ is a well-founded set.*

PROOF It follows immediately from the definition that $\succ$ and $\succcurlyeq$ are transitive, and $\succcurlyeq$ is reflexive. To prove that $\succ$ is well-founded, note that the existence of $\langle x_1, y_1\rangle \succ \langle x_2, y_2\rangle \succ \cdots$ would imply $x_1 > x_2 > \cdots$, which cannot be the case since $>$ is well-founded.

We still need to check that $\succcurlyeq$ is compatible with $\succ$.

- Suppose $\langle x, y\rangle \succ \langle x', y'\rangle$. Since $x > x'$ implies $x \geq x'$, we have $\langle x, y\rangle \succcurlyeq \langle x', y'\rangle$.
- Suppose $\langle x, y\rangle \succ \langle x', y'\rangle \succcurlyeq \langle x'', y''\rangle$. Since $x > x' \geq x''$ implies $x > x''$ and $\sqsupseteq$ is transitive, we have $\langle x, y\rangle \succ \langle x'', y''\rangle$. ∎

Now we interpret types as a particular kind of cost–size products.

Definition 3 (Interpretation of Types) Let $\mathcal{B}$ denote the set of sorts. An *interpretation key* $\mathcal{J}_\mathcal{B}$ for $\mathcal{B}$ maps each sort ι to a quasi-ordered set $(\mathcal{J}_\mathcal{B}(\iota), \sqsupseteq)$ with a minimum. For each type $\sigma \in \mathcal{T}_\mathcal{B}$, we define the cost–size interpretation of σ as the product $(\!|\sigma|\!) = \mathcal{C}_\sigma \times \mathcal{S}_\sigma$ with

$$\begin{aligned} \mathcal{C}_\sigma &= \mathbb{N} \times \mathcal{F}^{\mathsf{c}}_\sigma \\ \mathcal{F}^{\mathsf{c}}_\iota &= \mathtt{unit} & \mathcal{S}_\iota &= \mathcal{J}_\mathcal{B}(\iota) \\ \mathcal{F}^{\mathsf{c}}_{\iota\Rightarrow\tau} &= \mathcal{S}_\iota \Longrightarrow \mathcal{C}_\tau & \mathcal{S}_{\iota\Rightarrow\tau} &= \mathcal{S}_\iota \Longrightarrow \mathcal{S}_\tau \end{aligned}$$

where $\mathtt{unit} = \{\mathfrak{U}\}$ is quasi-ordered by $\geq$ with $\mathfrak{U} \geq \mathfrak{U}$. All $\mathcal{F}^{\mathsf{c}}_{\iota\Rightarrow\tau}$ and $\mathcal{S}_{\iota\Rightarrow\tau}$ are ordered by pointwise comparison. The set $\mathcal{C}_\sigma$ is ordered as follows: $(n, f) > (m, g)$ if $n > m$ and $f \geq g$, and $(n, f) \geq (m, g)$ if $n \geq m$ and $f \geq g$. This definition requires that all $(\mathcal{C}_\sigma, \geq)$ and $(\mathcal{S}_\sigma, \sqsupseteq)$ are quasi-ordered sets, which is guaranteed by the following lemma.

Lemma 2 *For any type σ, $(\mathcal{C}_\sigma, >, \geq)$ is a well-founded set and $(\mathcal{S}_\sigma, \sqsupseteq)$ is a quasi-ordered set with a minimum. Therefore, $(\!|\sigma|\!)$ is a cost–size product.*

PROOF When σ is a sort, $\mathcal{C}_\sigma = \mathbb{N} \times \mathtt{unit} \cong \mathbb{N}$ and $\mathcal{S}_\sigma = \mathcal{J}_\mathcal{B}(\sigma)$, so the statement is trivially true. When $\sigma = \iota \Rightarrow \tau$, we have $\mathcal{C}_\sigma = \mathbb{N} \times \mathcal{F}^{\mathsf{c}}_{\iota\Rightarrow\tau}$, $\mathcal{F}^{\mathsf{c}}_{\iota\Rightarrow\tau} = \mathcal{J}_\mathcal{B}(\iota) \Longrightarrow \mathcal{C}_\tau$ and $\mathcal{S}_\sigma = \mathcal{J}_\mathcal{B}(\iota) \Longrightarrow \mathcal{S}_\tau$. By induction, $(\mathcal{C}_\tau, \geq)$ and $(\mathcal{S}_\tau, \sqsupseteq)$ are quasi-ordered sets. So are $(\mathcal{F}^{\mathsf{c}}_{\iota\Rightarrow\tau}, \geq)$ and $(\mathcal{S}_\sigma, \sqsupseteq)$, which are ordered by pointwise comparison. By Lemma 1, $(\mathcal{C}_\sigma, >, \geq)$ is a well-founded set. One minimum of $(\mathcal{S}_\sigma, \sqsupseteq)$ is the constant function $\boldsymbol{\lambda} x.\bot$ where $\bot$ is a minimum of $(\mathcal{S}_\tau, \sqsupseteq)$. ∎

The cost component $\mathcal{C}_\sigma$ of $(\!|\sigma|\!)$ holds information about the cost of reducing a term of type σ to its normal form. It has two parts: one is numeric and the other is functional. The functional part $\mathcal{F}^{\mathsf{c}}_\sigma$ degenerates to $\mathtt{unit}$ when σ is just a sort, and is indeed a function space when $\sigma = \iota \Rightarrow \tau$ is a function type. In the latter case, $\mathcal{F}^{\mathsf{c}}_\sigma = \mathcal{S}_\iota \Longrightarrow \mathcal{C}_\tau$ consists of functions with domain $\mathcal{S}_\iota$, the size component of $(\!|\iota|\!)$. This is very much in line with the standard complexity notion based on Turing machines, where time complexity is parametrized by the size of input.

In order to use Definition 3 to interpret types, we need a concrete interpretation key, which chooses a size set for each sort. In our examples, a particular kind of interpretation keys map each sort ι to $(\mathbb{N}^{K[\iota]}, \sqsupseteq)$ where $K[\iota] \geq 1$ and $\langle x_1, \ldots, x_{K[\iota]} \rangle \sqsupseteq \langle y_1, \ldots, y_{K[\iota]} \rangle$ if $x_i \geq y_i$ for all i. Such interpretation keys are used unless otherwise stated. We take a semantic approach (cf. [16]) to determine the number $K[\iota]$ for each sort ι. For example, nat is the sort of natural numbers in unary and $n \in \mathbb{N}$ is represented as $\mathsf{s}(\ldots(\mathsf{s}\ \mathsf{0}))$ with n successive applications of s. The number of occurrences of s is a reasonable measure for the size of a natural number so we let $K[\mathsf{nat}]$ be 1. On the other hand, to characterize the size of a list, we need information about the individual elements in addition to the length of the list. So for each list, we keep track of its length as well as the maximum size of its elements. This way $K[\mathsf{list}] = 2$. See Example 2.

Definition 4 Cost–size tuples in $(\!|\sigma|\!)$ are written as $\langle (n, f^{\mathsf{c}}), f^{\mathsf{s}} \rangle$ where $n \in \mathbb{N}$, $f^{\mathsf{c}} \in \mathcal{F}^{\mathsf{c}}_\sigma$ and $f^{\mathsf{s}} \in \mathcal{S}_\sigma$. When σ is a function type, we refer to f^{c} as the *cost function* and f^{s} as the *size function*.

In order to define the interpretation of terms (Definition 7), we need a notion of application for cost–size tuples. Given $\boldsymbol{f} \in (\!|\iota \Rightarrow \tau|\!)$ and $\boldsymbol{x} \in (\!|\iota|\!)$, our goal is to define $\boldsymbol{f} \cdot \boldsymbol{x} \in (\!|\tau|\!)$. Let us demonstrate with an example. Recall from Example 1 the function $\mathtt{append} : \mathsf{list} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}$, which takes two lists q and l as input. Let $\mathtt{append}$ be interpreted as $\boldsymbol{f} = \langle (n, f^{\mathsf{c}}), f^{\mathsf{s}} \rangle \in (\!|\mathsf{list} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}|\!)$, where

$$
\begin{aligned}
& n \in \mathbb{N}, \\
& f^{\mathsf{c}} \in \overbrace{\mathcal{S}_{\mathsf{list}}}^{\text{size of } q} \Longrightarrow (\mathbb{N} \times (\overbrace{\mathcal{S}_{\mathsf{list}}}^{\text{size of } l} \Longrightarrow (\mathbb{N} \times \mathtt{unit}))), \text{ and} \\
& f^{\mathsf{s}} \in \overbrace{\mathcal{S}_{\mathsf{list}}}^{\text{size of } q} \Longrightarrow (\overbrace{\mathcal{S}_{\mathsf{list}}}^{\text{size of } l} \Longrightarrow \mathcal{S}_{\mathsf{list}}).
\end{aligned}
$$

For the first list q, take a cost–size tuple $\boldsymbol{x} = \langle (m, \mathfrak{U}), x^{\mathsf{s}} \rangle$ from $(\!|\mathsf{list}|\!)$. We apply f^{c} and f^{s} to x^{s}, and get $f^{\mathsf{c}}(x^{\mathsf{s}}) = (k, h) \in \mathbb{N} \times (\mathcal{S}_{\mathsf{list}} \Longrightarrow (\mathbb{N} \times \mathtt{unit}))$ and $f^{\mathsf{s}}(x^{\mathsf{s}}) \in \mathcal{S}_{\mathsf{list}} \Longrightarrow \mathcal{S}_{\mathsf{list}}$, respectively. Then we sum the numeric parts and collect all the data in the new cost–size tuple $\langle (n + m + k, h), f^{\mathsf{s}}(x^{\mathsf{s}}) \rangle$. This process is summarized in the following definition.

Definition 5 (Semantic Application) Given $\boldsymbol{f} = \langle (n, f^{\mathsf{c}}), f^{\mathsf{s}} \rangle \in (\!|\iota \Rightarrow \tau|\!)$ and $\boldsymbol{x} = \langle (m, \mathfrak{U}), x^{\mathsf{s}} \rangle \in (\!|\iota|\!)$, the *semantic application* of $\boldsymbol{f}$ to $\boldsymbol{x}$, denoted by $\boldsymbol{f} \cdot \boldsymbol{x}$, is $\langle (n + m + k, h), f^{\mathsf{s}}(x^{\mathsf{s}}) \rangle$ where $f^{\mathsf{c}}(x^{\mathsf{s}}) = (k, h)$.

Semantic application is left-associative, so $\boldsymbol{f} \cdot \boldsymbol{g} \cdot \boldsymbol{h}$ stands for $(\boldsymbol{f} \cdot \boldsymbol{g}) \cdot \boldsymbol{h}$. This definition conforms to the types, which is stated in the following lemma.

Lemma 3 *If $\boldsymbol{f} \in (\!|\iota \Rightarrow \tau|\!)$ and $\boldsymbol{x} \in (\!|\iota|\!)$, then $\boldsymbol{f} \cdot \boldsymbol{x} \in (\!|\tau|\!)$.*

Remark 1 Because $\mathbb{N} \times \mathtt{unit}$ is order-isomorphic to $\mathbb{N}$, we identify $\mathbb{N} \times \mathtt{unit}$ with $\mathbb{N}$ and $(m, \mathfrak{U})$ with m unless otherwise stated. So we write $\langle m, x^{\mathsf{s}} \rangle$ for cost–size tuples in $(\!|\iota|\!)$ where ι is a sort.

3.2 Cost–Size Tuple Algebras

Definition 6 A *cost–size tuple algebra* $(\langle\!|\cdot|\!\rangle, \mathcal{J})$ over a syntax signature $(\mathcal{B}, \mathcal{F}, \mathtt{ar})$ consists of

(i) a family of cost–size products $\{\langle\!|\sigma|\!\rangle\}_{\sigma \in \mathcal{T}_\mathcal{B}}$, and

(ii) an interpretation function $\mathcal{J} : \mathcal{F} \longrightarrow \biguplus_\sigma \langle\!|\sigma|\!\rangle$ which associates to each $\mathsf{f} : \sigma$ an element $\mathcal{J}_\mathsf{f} \in \langle\!|\sigma|\!\rangle$.

With innermost rewriting, we assume that variables have no cost.

Definition 7 Fix a cost–size tuple algebra $(\langle\!|\cdot|\!\rangle, \mathcal{J})$. A *valuation* $\alpha : \mathcal{X} \longrightarrow \biguplus_\sigma \langle\!|\sigma|\!\rangle$ is a function which maps each variable $x : \iota$ to a zero-cost tuple $\langle 0, x^\mathsf{s} \rangle \in \langle\!|\iota|\!\rangle$. The interpretation of a term s under valuation α, denoted by $[\![s]\!]^\mathcal{J}_\alpha$, is defined as follows:

$$[\![\mathsf{f}]\!]^\mathcal{J}_\alpha = \mathcal{J}_\mathsf{f} \qquad [\![x]\!]^\mathcal{J}_\alpha = \alpha(x) \qquad [\![s\ t]\!]^\mathcal{J}_\alpha = [\![s]\!]^\mathcal{J}_\alpha \cdot [\![t]\!]^\mathcal{J}_\alpha$$

As a corollary of Lemma 3, interpretation of terms conforms with types.

Lemma 4 *If $s : \sigma$ then $[\![s]\!]^\mathcal{J}_\alpha$ is in $\langle\!|\sigma|\!\rangle$, for all valuations α.*

Let σ be $\iota_1 \Rightarrow \ldots \Rightarrow \iota_m \Rightarrow \kappa$ where all ι_i and κ are sorts. Elements of $\mathcal{C}_\sigma$ can be written as

$$\begin{array}{l} (e_0, \boldsymbol{\lambda} x_1. \\ \qquad (e_1, \boldsymbol{\lambda} x_2. \\ \qquad\qquad \ldots \\ \qquad\qquad (e_{m-1}, \boldsymbol{\lambda} x_m. \\ \qquad\qquad\qquad (e_m, \mathfrak{U})) \ldots)). \end{array} \tag{1}$$

When $e_0 = e_1 = \cdots = e_{m-1} = 0$, we write $(\boldsymbol{\lambda} x_1 \ldots x_m . e_m)$ as a shorthand.

Example 2 Let $\mathcal{S}_\mathsf{nat}$ and $\mathcal{S}_\mathsf{list}$ be $\mathbb{N}$ and $\mathbb{N} \times \mathbb{N}$, respectively. Recall that the size of a natural number is the number of occurrences of s, and the size of a list is a pair $q = (q_\mathsf{l}, q_\mathsf{m})$ where q_l is the length and q_m is the maximum size of the elements. We interpret the constructors as follows:

$$\begin{array}{ll} \mathcal{J}_\mathsf{0} = \langle 0, 0 \rangle & \mathcal{J}_\mathsf{s} = \langle (\boldsymbol{\lambda} x.0), \boldsymbol{\lambda} x.x + 1 \rangle \\ \mathcal{J}_\mathsf{nil} = \langle 0, (0,0) \rangle & \mathcal{J}_\mathsf{cons} = \langle (\boldsymbol{\lambda} xq.0), \boldsymbol{\lambda} xq.(q_\mathsf{l} + 1, \max(x, q_\mathsf{m})) \rangle \end{array}$$

Both $\mathsf{0}$ and nil have no cost because they are constructors without a function type. With innermost rewriting, constructors with a function type, such as s and cons, have $e_0 = \cdots = e_m = 0$ for cost of form (1).

3.3 Compatibility Theorem

Roughly, the compatibility theorem (Theorem 1) states that if $\mathcal{R}$ is compatible with a tuple algebra $\mathcal{A}$, then the rewriting relation $\to^i_\mathcal{R}$ is embedded in the well-founded order on cost–size products. The next two lemmas are technical results needed in order to prove it. Lemma 5 states that interpretations are closed under substitution and Lemma 6 provides strong monotonicity to semantic application.

Definition 8 Fix a cost–size tuple algebra $(\langle\!|\cdot|\!\rangle, \mathcal{J})$. A substitution γ is *zero-cost* under valuation α if $[\![\gamma(x)]\!]^\mathcal{J}_\alpha$ is a zero-cost tuple for each variable x. Given a valuation α and a zero-cost substitution γ, the function $\alpha^\gamma = [\![\cdot]\!]^\mathcal{J}_\alpha \circ \gamma = [\![\gamma(\cdot)]\!]^\mathcal{J}_\alpha$ is thus a valuation.

Lemma 5 (Substitution) *If γ is a zero-cost substitution under valuation α, $[\![s\gamma]\!]^{\mathcal{J}}_{\alpha} = [\![s]\!]^{\mathcal{J}}_{\alpha^\gamma}$ for any term s.*

Lemma 6 $\mathtt{App}(\boldsymbol{f},\boldsymbol{x}) = \boldsymbol{f}\cdot\boldsymbol{x}$ *is strongly monotonic on both arguments.*

PROOF We need to prove (i) if $\boldsymbol{f} \succ \boldsymbol{g}$ and $\boldsymbol{x} \succcurlyeq \boldsymbol{y}$, then $\mathtt{App}(\boldsymbol{f},\boldsymbol{x}) \succ \mathtt{App}(\boldsymbol{g},\boldsymbol{y})$; (ii) if $\boldsymbol{f} \succcurlyeq \boldsymbol{g}$ and $\boldsymbol{x} \succ \boldsymbol{y}$, then $\mathtt{App}(\boldsymbol{f},\boldsymbol{x}) \succ \mathtt{App}(\boldsymbol{g},\boldsymbol{y})$; (iii) if $\boldsymbol{f} \succcurlyeq \boldsymbol{g}$ and $\boldsymbol{x} \succcurlyeq \boldsymbol{y}$, then $\mathtt{App}(\boldsymbol{f},\boldsymbol{x}) \succcurlyeq \mathtt{App}(\boldsymbol{g},\boldsymbol{y})$. Consider cost–size tuples $\boldsymbol{f},\boldsymbol{g} \in (\!|\iota \Rightarrow \tau|\!)$ and $\boldsymbol{x},\boldsymbol{y} \in (\!|\iota|\!)$. Let $\boldsymbol{f} = \langle (n, f^{\mathsf{c}}), f^{\mathsf{s}}\rangle$, $\boldsymbol{g} = \langle (m, g^{\mathsf{c}}), g^{\mathsf{s}}\rangle$, $\boldsymbol{x} = \langle x^{\mathsf{c}}, x^{\mathsf{s}}\rangle$, and $\boldsymbol{y} = \langle y^{\mathsf{c}}, y^{\mathsf{s}}\rangle$. We proceed to show (i) and observe that (ii) and (iii) follow similar reasoning. Indeed, if $\boldsymbol{f} \succ \boldsymbol{g}$ and $\boldsymbol{x} \succcurlyeq \boldsymbol{y}$ we have that $n > m$, $f^{\mathsf{c}} \geq g^{\mathsf{c}}$, $f^{\mathsf{s}} \sqsupseteq g^{\mathsf{s}}$, $x^{\mathsf{c}} \geq y^{\mathsf{c}}$, and $x^{\mathsf{s}} \sqsupseteq y^{\mathsf{s}}$. Hence, by letting $f^{\mathsf{c}}(x^{\mathsf{s}}) = (k,h)$ and $g^{\mathsf{c}}(y^{\mathsf{s}}) = (k',h')$, we get:

$$\mathtt{App}(\boldsymbol{f},\boldsymbol{x}) = \langle (n, f^{\mathsf{c}}), f^{\mathsf{s}}\rangle \cdot \langle x^{\mathsf{c}}, x^{\mathsf{s}}\rangle = \langle (n + x^{\mathsf{c}} + k, h), f^{\mathsf{s}}(x^{\mathsf{s}})\rangle > \left\langle (m + y^{\mathsf{c}} + k', h'), g^{\mathsf{s}}(y^{\mathsf{s}})\right\rangle = \mathtt{App}(\boldsymbol{g},\boldsymbol{y})$$

■

Definition 9 A TRS $\mathcal{R}$ is said to be *compatible* with a cost–size tuple algebra $((\!|\cdot|\!),\mathcal{J})$ if $[\![\ell]\!]^{\mathcal{J}}_{\alpha} \succ [\![r]\!]^{\mathcal{J}}_{\alpha}$ for all rules $\ell \to r \in \mathcal{R}$ and valuations α.

Theorem 1 (Compatibility) *Let $\mathcal{R}$ be a TRS compatible with a cost–size tuple algebra $((\!|\cdot|\!),\mathcal{J})$. Then, for any pair of terms s and t, whenever $s \to^{i}_{\mathcal{R}} t$ we have $[\![s]\!]^{\mathcal{J}}_{\alpha} \succ [\![t]\!]^{\mathcal{J}}_{\alpha}$.*

PROOF We proceed by induction on $\to^{i}_{\mathcal{R}}$. For the base case, $s \to^{i}_{\mathcal{R}} t$ by $\ell\gamma \to r\gamma$ and all subterms of $\ell\gamma$ are in $\to_{\mathcal{R}}$ normal form. Therefore, since $[\![\ell]\!]^{\mathcal{J}}_{\alpha} \succ [\![r]\!]^{\mathcal{J}}_{\alpha}$ by hypothesis, Lemma 5 gives us that $[\![\ell\gamma]\!]^{\mathcal{J}}_{\alpha} \succ [\![r\gamma]\!]^{\mathcal{J}}_{\alpha}$.

In the inductive step we use Lemma 6 combined with the (IH) as follows. Suppose $s \to^{i}_{\mathcal{R}} t$ by $s = s'u \to^{i}_{\mathcal{R}} s''u$ with $s' \to^{i}_{\mathcal{R}} s''$. Hence, $[\![s'u]\!]^{\mathcal{J}}_{\alpha} = [\![s']\!]^{\mathcal{J}}_{\alpha} \cdot [\![u]\!]^{\mathcal{J}}_{\alpha} = \mathtt{App}([\![s']\!]^{\mathcal{J}}_{\alpha}, [\![u]\!]^{\mathcal{J}}_{\alpha})$, henceforth the induction hypothesis gives $[\![s']\!]^{\mathcal{J}}_{\alpha} \succ [\![s'']\!]^{\mathcal{J}}_{\alpha}$, which combined with Lemma 6 implies $[\![s]\!]^{\mathcal{J}}_{\alpha} = \mathtt{App}([\![s']\!]^{\mathcal{J}}_{\alpha}, [\![u]\!]^{\mathcal{J}}_{\alpha}) \succ \mathtt{App}([\![s'']\!]^{\mathcal{J}}_{\alpha}, [\![u]\!]^{\mathcal{J}}_{\alpha}) = [\![t]\!]^{\mathcal{J}}_{\alpha}$. When $s \to^{i}_{\mathcal{R}} t$ with $s = s'\ u \to^{i}_{\mathcal{R}} s'\ u'$ the proof is analogous. ■

Example 3 Let $\mathsf{0},\mathsf{1} : \iota$, $\mathsf{g} : \iota \Rightarrow \iota \Rightarrow \iota$, and $\mathsf{f} : \iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota$. The rewrite system introduced by Toyama [22] and defined by $\mathcal{R} = \{\mathsf{g}\ x\ y \to x, \mathsf{g}\ x\ y \to y, \mathsf{f}\ \mathsf{0}\ \mathsf{1}\ z \to \mathsf{f}\ z\ z\ z\}$ was given to show that termination is not modular for disjoint unions of TRSs. Indeed, it admits the infinite rewriting sequence $\mathsf{f}\ \mathsf{0}\ \mathsf{1}\ (\mathsf{g}\ \mathsf{0}\ \mathsf{1}) \to_{\mathcal{R}} \mathsf{f}\ (\mathsf{g}\ \mathsf{0}\ \mathsf{1})\ (\mathsf{g}\ \mathsf{0}\ \mathsf{1})\ (\mathsf{g}\ \mathsf{0}\ \mathsf{1}) \to^{+}_{\mathcal{R}} \mathsf{f}\ \mathsf{0}\ \mathsf{1}\ (\mathsf{g}\ \mathsf{0}\ \mathsf{1})$. However, the innermost relation $\to^{i}_{\mathcal{R}}$ is terminating. In order to prove it, we introduce a non-numeric notion of size. Let $\mathcal{J}_{\mathcal{B}}(\iota) = \mathcal{P}(T(\mathcal{F},\mathcal{X}))$, i.e., the set of all subsets of $T(\mathcal{F},\mathcal{X})$. This set is partially ordered by set inclusion, so $x \sqsupseteq y$ iff $x \supseteq y$, which is a quasi-order. Consider the following interpretation:

$$\mathcal{J}_0 = \langle 0, \{0\}\rangle \qquad \mathcal{J}_1 = \langle 0, \{1\}\rangle \qquad \mathcal{J}_{\mathsf{g}} = \langle (\boldsymbol{\lambda} xy.1), \boldsymbol{\lambda} xy.x \cup y\rangle \qquad \mathcal{J}_{\mathsf{f}} = \langle (\boldsymbol{\lambda} xyz.H(x,y)), \boldsymbol{\lambda} xyz.\emptyset\rangle\,,$$

where H is a helper function defined by $H(x,y) = \mathtt{if}\ x \sqsupseteq \{0\} \wedge y \sqsupseteq \{1\}\ \mathtt{then}\ 1\ \mathtt{else}\ 0$. Notice that H is weakly monotonic and all terms in normal form are interpreted as sets of size ≤ 1. Checking compatibility is straightforward: $[\![\mathsf{g}\ x\ y]\!] = \langle 1, x \cup y\rangle \succ \langle 0, x\rangle = [\![x]\!]$ and $[\![\mathsf{g}\ x\ y]\!] = \langle 1, x \cup y\rangle \succ \langle 0, y\rangle = [\![y]\!]$; and $[\![\mathsf{f}\ \mathsf{0}\ \mathsf{1}\ z]\!] = \langle 1, \emptyset\rangle \succ \langle 0, \emptyset\rangle = [\![\mathsf{f}\ z\ z\ z]\!]$, because any instantiation of z is necessarily in normal form, so it cannot include both $\mathsf{0}$ and $\mathsf{1}$.

This example, albeit artificial, is interesting from a termination point of view. It shows that tuple interpretations can be used to deal with rewrite systems that only terminate via the innermost strategy.

4 Polynomial Bounds for Innermost Runtime Complexity

In this section, we study the applications of tuple interpretations to complexity analysis of compatible TRSs, i.e., rewriting systems that admit an interpretation in a tuple algebra $((\!|\cdot|\!),\mathcal{J})$. Even though cost and size are split in our setting, they are intertwined concepts (in a sense we make precise in this section) that constitute what we intuitively call "complexity" of a TRS.

4.1 Additive Tuple Interpretations

In order to establish upper bounds to $\mathtt{irc}_{\mathcal{R}}(n)$, it suffices to bound the cost component $[\![s]\!]^{\mathsf{c}}$ of all terms s where $|s| \leq n$. Furthermore, since basic terms are of the form $\mathsf{f}\ d_1 \dots d_m$, the size of data terms plays an important role in our analysis. In what follows, we use the default choice for interpretation key when interpreting types; that is, $\mathcal{J}_{\mathcal{B}}(\iota) = \mathbb{N}^{K[\iota]}$, with $K[\iota] \geq 1$ for each $\iota \in \mathcal{B}$.

Given $\sigma = \iota_1 \Rightarrow \ldots \Rightarrow \iota_m \Rightarrow \kappa$, the size component of $(\!|\sigma|\!)$ is $\mathcal{S}_\sigma = \mathbb{N}^{K[\iota_1]} \Longrightarrow \ldots \Longrightarrow \mathbb{N}^{K[\iota_m]} \Longrightarrow \mathbb{N}^{K[\kappa]}$. Size functions $f^{\mathsf{s}} \in \mathcal{S}_\sigma$ when fully applied can be written in terms of functional components. Hence, $f^{\mathsf{s}}(x_1,\dots,x_m) = \left\langle f^{\mathsf{s}}_1(x_1,\dots,x_m),\dots,f^{\mathsf{s}}_{K[\kappa]}(x_1,\dots,x_m)\right\rangle$.

Definition 10 Let σ be a type and $f^{\mathsf{s}} \in \mathcal{S}_\sigma$. The size function f^{s} is *linearly bounded* if each one of its component functions $f^{\mathsf{s}}_1,\dots,f^{\mathsf{s}}_{K[\kappa]}$ is upper-bounded by a positive linear polynomial, i.e., there is a positive constant $a \in \mathbb{N}$ such that for all $1 \leq l \leq m$, $f^{\mathsf{s}}_l(x_1,\dots,x_m) \leq a(1+\sum_{i=1}^{m}\sum_{j=1}^{K[\iota_i]} x_{ij})$. Analogously, we say f^{s} is *additive* if there is a constant $a \in \mathbb{N}$ such that $\sum_{l=1}^{K[\kappa]} f^{\mathsf{s}}_l(x_1,\dots,x_m) \leq a+\sum_{i=1}^{m}\sum_{j=1}^{K[\iota_i]} x_{ij}$.

Notice that by this definition linearly bounded (or additive) size functions are not required to be linear (or additive) but to be upper-bounded by a linear (additive) function. So this permits us to use for instance $\min(x,2y)$, whereas xy cannot be used. Size interpretations do not necessarily bound the absolute size of data terms. For instance, we may interpret a data constructor $\mathsf{c}\colon \iota \Rightarrow \kappa$ with $\mathcal{J}^{\mathsf{s}}_{\mathsf{c}} = \boldsymbol{\lambda} x.\lfloor x/2 \rfloor$ which would give us $|d| \geq [\![d]\!]^s$. This is specially useful when dealing with sublinear interpretations.

The next lemma ensures that by interpreting constructors additively the size interpretation of data terms is proportional to their absolute size:

Lemma 7 *Let $\mathcal{R}$ be a TRS compatible with a cost–size tuple algebra $(\!|\cdot|\!),\mathcal{J})$.*

(i) Assume $\mathcal{J}^{\mathsf{s}}_{\mathsf{c}}$ is additive for all data constructors c, then for all data terms d: if $|d| \leq n$, then there exists a constant $b > 0$ such that $[\![d]\!]^{\mathsf{s}}_l \leq bn$, for each size-component $[\![d]\!]^{\mathsf{s}}_l$ of $[\![d]\!]$.

(ii) Assume $\mathcal{J}^{\mathsf{s}}_{\mathsf{c}}$ is linearly bounded for all data constructors c, then for all data terms d: if $|d| \leq n$, then there exists a constant $b > 0$ such that $[\![d]\!]^{\mathsf{s}}_l \leq 2^{bn}$, for each size-component $[\![d]\!]^{\mathsf{s}}_l$ of $[\![d]\!]$.

The bound in (ii) is sharp. Indeed, define (when interpreting $\mathcal{R}_{\mathtt{add}}$): $\mathcal{J}_0 = \langle 0,1\rangle$, $\mathcal{J}_{\mathsf{s}} = \langle(\boldsymbol{\lambda} x.0),\boldsymbol{\lambda} x.2x+1\rangle$, and $\mathcal{J}_{\mathtt{add}} = \langle(\boldsymbol{\lambda} xy.y+1),\boldsymbol{\lambda} xy.x+y\rangle$. In this case, for a data term $\mathsf{n} = \mathsf{s}^n(0)$ its size interpretation is exactly $[\![\mathsf{n}]\!]^{\mathsf{s}} = 2^n + n \leq 2^{|\mathsf{n}|}$. However, whereas this choice is compatible with $\mathcal{R}_{\mathtt{add}}$, and hence proving its termination, it induces an exponential overhead on $\mathtt{irc}_{\mathcal{R}_{\mathtt{add}}}$, which is linearly bounded (see Example 4). Such a huge overestimation is not desirable in a complexity analysis setting. This behavior sets a strict upper-bound to the interpretation of data constructors; namely, we seek to bound constructor's size interpretations additively. It is easy to show that size components for nat and list in Example 2 are additive.

Definition 11 We say an interpretation $\mathcal{J}$ is additive if for each $\mathsf{c} \in \mathcal{C}$, $\mathcal{J}^{\mathsf{s}}_{\mathsf{c}}$ is additive.

4.2 Cost-Bounded Tuple Interpretations

In what follows, we consider rewriting systems with additive interpretations.

Definition 12 Let σ be a type and $\boldsymbol{f}^{\mathsf{c}} \in \mathcal{C}_\sigma$. We say $\boldsymbol{f}^{\mathsf{c}}$, written as in form (1), is linearly (additively) bounded whenever each e_i, $0 \leq i \leq m$, is linearly (additively) bounded. Additionally, $\mathcal{J}_{\mathsf{f}}$ is bounded by a functional f if both $\mathcal{J}^{\mathsf{c}}_{\mathsf{f}}$ and $\mathcal{J}^{\mathsf{s}}_{\mathsf{f}}$ are bounded by f.

In the next lemma, we collect the appropriate induced upper-bounds on innermost runtime complexity given that we can provide bounds to the cost–size components of interpretations.

Lemma 8 *Suppose $\mathcal{R}$ is a TRS compatible with a tuple algebra $(\!|\cdot|\!), \mathcal{J})$, then:*

(i) if, for all $\mathsf{f} \in \mathcal{F}$, $\mathcal{J}_\mathsf{f}^\mathsf{s}$ is logarithmically and $\mathcal{J}_\mathsf{f}^\mathsf{c}$ is additively bounded, then $\mathtt{irc}_\mathcal{R}(n) \in \mathcal{O}(\log n)$;

(ii) if, for all $\mathsf{f} \in \mathcal{F}$, $\mathcal{J}_\mathsf{f}$ is additively bounded, then $\mathtt{irc}_\mathcal{R}(n) \in \mathcal{O}(n)$; and

(iii) if, for all defined symbols f and constructors c, $\mathcal{J}_\mathsf{c}$ is additively and $\mathcal{J}_\mathsf{f}$ is polynomially bounded, then $\mathtt{irc}_\mathcal{R}(n) \in \mathcal{O}\left(n^k\right)$, for some $k \in \mathbb{N}$.

Example 4 Let us illustrate this behavior by interpreting functions from Example 1. Interpretation for constructors were given in Example 2.

$$\begin{array}{lcllcl}
\mathcal{J}_{\mathtt{add}} & = & \langle(\boldsymbol{\lambda} xy.y+1), \boldsymbol{\lambda} xy.x+y\rangle & \mathcal{J}_{\mathtt{sum}} & = & \langle(\boldsymbol{\lambda} q.2q_\mathsf{l}+q_\mathsf{l}q_\mathsf{m}), \boldsymbol{\lambda} q.q_\mathsf{l}q_\mathsf{m}\rangle \\
\mathcal{J}_{\mathtt{minus}} & = & \langle(\boldsymbol{\lambda} xy.y+1), \boldsymbol{\lambda} xy.x\rangle & \mathcal{J}_{\mathtt{rev}} & = & \left\langle(\boldsymbol{\lambda} q.q_\mathsf{l}+\frac{q_\mathsf{l}(q_\mathsf{l}+1)}{2}+1), \boldsymbol{\lambda} q.q\right\rangle \\
\mathcal{J}_{\mathtt{quot}} & = & \langle(\boldsymbol{\lambda} xy.x+xy+1), \boldsymbol{\lambda} xy.x\rangle & & & \\
\mathcal{J}_{\mathtt{append}} & = & \langle(\boldsymbol{\lambda} ql.q_\mathsf{l}+1), \boldsymbol{\lambda} ql.\langle q_\mathsf{l}+l_\mathsf{l}, \max(q_\mathsf{m}, q_\mathsf{m})\rangle\rangle & & &
\end{array}$$

Checking compatibility of this interpretation is straightforward. Notice that in each set of rules defining a function f in Example 1 size components are additively and cost components are polynomially bounded. By case (b) of Lemma 8, we have that $\mathtt{irc}_{\mathcal{R}_{\mathtt{add}}}$, $\mathtt{irc}_{\mathcal{R}_{\mathtt{append}}}$, and $\mathtt{irc}_{\mathcal{R}_{\mathtt{minus}}}$ are linear. Quadratic bounds can be derived to $\mathtt{irc}_{\mathcal{R}_{\mathtt{quot}}}$, $\mathtt{irc}_{\mathcal{R}_{\mathtt{sum}}}$, and $\mathtt{irc}_{\mathcal{R}_{\mathtt{rev}}}$.

Recall the semantic meaning given to size components, see Example 2, one can observe that the cost component of interpretations do not only bound the innermost runtime complexity of $\mathcal{R}_\mathsf{f}$ but also provide additional information on the role each size component plays in the reduction cost. For instance: the cost of adding two numbers depends solely on the size of `add`'s second argument; the cost of summing every element of a list has a linear dependency on its length and non-linear dependency on its length and maximum element. This is particularly useful in program analysis since one can detect a possible costly operation by analyzing the shape of interpretations themselves.

5 Automation

In this section, we limn an automation technique implementing a procedure to search for cost–size tuple interpretations.

A Pseudo-procedure to Search for Cost–Size Tuples. As it might be expected, tuple interpretations do not provide a complete proof method: there are innermost terminating systems that cannot be oriented. Nevertheless, it has the potential to be very powerful — if we choose the interpretation key $\mathcal{J}_\mathcal{B}$ right. Intuitively, we begin the search by assigning a measure of size to each sort, that is the number $K[\iota]$. In a fully automated setting, where no human input is allowed, all sorts ι start with $K[\iota] = 1$ and go up to a pre-determined bound K.

Main Procedure
Input: A syntax signature $(\mathcal{B}, \mathcal{F}, \mathtt{ar})$ for a TRS $\mathcal{R}$ together with its set of rules $\ell_i \to r_i$.
Output: YES (together with a representation of the cost–size tuple found), if a cost–size tuple could be found; MAYBE, if all steps below where executed and no interpretation could be found[1].

1. Split the signature into two disjoint sets of constructors and defined symbols, i.e., $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$.

[1] Notice that with only this approach we cannot possibly return NO.

2. For each constructor $\mathsf{c} : \iota_1 \Rightarrow \ldots \Rightarrow \iota_m \Rightarrow \kappa$, choose its cost interpretation as the zero-valued cost function $(\boldsymbol{\lambda} x_1 \ldots x_m.0)$; size interpretations are additive.
3. Split $\mathcal{D}$ into sets $\mathcal{D}_1, \ldots, \mathcal{D}_k$ such that for each $\mathsf{f} \in \mathcal{D}_i$, all function symbols occurring in the rules defining f are either constructors or in $\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_i$.
4. For all $1 \leq i \leq n$, choose an *interpretation shape* for the symbols in $\mathcal{D}_i$ based on the selector strategy $\mathcal{S}$ (to be defined below). Remove the associated choice from $\mathcal{S}$ parametrized by f.
   - If no choice can be made by $\mathcal{S}$ stop and return MAYBE.
5. *Simplify* $[\![\ell]\!] \succ [\![r]\!]$ so that the result is a set of order constraints that does not depend on any interpreted variable (we shall define this simplification step below).
6. *Check* if C holds.
   - If all constraints in C hold, then return YES.
   - Otherwise, increase $K[\iota]$ by one, update the additive size interpretation for the constructors, and return to step 4 choosing another interpretation shape.

Strategy-based Search for Tuple Interpretations. Two key aspects of the previous procedure remain to be defined. The strategy $\mathcal{S}$ for selecting interpretation shapes and the *check* command in step 6. Intuitively, a strategy is a procedure that implement choices for interpreting defined symbols in $\mathcal{D}_i$. For instance, we could randomly pick an interpretation shape from a list (the **blind** strategy); we could incrementally select interpretations from a list of possible attempts (the **progressive** strategy); or we could select interpretations based on their syntax patterns (the **pattern** strategy). The *check* procedure depends very much on the type of interpretations and which class of weakly monotonic functions is allowed. We illustrate each strategy and checker in the case where interpretations are polynomials and max-polynomials.

Definition 13 (Interpretation Shapes) Let $\sigma = \iota_1 \Rightarrow \ldots \Rightarrow \iota_m \Rightarrow \kappa$, and each f_{ij} appearing in the shapes below is a additively bounded weakly monotonic function over $\mathcal{S}_\sigma$.

- The additive class of interpretations contains additively bounded cost–size functionals of the following form:

$$\boldsymbol{\lambda} x_1 \ldots x_m. \sum_{i=1}^{m} \sum_{j=1}^{K[\iota_i]} x_{ij} + b_0 + b_1 f_1(x_1, \ldots, x_m) + \cdots + b_N f_N(x_1, \ldots, x_m);$$

- The linear class contains cost–size functionals written as:

$$\boldsymbol{\lambda} x_1 \ldots x_m. \sum_{i=1}^{m} \sum_{j=1}^{K[\iota_i]} a_{ij} f_{ij}(x_1, \ldots, x_m) x_{ij};$$

- The simple class contains cost–size functionals written as:

$$\boldsymbol{\lambda} x_1 \ldots x_m. \sum_{i_j \in \{0,1\}} a_{i_1}, \ldots, a_{i_m} f_{i_1}(\vec{x}), \ldots, f_{i_m}(\vec{x}) x_1^{i_1} \cdots x_m^{i_m};$$

- The simple quadratic class contains cost–size functionals written as a sum of a simple functional plus a quadratic component:

$$\boldsymbol{\lambda} x_1 \ldots x_m. \sum_{i_j \in \{0,1\}} a_{i_1} \ldots a_{i_m} f_{i_1}(\vec{x}) \ldots f_{i_m}(\vec{x}) x_1^{i_1} \cdots x_m^{i_m} + \sum_{1 \leq i \leq m} b_i x_i^2;$$

- and finally, the quadratic class contains cost–size functions where we allow general product of variables with degree at maximum 2:

$$\boldsymbol{\lambda} x_1 \dots x_m. \sum_{i_j \in \{0,1,2\}} a_{i_1}, \dots, a_{i_m} f_{i_1}(\vec{x}) \dots f_{i_m}(\vec{x}) x_1^{i_1} \cdots x_m^{i_m}.$$

An interpretation is a max-polynomial if its functional coefficients are composed of compositions of max functions only.

Hence, the blind strategy randomly selects one of the shapes above. The incremental strategy chooses interpretations in order, starting from additive up to quadratic. The pattern strategy is slightly more difficult to realize since we need heuristic analysis on the shape of rules. For instance, every rule of the form $\mathsf{f}\, x_1 \dots x_m \to x_i$ have constant cost functions ($\boldsymbol{\lambda} x_1 \dots x_m.1$) and additive size components. Rules that copy variables, i.e., follow the pattern $C[x] \to D[x,x]$, induce at least quadratic bound on cost. Notice that this is the case for all quadratic complexities in this paper.

In order to simplify constraints $[\![\ell]\!] \succ [\![r]\!]$ we have to simplify inequalities between polynomials (max-polynomials). To simplify polynomial (max-polynomial) shapes, we need to compare polynomials $P^{\mathsf{c}}_{\ell} > R^{\mathsf{c}}_{r}$ and $P^{\mathsf{s}}_{\ell_1} \sqsupseteq P^{\mathsf{s}}_{r_1} \wedge \dots \wedge P^{\mathsf{s}}_{\ell_{K[\tau]}} \sqsupseteq P^{\mathsf{s}}_{r_{K[\kappa]}}$. These conditions are then reduced to formulas in QF_NIA (Quantifier-Free Non-Linear Integer Arithmetic) and sent to an SMT solver, see [9]. Max polynomials are simplified using the rules $\max(x,y)+z \rightsquigarrow \max(x+z,y+z)$ and $\max(x,y)z \rightsquigarrow \max(xz,yz)$. The result has the form $\max_l P_l$ where each P_l is a polynomial without max occurrences [7].

Work-in-progress Implementation. Parallel to the theoretical development, we are working on implementing the essential structural codebase to run the **Main Procedure** above. Currently, this is still a prototype, in which we can handle simple interpretation shapes such as additive and linear. See [10] for more details.

6 Conclusion

In this paper we showed that cost–size tuple pairs can be adapted to handle innermost rewriting. The type-aware algebraic interpretation style provided the machinery necessary to deal with innermost termination and a mechanism to bound the innermost runtime complexity of compatible TRSs. We presented sufficient conditions for feasible (polynomial) bounds on $\mathtt{irc}_{\mathcal{R}}$ of compatible systems, which are in line with related works on the literature. This line of investigation is far from over. Since searching for interpretations can be cumbersome, our immediate future work is to develop new strategies and interpretation shapes. This has the potential to drastically improve the efficiency of our prototype tool.

Acknowledgments. We wish to thank Cynthia Kop — for the valuable discussions and guidance during the production of this paper; we thank Niels van der Weide, Marcos Bueno, and Edna Gomes — for carefully proofread the various manuscript versions of the paper; and we thank the anonymous referees — for the comments that helped improve the paper.

References

[1] M. Avanzini & G. Moser (2008): *Complexity Analysis by Rewriting*. In: *Proc. FLOPS*, doi:10.1007/978-3-540-78969-7_11.

[2] P. Baillot & U. Dal Lago (2016): *Higher-order interpretations and program complexity*. *IC*, doi:10.1016/j.ic.2015.12.008.

[3] A. Ben Cherifa & P. Lescanne (1987): *Termination of rewriting systems by polynomial interpretations and its implementation*. *Science of Computer Programming* 9(2), pp. 137–159, doi:https://doi.org/10.1016/0167-6423(87)90030-X.

[4] G. Bonfante, A. Cichon, J.-Y. Marion & H. Touzet (2001): *Algorithms with polynomial interpretation termination proof*. *Journal of Functional Programming* 11(1), p. 33–53, doi:10.1017/S0956796800003877.

[5] G. Bonfante, J. Marion & J. Moyen (2001): *On Lexicographic Termination Ordering with Space Bound Certifications*. In: *Proc. PSI*, doi:10.1007/3-540-45575-2_46.

[6] A. Cichon & P. Lescanne (1992): *Polynomial interpretations and the complexity of algorithms*. In: *CADE*, pp. 139–147, doi:10.1007/3-540-55602-8_161.

[7] M. Codish, I. Gonopolskiy, A. M. Ben-Amram, C. Fuhs & J. Giesl (2011): *SAT-based termination analysis using monotonicity constraints over the integers*. *Theory and Practice of Logic Programming* 11(4-5), p. 503–520, doi:https://doi.org10.1017/S1471068411000147.

[8] E. Contejan, C. Marché, A. P. Tomás & X. Urbain (2005): *Mechanically Proving Termination Using Polynomial Interpretations*. *JAR* (34), doi:10.1007/s10817-005-9022-x.

[9] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plucker, P. Schneider-Kamp, T. Stroder, S. Swiderski & R. Thiemann (2017): *Analyzing Program Termination and Complexity Automatically with AProVE*. *JAR* (58), pp. 3–31, doi:10.1007/s10817-016-9388-y.

[10] L. Guo & D. Vale (2022): *Hermes: Innermost Runtime Complexity Analysis Tool*. Software Repository. Available at `https://github.com/deividrvale/hermes`.

[11] N. Hirokawa & G. Moser (2008): *Automated Complexity Analysis Based on the Dependency Pair Method*. In: *Proc. IJCAR*, doi:10.1007/978-3-540-71070-7_32.

[12] D. Hofbauer (1992): *Termination proofs by multiset path orderings imply primitive recursive derivation lengths*. *Proc. TCS*, doi:10.1007/3-540-53162-9_50.

[13] D. Hofbauer (2001): *Termination Proofs by Context-Dependent Interpretations*. In: *Proc. RTA*, doi:10.1007/3-540-45127-7_10.

[14] D. Hofbauer & C. Lautemann (1989): *Termination proofs and the length of derivations*. In: *Proc. RTA*, doi:10.1007/3-540-51081-8_107.

[15] G. Huet & D.C Oppen (1980): *Equations and rewrite rules: a survey*. Formal Language Theory: Perspectives and Open Problems. Available at `http://rewriting.loria.fr/documents/CS-TR-80-785.pdf`.

[16] C. Kop & D. Vale (2021): *Tuple Interpretations for Higher-Order Complexity*. In: *FSCD*, pp. 31:1–31:22, doi:10.4230/LIPIcs.FSCD.2021.31.

[17] C. Lautemann (1988): *A note on polynomial interpretation*. *Bulletin EATCS* volume 4, pp. 129–131.

[18] F. Mitterwallner & A. Middeldorp (2022): *Polynomial Termination Over $\mathbb{N}$ Is Undecidable*. In: *Proc. FSCD*, pp. 27:1–27:17, doi:https://doi.org/10.4230/LIPIcs.FSCD.2022.27.

[19] G. Moser (2017): *Uniform Resource Analysis by Rewriting: Strengths and Weaknesses (Invited Talk)*. In: *Proc. FSCD*, pp. 2:1–2:10, doi:10.4230/LIPIcs.FSCD.2017.2.

[20] G. Moser, A. Schnabl & J. Waldmann (2008): *Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations*. In: *Proc. IARCS*, pp. 304–315, doi:10.4230/LIPIcs.FSTTCS.2008.1762.

[21] L. Noschinski, F. Emmes & J. Giesl (2011): *A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems*. In: *CADE-23*, pp. 422–438, doi:10.1007/978-3-642-22438-6_32.

[22] Yoshihito T. (1987): *Counterexamples to termination for the direct sum of term rewriting systems*. *Information Processing Letters* 25(3), pp. 141–143, doi:https://doi.org/10.1016/0020-0190(87)90122-0.

[23] A. Weiermann (1995): *Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths*. *TCS*, doi:10.1016/0304-3975(94)00135-6.

[24] A. Yamada (2022): *Tuple Interpretations for Termination of Term Rewriting*. J Autom Reasoning, doi:https://doi.org/10.1007s10817-022-09640-4.