Tuple Interpretations for Higher-Order Complexity

Cynthia Kop
Department of Software Science, Radboud University Nijmegen, The Netherlands

Deivid Vale
Department of Software Science, Radboud University Nijmegen, The Netherlands

Abstract

We present a style of algebra interpretations for many-sorted and higher-order term rewriting based on interpretations to *tuples*; intuitively, a term is mapped to a sequence of values identifying for instance its evaluation cost, size and perhaps other values. This could give a more fine-grained notion of the complexity of a term or TRS than notions such as runtime or derivational complexity.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Complexity, higher-order term rewriting, many-sorted term rewriting, polynomial interpretations, weakly monotonic algebras

Funding The authors are supported by the NWO TOP project "ICHOR", NWO 612.001.803/7571.

1 Introduction

In the study of complexity of term rewriting systems, it is common to consider termination techniques: if a TRS can be proved terminating by a certain technique, this typically implies a specific bound on the number of steps that may be needed to reduce a term in that TRS to normal form (see, e.g., [2, 4, 5, 7]). Some approaches (e.g., [5, 7]) consider *interpretations* of terms s. Interpretations impose a natural bound on reduction length for given terms.

By their nature, interpretations to natural numbers do not tend to give *tight* bounds. Consider for example the term rewriting system implementing addition, which is given by the rules $\mathsf{add}(x, \mathsf{0}) \to x$ and $\mathsf{add}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{add}(x, y))$. An interpretation would need to be monotonic, and have $[\![\ell]\!] > [\![r]\!]$ for both rules. This leads to for instance: $[\![\mathsf{0}]\!] = 0$, $[\![\mathsf{s}(x)]\!] = [\![x]\!] + 1$ and $[\![\mathsf{add}(x, y)]\!] = [\![x]\!] + 2 \cdot [\![y]\!] + 1$. With these choices, we indeed have:

$$\begin{array}{rclclcl} [\![\mathsf{add}(x,\mathsf{0})]\!] & = & [\![x]\!] + 1 & > & [\![x]\!] & = & [\![x]\!] \\ [\![\mathsf{add}(x,\mathsf{s}(y))]\!] & = & [\![x]\!] + 2 \cdot [\![y]\!] + 2 & > & [\![x]\!] + 2 \cdot [\![y]\!] + 1 & = & [\![\mathsf{s}(\mathsf{add}(x,y))]\!] \end{array}$$

But $[\![\mathsf{add}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))]\!] = n + 2m + 1$, even though only $n + m + 1$ steps can be done before reaching normal form. This is because the interpretation captures not only the reduction cost, but also the size of the normal form. This is not problematic for the example above, because the result is still *linear* runtime complexity. However, for *exponential* bounds, the consequences are more severe: consider $\mathcal{O}(2^n)$ versus $\mathcal{O}(2^{3n}) = \mathcal{O}(8^n)$. And particularly when considering higher-order term rewriting, exponential bounds are often very relevant.

The situation could be improved by splitting interpretations into separate *cost* and *size* components, as was done for conditional rewriting in [6]. For instance, in the example above we could take $[\![\mathsf{add}(x,y)]\!]_{\mathtt{size}} = [\![x]\!]_{\mathtt{size}} + [\![y]\!]_{\mathtt{size}}$ and $[\![\mathsf{add}(x,y)]\!]_{\mathtt{cost}} = [\![x]\!]_{\mathtt{cost}} + [\![y]\!]_{\mathtt{cost}} + [\![y]\!]_{\mathtt{size}} + 1$. More generally, we could interpret terms to tuples of arbitrary size. This essentially generalises matrix interpretations [7] as well, by mapping terms to a vector but not imposing restrictions on the shape of interpretation functions. This could be particularly useful for many-sorted and higher-order term rewriting systems, where the choice of tuple length may be type-dependent.

The present short paper explores the ideas above. It documents work in progress with the aim to help establish a more fine-grained notion of complexity for term rewriting—which captures both time, space and perhaps other properties such as the shape of normal forms. The technique we develop may also be useful for resource analysis of higher-order programs.

2 Preliminaries: many-sorted and higher-order term rewriting

We assume familiarity with first-order term rewriting. In *many-sorted* rewriting, all function symbols have a sequence of input sorts, and an output sort; and terms must be well-typed.

▶ **Example 1.** The TRS $\mathcal{R}_+$, for arithmetic and lists, has six function symbols: $0 :: \mathsf{nat}$, $\mathsf{nil} :: \mathsf{list}$, $\mathsf{s} :: \mathsf{nat} \Longrightarrow \mathsf{nat}$, $\mathsf{add} :: \mathsf{nat} \times \mathsf{nat} \Longrightarrow \mathsf{nat}$, $\mathsf{mult} :: \mathsf{nat} \times \mathsf{nat} \Longrightarrow \mathsf{nat}$, $\mathsf{dList} :: \mathsf{list} \to \mathsf{list}$, and $\mathsf{cons} :: \mathsf{nat} \times \mathsf{natlist} \to \mathsf{natlist}$. It is given by rules of sort nat and list, as follows:

$$\begin{aligned} \mathsf{add}(x, 0) &\to x & \mathsf{d}(0) &\to 0 \\ \mathsf{add}(x, \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{add}(x, y)) & \mathsf{d}(\mathsf{s}(x)) &\to \mathsf{s}(\mathsf{s}(\mathsf{d}(x))) \\ \mathsf{mult}(x, 0) &\to 0 & \mathsf{dList}(\mathsf{nil}) &\to \mathsf{nil} \\ \mathsf{mult}(x, \mathsf{s}(y)) &\to \mathsf{add}(x, \mathsf{mult}(x, y)) & \mathsf{dList}(\mathsf{cons}(x, q)) &\to \mathsf{cons}(\mathsf{d}(x), \mathsf{dList}(q)) \end{aligned}$$

For *higher-order* rewriting, we use a formalism where function symbols take a sequence of *simple types* as input (i.e., generated from a set $\mathcal{B}$ of sorts and a right-associative binary type constructor $\Rightarrow$) and a sort as output; term formation allows for function application ($\mathsf{f}(s_1, \dots, s_m) : \iota$ if $\mathsf{f} : \sigma_1 \times \cdots \times \sigma_m \Longrightarrow \iota$ is a symbol and each $s_i : \sigma_i$), as well as application (i.e., if $s : \sigma \Rightarrow \tau$ and $t : \sigma$ than $s\ t : \tau$) and λ-abstraction as in the simply-typed λ-calculus. The β-reduction rule $(\lambda x.s)\ t \to s[x := t]$ is always included in the reduction relation $\to_{\mathcal{R}}$.

▶ **Example 2.** Let $\mathcal{R}_{\mathtt{fold}}$ be the higher-order TRS with symbols $\mathsf{nil} :: \mathsf{list}$, $\mathsf{cons} :: \mathsf{nat} \times \mathsf{list} \Longrightarrow \mathsf{list}$, $\mathsf{map} :: (\mathsf{nat} \Rightarrow \mathsf{nat}) \times \mathsf{list} \Longrightarrow \mathsf{list}$ and $\mathsf{foldl} :: (\mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}) \times \mathsf{nat} \times \mathsf{list} \Longrightarrow \mathsf{nat}$ and rules:

$$\begin{aligned} \mathsf{foldl}(f, z, \mathsf{nil}) &\to z & \mathsf{map}(f, \mathsf{nil}) &\to \mathsf{nil} \\ \mathsf{foldl}(f, z, \mathsf{cons}(x, q)) &\to \mathsf{foldl}(f, (f\ z\ x), q) & \mathsf{map}(f, \mathsf{cons}(x, q)) &\to \mathsf{cons}(f\ x, \mathsf{map}(f, q)) \end{aligned}$$

3 First-Order type-based interpretation

It is common in the rewriting literature to use termination proofs to assess the difficulty of rewriting a term to a normal form [2, 5]. For example, in [5], Hofbauer gives a first upper-bound for the derivational complexity of first-order TRS's with polynomial interpretation as termination proofs. This technique has been extended to other termination proofs as well [2]. Polynomial interpretations are a form of *algebra interpretations*:

▶ **Definition 3** (adapted from [8]). *An* algebra $\mathcal{A}$ *for many-sorted first-order terms consists of a mapping from each sort* $\iota \in \mathcal{B}$ *to a well-founded set* $(A_\iota, >_\iota, \geq_\iota)$ *together with an interpretation function* $\mathcal{J}$ *which assigns to each* $\mathsf{f} :: \iota_1 \times \cdots \times \iota_m \Longrightarrow \kappa \in \mathcal{F}$ *a monotonic function* $\mathcal{J}_\mathsf{f} \in A_{\iota_1} \to \cdots \to A_{\iota_m} \to A_\kappa$ *(monotonic:* $\mathcal{J}_\mathsf{f}(\dots, x, \dots) > \mathcal{J}_\mathsf{f}(\dots, y, \dots)$ *if* $x > y$*).*

If α *is a mapping from variables of sort* ι *to* A_ι*, term interpretation is defined recursively with* $[\![x]\!]^{\mathcal{J}}_\alpha = \alpha(x)$ *and* $[\![\mathsf{f}(s_1, \dots, s_m)]\!]^{\mathcal{J}}_\alpha = \mathcal{J}_\mathsf{f}([\![s_1]\!]^{\mathcal{J}}_\alpha, \dots, [\![s_m]\!]^{\mathcal{J}}_\alpha)$*. We usually omit* α *and* $\mathcal{J}$ *and just write* $[\![s]\!]$*. Termination follows if* $[\![\ell]\!]^{\mathcal{J}}_\alpha > [\![r]\!]^{\mathcal{J}}_\alpha$ *for all* α *and a fixed* $\mathcal{J}$*.*

If each $A_\iota = \mathbb{N}$, then $[\![s]\!]$ gives a worst-case boundary on the number of rewriting steps starting from s (as observed in the introduction); this can be used to bound the number of steps starting from an arbitrary term of size n, depending on the shape of the interpretation.

As an alternative, we consider interpretations with $A_\iota = \mathbb{N}^{K_\iota}$. We let $(n_1, \dots, n_{K_\iota}) \geq (n'_1, \dots, n'_{K_\iota})$ if each $n_i \geq n'_i$, and $(n_1, \dots, n_{K_\iota}) > (n'_1, \dots, n'_{K_\iota})$ if $n_1 > n'_1$ and each $n_i \geq n'_i$. For example, we let $A_{\mathsf{nat}} = \mathbb{N}^2$ and $A_{\mathsf{list}} = \mathbb{N}^3$. Intuitively, the first component in both cases indicates "cost": the number of steps needed to reduce a term to normal form. The second component of A_{nat} represents the size of the natural number, and the second and third component of A_{list} represent the list length and maximum element size respectively.

▶ **Example 4.** Consider the signature of Example 1. We set its interpretation as follows below, where s_{c} is syntactic sugar for $[\![s]\!]_1$ (the cost component of s), s_{s} and s_{l} are $[\![s]\!]_2$ (the size or length component) and s_{m} is $[\![s]\!]_3$ (the component for maximum element size).

$$\begin{array}{rclrcl} [\![\mathsf{0}]\!] & = & \langle 0, 1\rangle & [\![\mathsf{nil}]\!] & = & \langle 0,0,0\rangle \\ [\![\mathsf{s}(x)]\!] & = & \langle x_{\mathsf{c}}, x_{\mathsf{s}}+1\rangle & [\![\mathsf{cons}(x,q)]\!] & = & \langle x_{\mathsf{c}}+q_{\mathsf{c}}, q_{\mathsf{l}}+1, \max(x_{\mathsf{s}},q_{\mathsf{m}})\rangle \\ [\![\mathsf{d}(x)]\!] & = & \langle 1+x_{\mathsf{c}}+x_{\mathsf{s}}, 2\cdot x_{\mathsf{s}}\rangle & [\![\mathsf{dList}(q)]\!] & = & \langle 1+q_{\mathsf{c}}+q_{\mathsf{l}}\cdot(2+q_{\mathsf{m}}), q_{\mathsf{l}}, 2\cdot q_{\mathsf{m}}\rangle \\ [\![\mathsf{add}(x,y)]\!] & = & \langle 1+x_{\mathsf{c}}+y_{\mathsf{c}}+y_{\mathsf{s}}, x_{\mathsf{s}}+y_{\mathsf{s}}\rangle & & & \\ [\![\mathsf{mult}(x,y)]\!] & = & \langle 1+x_{\mathsf{c}}+y_{\mathsf{c}}+x_{\mathsf{s}}\cdot(2+x_{\mathsf{c}}+x_{\mathsf{s}}\cdot y_{\mathsf{s}}), x_{\mathsf{s}}\cdot y_{\mathsf{s}}\rangle & & & \end{array}$$

We can easily check that $[\![\ell]\!] > [\![r]\!]$ for all rewrite rules $\ell \to r$; that is, there is a strict decrease in the "cost" component and a weak decrease (with $\geq$) in the others. For example:

$$\begin{array}{rl} & [\![\mathsf{dList}(\mathsf{cons}(x,q))]\!] \\ = & \langle 1+(x_{\mathsf{c}}+q_{\mathsf{c}})+(q_{\mathsf{l}}+1)\cdot(2+\max(x_{\mathsf{s}},q_{\mathsf{m}})), q_{\mathsf{l}}+1, 2\cdot\max(x_{\mathsf{s}},q_{\mathsf{m}})\rangle \\ = & \langle 3+x_{\mathsf{c}}+q_{\mathsf{c}}+\max(x_{\mathsf{s}},q_{\mathsf{m}})+q_{\mathsf{l}}\cdot(2+\max(x_{\mathsf{s}},q_{\mathsf{m}})), q_{\mathsf{l}}+1, 2\cdot\max(x_{\mathsf{s}},q_{\mathsf{m}})\rangle \\ > & \langle 2+x_{\mathsf{c}}+q_{\mathsf{c}}+x_{\mathsf{s}}+q_{\mathsf{l}}\cdot(2+q_{\mathsf{m}}), q_{\mathsf{l}}+1, \max(2\cdot x_{\mathsf{s}}, 2\cdot q_{\mathsf{m}})\rangle \\ = & \langle (1+x_{\mathsf{c}}+x_{\mathsf{s}})+(1+q_{\mathsf{c}}+q_{\mathsf{l}}\cdot(2+q_{\mathsf{m}})), q_{\mathsf{l}}+1, \max(2\cdot x_{\mathsf{s}}, 2\cdot q_{\mathsf{m}})\rangle \\ = & [\![\mathsf{cons}(\mathsf{d}(x),\mathsf{dList}(q))]\!] \end{array}$$

Note that our interpretation method has some similarities with matrix interpretations [3], as each term is associated to an n-tuple. However, the interpretation function is not restricted to linear multivariate polynomials, allowing interpretations such as those for cons and mult. Tuple interpretations give information on more than just the cost of evaluating a term.

▶ **Example 5** (Bounds for arithmetic)**.** We have $[\![\mathsf{dList}(\mathsf{cons}(\mathsf{s}^3(\mathsf{0}),\mathsf{cons}(\mathsf{s}^1(\mathsf{0}),\mathsf{nil})))]\!] = \langle 11,2,6\rangle$. Given the way the interpretation was constructed, this implies that an evaluation to normal form takes at most 11 steps, and the normal form has length at most 2 and a greatest element at most $s^6(0)$. The cost component is not tight: it only takes 8 steps to evaluate the term (11 is the maximum number of steps to evaluate $\mathsf{dList}(q)$ for *any* constructor-list q of length 2 and with greatest element $\mathsf{s}^3(\mathsf{0})$). The other two values are tight.

4 Higher-order type-based interpretations

In first-order term rewriting, the complexity of a TRS is often measured as *runtime* or *derivational* complexity: both measures are parametrised by the size of an initial term. This is not a good measure for terms with immediate subterms of higher-order type: the behaviour of such subterms on given arguments should be considered, as the next example shows.

▶ **Example 6.** Consider $\mathcal{R}_+ \cup \mathcal{R}_{\mathsf{foldl}}$. The evaluation cost of a term $\mathsf{foldl}(F,t,q)$ depends almost completely on the behaviour of the functional subterm F, and not only on its evaluation cost. If F is $\lambda x.\lambda y.\mathsf{d}(x)$—so a *size-increasing* term—evaluating $\mathsf{foldl}(F,t,q)$ takes exponentially many steps, even though d runs in linear steps and F is executed only $|q|$ times. Thus, higher-order rewriting in particular is a natural place to separate cost and size.

Algebra interpretations for higher-order rewriting were defined in [8]. Essentially, the interpretations of Definition 3 are extended by letting $\mathcal{A}_{\sigma\Rightarrow\tau}$ be the set of *weakly monotonic functions* from $\mathcal{A}_\sigma$ to $\mathcal{A}_\tau$ (that is, $f(\dots,x,\dots) \geq_\tau f(\dots,y,\dots)$ if $x \geq_\sigma y$), with $>_{\sigma\Rightarrow\tau}$ and $\geq_{\sigma\Rightarrow\tau}$ being point wise comparisons. While the author of [8] and followup work used $\mathbb{N}$ for $\mathcal{A}_\iota$ (with $\iota \in \mathcal{B}$), the method needs no modification when tuple interpretations are used instead. For elements of $\mathcal{A}_{\iota\Rightarrow\sigma}$, we moreover limit interest to functions f such that always $f(x_1,x_2\dots,x_n)_i = f(x'_1,x_2,\dots,x_n)_i$ for $i > 1$; that is, the size, length and "greatest element" components do not depend on the cost component (but may depend on each other).

▶ **Example 7.** Let $A_{\mathsf{nat}} = \mathbb{N}^2$ and $A_{\mathsf{list}} = \mathbb{N}^3$ as before, and assume cons and nil are interpreted as in Example 4. We can use the following interpretation for map:

$$[\![\mathsf{map}(f,q)]\!] = \langle \quad 1 + q_{\mathsf{c}} + 2 \cdot q_{\mathsf{l}} + (q_{\mathsf{l}} + 1) \cdot [\![f]\!](q_{\mathsf{c}}, q_{\mathsf{m}})_1, \quad q_{\mathsf{l}}, \quad [\![f]\!](q_{\mathsf{c}}, q_{\mathsf{m}})_2 \quad \rangle$$

This expresses that the list length is retained (as the length component is just q_{l}), the greatest element of the result map is bounded by the value of f on the greatest element of q, and the evaluation cost is mostly expressed by a linear number of f steps. For $[\![\mathsf{map}(\lambda x.\mathsf{d}(x), q)]\!]$ we obtain $\langle 1 + q_{\mathsf{c}} + 2 \cdot q_{\mathsf{l}} + (q_{\mathsf{l}} + 1) \cdot (1 + q_{\mathsf{c}} + q_{\mathsf{m}}), q_l, 2 \cdot q_{\mathsf{m}} \rangle$. This is slightly larger than $[\![\mathsf{dList}(q)]\!]$ (which evaluates to the same term), but has a similar order of magnitude.

For foldl, we can use an interpretation like the one below, where $Q_{g,h,a,m} : \mathbb{N}^2 \to \mathbb{N}^2$ is defined as follows: $Q_{g,h,a,m}(c,s) = \langle c + a + g(c,s,a,m), h(s,m) \rangle$; the superscript denotes repeated function application (e.g., $Q^3(x) = Q(Q(Q(x)))$) and $+$ indicates placewise addition.

$$[\![\mathsf{foldl}(f,z,q)]\!] = \langle 1 + q_{\mathsf{l}} + q_{\mathsf{c}}, 0 \rangle + [\![f]\!](\langle 0, 0 \rangle) + Q^{q_{\mathsf{l}}}_{g,h,q_{\mathsf{c}},q_m max}([\![z]\!])$$

Where $g := \lambda xc, xs, yc, ys.[\![f]\!](\langle xc, xs \rangle, \langle yc, ys \rangle)_1$ and $h := \lambda xs, ys.[\![f]\!](\langle 0, xs \rangle, \langle 0, ys \rangle)_2$ (respectively, the cost and size parts of $[\![f]\!]$. This is much harder to read, but can still be used to gain an idea of the complexity for specific (groups of) instantiations of f.

5 Discussion

This paper aims to start a line of research for termination and complexity analysis of higher-order term rewriting. We abandon the classical notions of derivational and runtime complexity that are often used for this task, since these do not naturally match the behaviour of higher-order terms. We separate cost and size (and other structural properties) in our analysis, which is a similar idea (but very different angle) to analysis using *sized types* [1].

In the future, we plan to further develop the method, and find interpretations to other classic higher-order functions that often occur as part of larger systems. We aim to investigate properties of the technique, and hope to find connections both in the broader area of computational complexity and in the analysis of term rewriting. We are also interested in automating the construction of interpretations, and in applications in functional programming.

References

1. M. Avanzini and U. Dal Lago. Automating sized-type inference for complexity analysis. In *Proc. ICFP*, ACM, page Article 43, 2017.
2. M. Avanzini and G. Moser. Complexity analysis by rewriting. In *Proc. FLOPS*, volume 4989 of *LNCS*, pages 130–146, 2008.
3. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *JAR*, 40:195–220, 2008.
4. N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR 08*, volume 5195 of *LNCS*, pages 364–379, 2008.
5. D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. RTA*, volume 355 of *LNCS*, pages 167–177, 1989.
6. C. Kop, A. Middeldorp, and T. Sternagel. Complexity of conditional term rewriting. *LMCS*, 13(1), 2017.
7. G. Moser, A. Schnabl, and J. Waldmann. Complexity analysis of term rewriting based on matrix and context dependent interpretations. In *Proc. FSTTCS 08*, volume 2 of *LIPIcs*, pages 304–315, 2008.
8. J.C. van de Pol. *Termination of Higher-order Rewrite Systems.* PhD thesis, University of Utrecht, 1996.