Faculty of Mathematics, Computer Science and Natural Sciences Research Group Computer Science 2

RWTH Aachen University, Germany

Bachelor Thesis

A Term Encoding to Analyze Runtime Complexity via Innermost Runtime Complexity

submitted by

Simeon Valchev

Coming soon

Reviewers
Prof. Dr. Jürgen Giesl
apl. Prof. Dr. Thomas Noll

Supervisor
Stefan Dollase, M.Sc.

Statutory Declaration in Lieu of an Oath

Abstract

The automatic tools used to analyze full runtime complexity (rc) of term rewrite systems (TRSs) are still lacking as is shown in the results of the Termination Competition. Meanwhile upper bounds for innermost runtime complexity (irc) are much easily derived. This creates an opportunity to use techniques for irc to infer upper bound on rc. An encoding of the TRS, whose irc coincides with the rc of the original TRS, fills exactly this gap. The technique is implemented in the tool AProVE and tested sufficiently to show it improves the overall automatic complexity analysis.

Contents

Chapter 1

Introduction

Term rewrite systems(TRSs) offer a theoretical foundation to which computer programs can be abstracted to and analyzed. One of the topics of research concerning TRSs has to deal with the worst-case lengths of rewrite sequences, more commonly knows as complexity. In its analysis a distinction is made based on the starting term of the rewrite sequence: *derivational complexity*, considers any term, and *runtime complexity*, considers only basic terms. The more intuitive notion is for a function to be called with some data objects as inputs, which is analogous to rc, and is what we focus on. A further distinction is made according to the evaluation strategy. There is *innermost* runtime complexity (irc), whose eager strategy never evaluates a term before its subterms are in normal form, i.e. can no longer be evaluated, and *full* runtime complexity (rc), which utilizes any strategy.

Most of the work in the field has been focused on irc, which is closer to the evaluation of imperative programs. While the topic of rc may not be as well studied, some notable papers on it are [1, 2, 3] with the most recent being [4], which introduced a novel technique on inferring upper bounds for rc by analyzing irc. However, there is still a wide gap in the automatic complexity analysis on rc as can be observed in the results of the annual *Termination Competition* [5]. Of the 959 TRSs analyzed for rc, an upper bound was discover by at least one of the participants in only 345 cases (36%).

The technique in [4] overapproximates some TRSs as non-dup generalized (ndg) and shows they have the property $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$. This combined with the much more powerful analysis of irc allows inferring upper bounds on rc from upper bounds on irc. All non-ndg TRSs are ignored by this method, leaving a gap, which this thesis aims to fill. This is done by extending the technique from [4] by encoding non-ndg TRSs in a way that $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}'}$, where $\mathcal{R}'$ is the encoding of $\mathcal{R}$.

The two main problems in defining the encoding are calculating the set of non-ndg positions in a TRS and the addition of non-terminating rules. Both of them are tackled in this thesis, but neither is fully solved.

The structure of the thesis is as follows. Chapter 2 introduces some of the preliminary knowledge regarding TRSs. Chapter 3 deals with the problems mentioned above and also introduces the definition of sets that the encoding later uses. In Chapter 4 the encoding is defined and a proof of its correctness is presented. In Chapter 5 are presented the results from the implementation of the technique in the tool AProVE and it also serves as a summary.

Chapter 2

Preliminaries

In this chapter is introduced some of the necessary groundwork on term rewrite systems with accompanying examples.

For now a term can be viewed as some object. Rewriting is then simply a transformation of one term into another. This rewriting is guided by rules, which describe what the input and output of the transformation is. Rules are also associated with some cost, which is considered when analyzing runtime complexity. Unless stated otherwise all rules have a cost of *one*. Rules can also have conditions besides what input they take, however such conditional rules are not in the focus of this thesis. Combining rules together creates a term rewrite system.

Example 1 (Addition)**.**

$$\begin{aligned} \alpha_1 &: \mathsf{add}(\mathsf{x},\mathsf{0}) &&\to \quad \mathsf{x} \\ \alpha_2 &: \mathsf{add}(\mathsf{x},\mathsf{s}(\mathsf{y})) &&\to \quad \mathsf{add}(\mathsf{s}(\mathsf{x}),\mathsf{y}) \end{aligned}$$

Example 1. shows the simple TRS $\mathcal{R}_1$ for the calculation of addition on natural numbers. Numbers here are represented by the so-called successor function and the symbol 0, i.e. 1 would be $\mathsf{s}(\mathsf{0})$, 2 would be $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ and so on. To enhance readability, it is denoted $\mathsf{f}^{\mathsf{n}}(\mathsf{x})$, when some function f is applied n-many times to an input x.

The base case would be represented in α_1, where $x + 0 = x$. In α_2 the TRS recursively subtracts 1 from the second position and adds 1 to the first position. This repeats until the second position reaches 0 and thus the first position is output. One could intuitively imagine this as follows:

$$\begin{array}{ccccccccc} 3+2 & = & 4+1 & = & 5+0 & = & 5 \\ \mathsf{add}(\mathsf{s}^3(\mathsf{0}),\mathsf{s}^2(\mathsf{0})) & \to_{\mathcal{R}_1} & \mathsf{add}(\mathsf{s}^4(\mathsf{0}),\mathsf{s}(\mathsf{0})) & \to_{\mathcal{R}_1} & \mathsf{add}(\mathsf{s}^5(\mathsf{0}),\mathsf{0}) & \to_{\mathcal{R}_1} & \mathsf{s}^5(\mathsf{0}) \end{array}$$

While not explicitly stated in the example, x and y are variables which can be instantiated with other terms. Without a proper definition, one could assume they are some constants like $\mathsf{0}$. Therefore each TRS is associated with a set, which holds all function symbols.

Definition 1 (Signature [6])**.**

A **signature** Σ is a set of function symbols. The number of inputs n of each function symbol is referred to as its arity and it cannot be negative. The subset $\Sigma^{(n)} \subseteq \Sigma$ holds all symbols with arity n. Symbols with arity 0 are called constant.

The signature of $\mathcal{R}_1$ is therefore $\{\mathsf{add},\mathsf{s},\mathsf{0}\}$ with the accompanying $\Sigma^{(2)} = \{\mathsf{add}\}$, $\Sigma^{(1)} = \{\mathsf{s}\}$ and $\Sigma^{(0)} = \{\mathsf{0}\}$, whereas x and y are variables.

Definition 2 (Terms [6])**.**

Let Σ be a signature and $\mathcal{V}$ a set of variables such that $\Sigma \cap \mathcal{V} = \emptyset$. The set $\mathcal{T}(\Sigma, \mathcal{V})$ of all **terms** over Σ and $\mathcal{V}$ is inductively defined as:

- $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$
- for all $n \geq 0$, all $f \in \Sigma^{(n)}$ and all $t_1, ..., t_n \in \mathcal{T}(\Sigma, \mathcal{V})$, we have $f(t_1, ..., t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$

The first item states that each variable on its own is a term and the second item specifies that for arbitrary n many terms $t_1, ..., t_n$ and a function symbol f with arity n, $f(t_1, ..., t_n)$ is also a term. $\mathcal{T}(\Sigma, \mathcal{V})$ is denoted as $\mathcal{T}$, if Σ and $\mathcal{V}$ are irrelevant or clear from the context [4] .

Given the signature $\Sigma = \{\mathsf{add}, \mathsf{s}, \mathsf{0}\}$ of $\mathcal{R}_1$, we can construct terms like $\mathsf{s(0)}$ or $\mathsf{add(x, y)}$ or more complex ones like $\mathsf{add(add(x, s(s(0))), s(0))}$.

Definition 3 (Term positions [6, 4])**.**

1. Let Σ be a signature, $\mathcal{V}$ be a set of variables with $\Sigma \cap \mathcal{V} = \emptyset$ and s some term in $\mathcal{T}(\Sigma, \mathcal{V})$. The set of **positions** of s contains sequences of natural numbers, denoted $pos(s)$ and inductively defined as follows:
   - If $s = x \in \mathcal{V}$, then $pos(s) := \{\varepsilon\}$
   - If $s = f(s_1, ..., s_n)$, then
   $$pos(s) := \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i.\pi \mid \pi \in pos(s_i)\}$$

   The symbol ε is used to point to the **root position** of a term. The function symbol at that position is called the **root symbol** denoted by $root(s)$. Positions can be compared to each other

   $$\pi \leq \tau, \text{ iff there exists } \pi' \text{ such that } \pi.\pi' = \tau$$

   Positions for which neither $\pi \leq \tau$, nor $\pi \geq \tau$ hold, are called **parallel positions** denoted by $\pi \parallel \tau$.
2. The **size** of a term s is defined as $|s| := |pos(s)|$.
3. For $\pi \in pos(s)$, the **subterm of** s **at position** π denoted by $s|_\pi$ is defined as
   $$s|_\varepsilon := s$$
   $$f(s_1, ..., s_n)|_{i.\pi} := s_i|_\pi$$

   If $\pi \neq \varepsilon$, then $s|_\pi$ is a **proper subterm** of s.
4. For $\pi \in pos(s)$, the **replacement in** s **at position** π **with term** t denoted by $s[t]_\pi$ is defined as
   $$s[t]_\varepsilon := t$$
   $$f(s_1, ..., s_n)[t]_{i.\pi} := f(s_1, ..., s_i[t]_\pi, ..., s_n)$$
5. The set of **variables occurring in** s, denoted $\mathcal{V}ar(s)$ is defined as
   $$\mathcal{V}ar(s) := \{x \in \mathcal{V} \mid \exists \pi \in pos(s) \text{ such that } s|_\pi = x\}$$

Consider the term $t = \mathsf{add(x, s(y))}$. The set of position is $pos(t) = \{\varepsilon, 1, 2, 2.1\}$. Some statements that hold for example are $\varepsilon \leq 2 \leq 2.1$ and also $1 \parallel 2$. Some expressions to illustrate are $t|_2 = \mathsf{s(y)}$ and $t[s(0)]_1 = \mathsf{add(s(0), s(y))}$.

Definition 4 (Substitution [6])**.**

Let Σ be a signature and $\mathcal{V}$ a finite set of variables. A **substitution** is a function $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$ such that $\sigma(x) \neq x$ holds for only finitely many xs. It can be written as

$$\sigma = \{x_1 \mapsto \sigma(x_1) \ , \ \cdots \ , \ x_n \mapsto \sigma(x_n)\} \ .$$

The term resulting from $\sigma(s)$ is called an **instance** of s.

Definition 5 (Term rewrite systems [4])**.**

A **rewrite rule** is a pair of terms $\ell \to r$ such that ℓ is not a variable and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$. It is referred to ℓ as the left-hand side(lhs) and r as the right-hand side(rhs). A **term rewrite system** is a finite set of such rewrite rules combined with a signature Σ.

1. Given the signature Σ of a TRS $\mathcal{R}$ we have
   - the **set of defined symbols** $\Sigma_d := \{root(\ell) \mid \ell \to r \in \mathcal{R}\}$
   - the **set of constructor symbols** $\Sigma_c := \Sigma \setminus \Sigma_d$
2. A term $f(s_1, ..., s_n)$ is **basic**, if $f \in \Sigma_d$ and $s_1, ..., s_n \in \mathcal{T}(\Sigma_c, \mathcal{V})$. The **set of all basic terms** over Σ and $\mathcal{V}$ is denoted by $\mathcal{T}_B(\Sigma, \mathcal{V})$.
3. The **number of occurrences of** x **in term** t is denoted by $\#_x(t)$.
4. A **redex** (**red**ucible **ex**pression) is an instance of ℓ of some rule $\ell \to r$.
   - A term t is an **innermost redex**, if none of its proper subterms is a redex.
   - A term t is in **normal form**, if none of its subterms is a redex.
   - A proper subterm t_i of a term t is **nested**, if the t is a redex and $root(t_i) \in \Sigma_d$
5. Rules are also associated with a **cost** and a **condition**. For the purposes of this thesis all rules have no conditions and a cost of either 0 ,denoted $\ell \xrightarrow{0} r$, or 1, denoted $\ell \to r$.

For $\mathcal{R}_1$ in example 1. we have $\Sigma_d = \{\mathsf{add}\}$ and $\Sigma_c = \{\mathsf{0}, \mathsf{s}\}$.
Consider the term $s = \mathsf{add}(\mathsf{add}(\mathsf{0}, \mathsf{0}), \mathsf{0})$. With a substitution $\sigma = \{\mathsf{x} \mapsto \mathsf{add}(\mathsf{0}, \mathsf{0})\}$ it holds that s is an instance of the left-hand side of rule α_1 and therefore s is a redex. But it is not an innermost one as the subterm $s|_1 = \mathsf{add}(\mathsf{0}, \mathsf{0})$ can be reduced to $\mathsf{0}$. Since none of the subterms of $s|_1$ can be reduced, it is an innermost redex.
From now on the i-th rule in order of appearance in a TRS will be denoted as α_i.

Definition 6 (Rewrite step [4])**.**

A **rewrite step** is a reduction(or evaluation) of a term s to t by applying a rule $\ell \to r$ at position π, denoted by $s \to_{\ell \to r, \pi} t$ and means that for some substitution σ, $s|_\pi = \sigma(\ell)$ and $t = s[\sigma(r)]_\pi$ holds.

1. Rule and position in the subscript can be omitted, if they are irrelevant. We denote $s \to_{\mathcal{R}} t$, if it holds for some rule in $\mathcal{R}$, and in order to distinguish between rules and rewrite steps.
2. A sequence of rewrite steps(or rewrite sequence) $s = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_m = t$ is denoted by $s \to^m_{\mathcal{R}} t$.
3. Only rules with cost 1 contribute to the length of a rewrite sequence, i.e. for $s \to^m_{\mathcal{R}} t$ it holds that $m = 0$, if it is a single rewrite step that uses a rule with 0 cost.
4. A rewrite step is called **innermost**, if $s|_\pi$ is an innermost redex, and is denoted by $s \xrightarrow{i}_\pi r$

The term $s = \mathsf{add}(\mathsf{add}(\mathsf{0}, \mathsf{0}), \mathsf{0})$ mentioned above can be used to create the following sequence in $\mathcal{R}_1$

$$\mathsf{add}(\mathsf{add}(\mathsf{0}, \mathsf{0}), \mathsf{0}) \to_{\mathcal{R}_1} \mathsf{add}(\mathsf{0}, \mathsf{0}) \to_{\mathcal{R}_1} \mathsf{0}$$

Definition 7 (Derivation height [4])**.**

The **derivation height** $dh : \mathcal{T} \times 2^{\mathcal{T}\times\mathcal{T}} \to \mathbb{N} \cup \{\omega\}$ is a function with two inputs: a term t and a binary relation on terms, in this case $\to$. It defines the length of the longest sequence of rewrite steps starting with term t, i.e.

$$dh(t, \to) = \sup\{m \mid \exists t' \in \mathcal{T} \ , \ t \to^m t'\}$$

In the case of an infinitely long rewrite sequence starting with term s, it is denoted as $dh(s, \to) = \omega$.

Definition 8 ((Innermost) Runtime complexity [4])**.**

The **runtime complexity**(rc) of a TRS $\mathcal{R}$ maps any $n \in \mathbb{N}$ to the length of the longest $\to$-sequence (rewrite sequence) starting with a basic term t with $|t| \leq n$. The innermost runtime complexity(irc) is defined analogously, but it only considers innermost rewrite steps. More precisely $\mathrm{rc}_\mathcal{R} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ and $\mathrm{irc}_\mathcal{R} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$, defined as

$$\mathrm{rc}_\mathcal{R}(n) = \sup\{dh(t, \to_\mathcal{R}) \mid t \in \mathcal{T}_B, |t| \leq n\}$$
$$\mathrm{irc}_\mathcal{R}(n) = \sup\{dh(t, \xrightarrow{i}_\mathcal{R}) \mid t \in \mathcal{T}_B, |t| \leq n\}$$

Since every $\xrightarrow{i}$-sequence can be viewed as a $\to$-sequence, it is clear that $\mathrm{irc}_\mathcal{R}(n) \leq \mathrm{rc}_\mathcal{R}(n)$. Therefore, an upper bound for $\mathrm{rc}_\mathcal{R}$ infers an upper bound for $\mathrm{irc}_\mathcal{R}$ and a lower bound for $\mathrm{irc}_\mathcal{R}$ infers a lower bound for $\mathrm{rc}_\mathcal{R}$.

For now we can say that the length of rewrite sequences in $\mathcal{R}_1$ depends solely on the second argument of the starting basic term. The size of this subterm is decremented by one after each rewrite until it reaches $\mathsf{0}$, terminating after one more rewrite. $\mathcal{R}_1$ also belongs to a class of system, for which $\mathrm{rc}_\mathcal{R} = \mathrm{irc}_\mathcal{R}$. This class was the focus of [4] and we aim to go beyond it in this thesis.

Definition 9 (Non-dup-generalized (ndg) rewrite step [4, 7])**.**

The rewrite step $s \to_{\ell\to r,\pi} t$ with the matching substitution σ is called **non-dup-generalized**(ndg), if

- For all variables x with $\#_x(r) > 1$, $\sigma(x)$ is in normal form, and
- For all $\tau \in pos(\ell)\backslash\{\varepsilon\}$ with $root(\ell|_\tau) \in \Sigma_d$, it holds that $\sigma(\ell|_\tau)$ is in normal form.

A TRS $\mathcal{R}$ is called ndg, if every rewrite sequence starting with a basic term consists of only ndg rewrite steps. A **non-ndg** rewrite step does not fulfill one of the criteria and a TRS is non-ndg, if it produces a sequence with a non-ndg rewrite step.

The first condition regards duplication, especially the one of defined symbols. It is obvious that the duplication of function calls affects the complexity and is by definition unreachable via an innermost evaluation strategy.

The second condition regards nested defined symbols in the input of a rewrite. This means a function call is ignored, which once again can affect the complexity in unexpected ways.

One can now easily observe that the TRS $\mathcal{R}_1$ is indeed ndg, since none of its rules have duplicated variables on the rhs of rules or nested defined symbols on the lhs of rules. It is impossible for a rewrite sequence starting with a basic term to reach a non-ndg rewrite step. Therefore it holds that $\mathrm{rc}_{\mathcal{R}_1} = \mathrm{irc}_{\mathcal{R}_1}$.

Example 2.

Here we have the simple non-ndg TRS $\mathcal{R}_2$:

$$\begin{array}{lcl} \mathsf{f(x)} & \to & \mathsf{dbl(g(x))} \\ \mathsf{dbl(x)} & \to & \mathsf{d(x,x)} \\ \mathsf{g(s(x))} & \to & \mathsf{g(x)} \end{array}$$

Rule α_2 is duplicating and while that alone is no reason to call this TRS non-ndg, analyzing some sequences it produces proves exactly that. The longest sequence using the basic term $\mathsf{f}(\mathsf{s}(0))$ is

$$\mathsf{f}(\mathsf{s}(0)) \to_{\mathcal{R}_2} \mathsf{dbl}(\mathsf{g}(\mathsf{s}(0))) \to_{\mathcal{R}_2} \mathsf{d}(\mathsf{g}(\mathsf{s}(0)), \mathsf{g}(\mathsf{s}(0))) \to^2_{\mathcal{R}_2} \mathsf{d}(\mathsf{g}(0), \mathsf{g}(0)),$$

which is just one step longer than the innermost rewrite sequence

$$\mathsf{f}(\mathsf{s}(0)) \to_{\mathcal{R}_2} \mathsf{dbl}(\mathsf{g}(\mathsf{s}(0))) \to_{\mathcal{R}_2} \mathsf{dbl}(\mathsf{g}(0)) \to_{\mathcal{R}_2} \mathsf{d}(\mathsf{g}(0), \mathsf{g}(0)).$$

Of course, as the size of the starting term grows the difference between both rewrite sequences increases, but it is important to note that both are linear in terms of asymptotic notation.

While a non-ndg TRS $\mathcal{R}$ fails to satisfy $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$, it is possible that a TRS $\mathcal{R}'$ exists such that $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}'}$. Ideally it would also hold that every full rewrite sequence in $\mathcal{R}$ has an equally long innermost rewrite sequence in $\mathcal{R}'$ and vice versa. The approach taken in this thesis is to transform the original TRS $\mathcal{R}$ by changing some rules and adding new ones, turning it into an encoded version of itself.

Chapter 3

Foundations of the Encoding

By definition every TRS $\mathcal{R}$ has finitely many rules and therefore finitely many positions, which can be changed, i.e. encoded. However, for practical purposes encodings that encode every possible position are not useful. Generally speaking, encodings like this introduce new rules, which can produce loops and create infinitely long rewrite sequences albeit using rules with 0 cost. The focus of this chapter lies in the optimization of which positions to encode (Section 1.) and the limitation of non-terminating rules (Section 2.). To this effort the goal of the encoding is to ensure each innermost rewrite sequence in the encoded TRS $\mathcal{R}'$ has an equally long full rewrite sequence in the TRS $\mathcal{R}$ and vice versa, which is later relevant for the proof of $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}'}$.

Before we continue, it is important to clear one possible misconception. The encoding does not transform a non-ndg TRS $\mathcal{R}$ into an ndg $\mathcal{R}'$. We can show this best via an example.

Example 3.

TRS $\mathcal{Q}$			encoded TRS $\mathcal{Q}'$		
h	$\to$	$\mathsf{g(a,a)}$	h	$\to$	$\mathsf{g(i(}l_{\mathsf{a}}\mathsf{),i(}l_{\mathsf{a}}\mathsf{))}$
a	$\to$	b	a	$\to$	b
$\mathsf{g(x,a)}$	$\to$	$\mathsf{f(x,x)}$	$\mathsf{g(x,}l_{\mathsf{a}}\mathsf{)}$	$\to$	$\mathsf{f(i(x),i(x))}$
			$\mathsf{i(}l_{\mathsf{a}}\mathsf{)}$	$\xrightarrow{0}$	$\mathsf{i(a)}$
			$\mathsf{i(x)}$	$\xrightarrow{0}$	x

Colored in red are the places where the encoded TRS $\mathcal{Q}'$ differs from $\mathcal{Q}$. We can observe the introduction of two new symbols, i.e. the defined symbol i and the constructor l_a. Two new rules are also added and their significance will be discussed later. Currently we want to show that the encoded TRS $\mathcal{Q}'$ is non-ndg.

The duplication of variables can not be avoided in $\mathcal{Q}'$, leaving the possibility of a non-ndg rewrite step. Consider the following sequence

$$\mathsf{h} \to_{\mathcal{R}} \mathsf{g(i(}l_{\mathsf{a}}\mathsf{),i(}l_{\mathsf{a}}\mathsf{))} \xrightarrow{0}_{\mathcal{R}} \mathsf{g(i(a),i(}l_{\mathsf{a}}\mathsf{))} \xrightarrow{0}_{\mathcal{R}} \mathsf{g(i(a),}l_{\mathsf{a}}\mathsf{)} \xrightarrow{0}_{\mathcal{R}} \mathsf{g(a,}l_{\mathsf{a}}\mathsf{)} \to_{\mathcal{R}} \mathsf{f(i(a),i(a))}.$$

The last rewrite step duplicates the term a, which is not in normal form. Therefore neither $\mathcal{Q}$, nor $\mathcal{Q}'$ are ndg.

The new defined symbol i can be more intuitively explained as a helper function symbol, which can choose what part of the term is to be rewritten next. This is what enables the usage of the irc analysis on the encoded TRS to give us results for the original TRS. An analysis on the rules can give us the positions in the TRS, which are to be encoded. The encoding of these positions usually consists of replacing a defined symbol with its new constructor counterpart, e.g. replacing a with l_{a}, and/or enclosing the position in i.

The sequence above can be continued to eventually result in the longest sequence in $\mathcal{Q}'$ and it would also be as long as the longest in $\mathcal{Q}$. It is however not an innermost one. With a slight change of evaluation, the following innermost rewrite sequence is produced:

$$\mathsf{h} \quad \to_{\mathcal{R}} \quad \mathsf{g}(\mathsf{i}(l_{\mathsf{a}}), \mathsf{i}(l_{\mathsf{a}})) \quad \xrightarrow{0}_{\mathcal{R}} \quad \mathsf{g}(l_{\mathsf{a}}, l_{\mathsf{a}}) \quad \to_{\mathcal{R}} \quad \mathsf{f}(\mathsf{i}(l_{\mathsf{a}}), \mathsf{i}(l_{\mathsf{a}})) \quad \xrightarrow{0}_{\mathcal{R}} \quad \mathsf{f}(\mathsf{i}(\mathsf{a}), \mathsf{i}(\mathsf{a})) \quad \to^{2}_{\mathcal{R}} \quad \mathsf{f}(\mathsf{b}, \mathsf{b}).$$

This is one of the sequence that the automatic complexity analysis will discover and use for its result, as the rest will either be shorter or equal in length.

The TRS $\mathcal{Q}$ is still rather simple, but its longest rewrite sequence is not an innermost one. The reason is that the redex $\mathsf{g}(\mathsf{a}, \mathsf{a})$ is not innermost. It would be great, if the a-symbols inside the term could be constructors instead, making it an innermost redex. This is what the encoding sets out to do. But while we have reasons encode the duplication at rule α_3, we also need to reason for the encoding of the a-symbols in rule α_1. They may not directly break any ndg-criteria, but in the context of a rewrite sequence, they can certainly lead to a case where ndg-criteria is broken.

3.1 Non-ndg locations

In the context of rewrite sequences, it is important to ask if certain TRS-positions are reachable in a rewrite sequence and what 'flows' into them. It is clear in example 4. that the rhs of α_1 can be matched against the lhs of α_3 creating a so called **data flow**, i.e. the a-symbol at position 1 of the rhs of α_1 flows into the variable X at position 1 of the lhs of α_3. Since it is very confusing to refer to positions in a TRS like that, we can combine this information in a *triple*.

Definition 10 (Location)**.**

Let $\mathcal{R}$ be a TRS. The set of all **locations**(TRS-positions) of $\mathcal{R}$ is defined as

$$\mathcal{L}_{\mathcal{R}} := \{(\alpha, x, \pi) \mid \alpha = \ell \to r \in \mathcal{R}, x \in \{\mathtt{L}, \mathtt{R}\}, \pi \in pos(\ell) \text{ if } x = \mathtt{L}, \text{ else } \pi \in pos(r)\}.$$

- For all $\ell \to r \in \mathcal{R}$ we write $\mathcal{R}|_{(\ell \to r, \mathtt{L}, \pi)} = \ell|_\pi$ and $\mathcal{R}|_{(\ell \to r, \mathtt{R}, \pi)} = r|_\pi$.
- A location λ is called a **variable location**, if $\mathcal{R}|_\lambda \in \mathcal{V}$.
- The function $loc_{\mathcal{R}} : \mathcal{L}_{\mathcal{R}} \to \mathbb{P}(\mathcal{L}_{\mathcal{R}})$ returns the set of all **sub-locations** of a location $(\alpha, \mathtt{X}, \tau)$ with $\mathtt{X} \in \{\mathtt{L}, \mathtt{R}\}$ and is defined as

  $$loc_{\mathcal{R}}((\alpha, \mathtt{X}, \tau)) := \{(\alpha, \mathtt{X}, \pi) \mid \tau \leq \pi\},$$

  where $\mathbb{P}(\mathcal{L}_{\mathcal{R}})$ is the power set of $\mathcal{L}_{\mathcal{R}}$.

While some work on data flow analysis exists [8], it was deemed not precise enough to use for our encoding. Instead it was preferred to create a separate algorithm, that would return all to-be-encoded locations in a given TRS. After introducing the algorithm, we are going to apply it to the TRS $\mathcal{R}$ from example 4. in order to show how the encoded locations are calculated.

The CAP-function will be used in the algorithm to ease the matching of terms and the detection of data flows. It will however be a cause of less precision in our algorithm as will be shown later.

Definition 11 ($\mathrm{CAP}_{\mathcal{R}}$ [9])**.**

For a given TRS $\mathcal{R}$ the function is defined like $\mathrm{CAP}_{\mathcal{R}} : \mathcal{T} \to \mathcal{T}$. In the resulting term $\mathrm{CAP}_{\mathcal{R}}(q)$ all proper subterms of q with a defined root symbol are replaced with different fresh variables. Multiple occurrences of the same subterm are replaced by pairwise different variables.

$$\mathrm{CAP}_{\mathcal{R}}(q) = \mathrm{CAP}_{\mathcal{R}}(f(t_1, ..., t_n)) = f(t'_1, ..., t'_n) \text{ with}$$

- $t'_k = t_k$, if $t_k \in \mathcal{V}$

- $t'_k = \mathrm{CAP}_{\mathcal{R}}(t_k)$, if $root(t_k) \in \Sigma_c$
- $t'_k = x \in \mathcal{V}$ and $x \notin \mathcal{V}ar(t'_j)$ for $1 \leq j < k$, if $root(t_k) \in \Sigma_d$

Consider the TRS $\mathcal{R}$ from example 1. and the term $q = \mathsf{add}(\mathsf{s}(\mathsf{add}(\mathsf{x},\mathsf{y})),\mathsf{add}(\mathsf{x},\mathsf{y}))$. It holds that

$$\mathrm{CAP}_{\mathcal{R}}(q) = \mathsf{add}(\mathsf{s}(\mathsf{x}_1),\mathsf{x}_2).$$

The first of many sets used to define the encoding is one that contains all so-called **non-ndg** locations, named like this since they potentially cause ndg criteria to be broken.

Definition 12.

For some given TRS $\mathcal{R}$ let the **set of non-ndg locations** $\mathcal{X}_{\mathcal{R}}$ be the smallest set such that:

- $\{(\alpha,\mathtt{L},\tau) \mid \alpha = \ell \to r \in \mathcal{R}, \tau \in pos(\ell)\backslash\{\varepsilon\}, root(\ell|_\tau) \in \Sigma_d\} \subseteq \mathcal{X}_{\mathcal{R}}$ $(S1)$

  This set includes all locations of nested defined symbols on the lhs of rules.

- $\{(\alpha,\mathtt{R},\tau) \mid \alpha = \ell \to r \in \mathcal{R}, \tau,\pi \in pos(r), \tau \neq \pi, r|_\tau = r|_\pi \in \mathcal{V}\} \subseteq \mathcal{X}_{\mathcal{R}}$ $(S2)$

  This set includes all locations of duplicated variables on the rhs of rules.

- $\{(\alpha,\mathtt{L},\tau) \mid \alpha = \ell \to r \in \mathcal{R}, \tau \in pos(\ell), \pi \in pos(r), \ell|_\tau = r|_\pi \in \mathcal{V}\} \subseteq \mathcal{X}_{\mathcal{R}}$, if $(\alpha,\mathtt{R},\pi) \in \mathcal{X}_{\mathcal{R}}$ $(S3)$

  This set includes all locations of variables on the lhs of rules, that flow into locations on the rhs of the same rule, which are also included in $\mathcal{X}_{\mathcal{R}}$.

- $\{(\alpha,\mathtt{R},\tau) \mid \begin{array}{l}\alpha = \ell \to r \in \mathcal{R}, \tau \in pos(r), \tau = \pi.\upsilon, \upsilon \neq \varepsilon \\ \beta = s \to t \in \mathcal{R}, \omega \in pos(s), \upsilon \nparallel \omega \\ \exists \text{ MGU for } \mathrm{CAP}_{\mathcal{R}}(r|_\pi) \text{ and } s \quad\}\end{array} \subseteq \mathcal{X}_{\mathcal{R}}$, if $(\beta,\mathtt{L},\omega) \in \mathcal{X}_{\mathcal{R}}$ $(S4)$

  This set includes all locations on rhs of rules, that flow into locations on lhs of rules, which are also included in $\mathcal{X}_{\mathcal{R}}$.

- $\{(\alpha,\mathtt{R},\tau) \mid \begin{array}{l}\alpha = \ell \to r \in \mathcal{R}, \tau \in pos(r), \\ \beta = s \to t \in \mathcal{R}, \pi \in pos(t), root(t|_\pi) \in \Sigma_d, \\ \exists \text{ MGU for } \mathrm{CAP}(t|_\pi) \text{ and } \ell \quad\}\end{array} \subseteq \mathcal{X}_{\mathcal{R}}$, if $(\beta,\mathtt{R},\pi) \in \mathcal{X}_{\mathcal{R}}$ $(S5)$

  This set includes all locations on rhs of rules, whose lhs root symbol is also at a rhs location included in $\mathcal{X}_{\mathcal{R}}$.

Definition 13.

For some given TRS $\mathcal{R}$ and a set of locations $\Delta = \{(\alpha,\mathtt{R},\tau) \mid \alpha = \ell \to r \in \mathcal{R}, root(r|_\tau) \in \Sigma_d\}$ let $\mathcal{Y}^{\Delta}_{\mathcal{Q}}$ be the smallest set containing Δ and all locations, which locations from Δ flow into:

- $\Delta \subseteq \mathcal{Y}^{\Delta}_{\mathcal{R}}$ $(S6)$

  This set includes all locations on the rhs of rules with a defined symbol.

- $\{(\beta,\mathtt{L},\omega) \mid \begin{array}{l}\alpha = \ell \to r \in \mathcal{R}, \tau \in pos(r), \tau = \pi.\upsilon, \upsilon \neq \varepsilon \\ \beta = s \to t \in \mathcal{R}, \omega \in pos(s), \upsilon \nparallel \omega, \\ \exists \text{ MGU for } \mathrm{CAP}_{\mathcal{R}}(r|_\pi) \text{ and } s\}\end{array} \subseteq \mathcal{Y}^{\Delta}_{\mathcal{R}}$, if $(\alpha,\mathtt{R},\tau) \in \mathcal{Y}^{\Delta}_{\mathcal{R}}$ $(S7)$

  The definition of this subset is similar to $(S4)$. Changed is which location from a data flow is included.

- $\{(\alpha, \mathtt{R}, \tau) \mid \alpha = \ell \to r \in \mathcal{R}, \pi \in pos(\ell), \tau \in pos(r), \ell|_\pi = r|_\tau \in \mathcal{V}\} \subseteq \mathcal{Y}^{\Delta}_{\mathcal{R}}$, if $(\alpha, \mathtt{L}, \pi) \in \mathcal{Y}^{\Delta}_{\mathcal{R}}$ $(S8)$

  The definition of this subset is similar to $(S3)$. Changed is which location from a data flow is included.

Definition 14.

For some given TRS $\mathcal{R}$ let the set of all locations to be encoded $\mathrm{ENC}_{\mathcal{R}}$ be defined as

$$\mathrm{ENC}_{\mathcal{R}} := (\mathcal{X}_{\mathcal{R}} \cap \mathcal{Y}^{\Delta}_{\mathcal{R}}) \backslash \{(\alpha, \mathtt{L}, \pi) \mid \mathcal{R}|_{(\alpha,\mathtt{L},\pi)} \in \mathcal{V}\}.$$

In order to better illustrate why the subsets $(S1 - S8)$ are defined this way, we are going to apply the algorithm to the TRS $\mathcal{Q}$ from example 4. and go into further detail. We call $(S1 - S2)$ and $(S6)$ *base subsets* as they do not depend on other locations being included in their respective larger sets. The rest we call *conditional subsets*.

- $\{(\alpha_3, \mathtt{L}, 1)\} \subseteq \mathcal{X}_{\mathcal{Q}}$ $(S1)$

The first of the two base subsets in $\mathcal{X}_{\mathcal{Q}}$ contains locations of nested defined symbols on lhs of rules. Such locations, if reachable in a sequence, must be replaced by constructors, otherwise the encoded TRS $\mathcal{Q}'$ can not discover an equivalent innermost rewrite sequence, as it will be forced to evaluate the nested defined symbol.

- $\{(\alpha_3, \mathtt{L}, 2), (\alpha_3, \mathtt{R}, 1), (\alpha_3, \mathtt{R}, 2)\} \subseteq \mathcal{X}_{\mathcal{Q}}$ $(S2)$

The second of the two base subsets in $\mathcal{X}_{\mathcal{Q}}$ contains locations of duplicated variables on rhs of rules. It is most commonly the duplication of defined symbols that creates a discrepancy between innermost and full runtime complexity. However, it is not always the case that a redex is duplicated and this is what the second part of the algorithm in $\mathcal{Y}^{\Delta}_{\mathcal{Q}}$ deals with. For now they are included for having the potential to duplicate redexes.

- $\{(\alpha_3, \mathtt{L}, 2), (\alpha_3, \mathtt{R}, 1), (\alpha_3, \mathtt{R}, 2), (\alpha_3, \mathtt{L}, 1)\} \subseteq \mathcal{X}_{\mathcal{Q}}$ $(S3)$

The first conditional subset deals with data flows from left- to right-hand side location. We want to include variable locations on the lhs of rules that flow into non-ndg locations. Later these locations included via $(S3)$ will be removed, as the encoding does not change them, but for the purposes of building $\mathcal{X}_{\mathcal{Q}}$ now, they must be considered.

- $\{(\alpha_3, \mathtt{L}, 2), (\alpha_3, \mathtt{R}, 1), (\alpha_3, \mathtt{R}, 2), (\alpha_3, \mathtt{L}, 1), (\alpha_1, \mathtt{R}, 1), (\alpha_1, \mathtt{R}, 2)\} \subseteq \mathcal{X}_{\mathcal{Q}}$ $(S4)$

The second conditional subset deals with data flows from right- to left-hand side locations

We first split τ in two parts. The symbol at π is the root symbol of the term we try to match against some lhs of any rule. While $\pi = \varepsilon$ is allowed $\upsilon = \varepsilon$ is not, because this would include the ε positions of rhs of rules, which can not flow between locations.

`TO DO: CAP-FUNCTION ELABORATION?`

The MGU is the matcher between the two terms and it means that a term reached via a rewrite step using rule α, can then potentially be used as input for rule β.

Lastly, we require that $\upsilon \nparallel \omega$. This part ensures that the newly included location in α actually flows into a non-ndg location. In our example $\lambda_1 := (\alpha_1, \mathtt{R}, 1)$ flows only into $\lambda_2 := (\alpha_3, \mathtt{L}, 1)$ and we would want to include λ_1, only if λ_2 is non-ndg.

- $\{(\alpha_3, \mathtt{L}, 2), (\alpha_3, \mathtt{R}, 1), (\alpha_3, \mathtt{R}, 2), (\alpha_3, \mathtt{L}, 1), (\alpha_1, \mathtt{R}, 1), (\alpha_1, \mathtt{R}, 2), (\alpha_2, \mathtt{R}, \varepsilon)\} \subseteq \mathcal{X}_{\mathcal{Q}}$ $(S5)$

The third conditional subset includes the so-called *return positions* of non-ndg locations with defined symbols on the rhs of rules. Because they are non-ndg, it means that they flow into a non-ndg location. But in a rewrite sequence that defined symbol may be rewritten before flowing into said non-ndg location. Therefore we must also include these locations.

While this example was simple enough as to allow for the computation of each of the subsets in a single step, most TRS would require that the subsets ($S3 - S5$) be continuously reevaluated, until no further additions are made.

We can now move onto the second part of the algorithm. It may be the case that only constructor terms flow into some variables, which have no affect on runtime complexity, if duplicated. Similarly, instances of lhs of some rule with a nested defined symbol may never be reachable, when starting with a basic term. In both cases encoding such locations does not in any way better the analysis of the encoded version. Therefore, excluding all locations from $\mathcal{X}_\mathcal{R}$ that have no location of a defined symbol flowing into them creates a more precise encoding.

- $\{(\alpha_1\mathtt{R}, \varepsilon), (\alpha_1, \mathtt{R}, 1), (\alpha_1, \mathtt{R}, 2)\} \subseteq \mathcal{Y}^\Delta_\mathcal{Q}$ $(S6)$

The only base subset of $\mathcal{Y}^\Delta_\mathcal{Q}$ includes any location of a defined symbol on the rhs of rules. In this example they all occur in rule α_1.

- $\{(\alpha_1\mathtt{R}, \varepsilon), (\alpha_1, \mathtt{R}, 1), (\alpha_1, \mathtt{R}, 2), (\alpha_3, \mathtt{L}, 1), (\alpha_3, \mathtt{L}, 2)\} \subseteq \mathcal{Y}^\Delta_\mathcal{Q}$ $(S7)$

This and the following conditional subsets are similarly defined to others in $\mathcal{X}_\mathcal{Q}$, but here included are the receivers of the data flow, rather than the source.

As stated previously the $\mathrm{CAP}_\mathcal{Q}$ has its problems and does not entirely remove overapproximation of non-ndg locations. In order to show this, assume $\mathcal{Q}|_{(\alpha_1,\mathtt{R},2)} = \mathsf{h}$. Then obviously the lhs of α_3 is not reachable, when starting with a basic term and should not be encoded. But since $\mathrm{CAP}_\mathcal{Q}(\mathsf{g}(\mathsf{a},\mathsf{h})) = \mathsf{g}(\mathsf{x}_1,\mathsf{x}_2)$ can be matched with the lhs of α_3, we include $(\alpha_3, \mathtt{L}, 2)$ in $\mathcal{Y}^\Delta_\mathcal{R}$ as well. For now this is an acceptable overapproximation.

- $\{(\alpha_1\mathtt{R}, \varepsilon), (\alpha_1, \mathtt{R}, 1), (\alpha_1, \mathtt{R}, 2), (\alpha_3, \mathtt{L}, 1), (\alpha_3, \mathtt{L}, 2), (\alpha_3, \mathtt{R}, 1), (\alpha_3, \mathtt{R}, 2)\} \subseteq \mathcal{Y}^\Delta_\mathcal{Q}$ $(S8)$

As mentioned above this conditional subset is defined similarly to one in $\mathcal{X}_\mathcal{Q}$. The included location and the location in the condition are swapped just like in $(S7)$. In $\mathcal{Q}$ this only happens in α_3, which adds the last element to $\mathcal{Y}^\Delta_\mathcal{Q}$. Analogously to $\mathcal{X}_\mathcal{Q}$, this set is also continuously reevaluated until no further additions are made.

We can now exclude the variable locations on the lhs of rules, since they do not need to be encoded. It is also acceptable to leave them in the set, but to define the encoding in such a way as to ignore them. Here the former approach is taken.

- $\mathrm{ENC}_\mathcal{Q} = (\mathcal{X}_\mathcal{Q} \cap \mathcal{Y}^\Delta_\mathcal{Q}) \backslash \{(\alpha_3\mathtt{L}, 1)\} = \{(\alpha_3, \mathtt{R}, 1), (\alpha_3, \mathtt{R}, 2), (\alpha_3, \mathtt{L}, 2)(\alpha_1, \mathtt{R}, 1), (\alpha_1, \mathtt{R}, 2)\}$

As we can see the non-ndg locations included in $\mathrm{ENC}_\mathcal{Q}$ correspond exactly to the encoded locations in the TRS $\mathcal{Q}'$. However, the encoding also adds new rules in order to define how the i-symbol evaluates. While we have answered the question of what positions we need to encode, we still need to investigate what rules to add to the encoded version of a TRS. This will finally allow us to define the encoding.

3.2 Rules

After defining what locations are to be encoded in a TRS, we can now turn our attention to the additional rules added by the encoding. They deal exclusively with the newly introduced i-symbol and can be split into three categories, summariezed as follows:

- $\mathsf{i}(l_\mathsf{a}) \to \mathsf{i}(\mathsf{a})$

  The *executing rules* usually take the constructor version of a defined symbol (in this case a) as input and evaluates to it. Since this is happening in the context of an innermost evaluation strategy, this means that the next rewrite step will evaluate the a-symbol.

  An example of this is the sequence using TRS $\mathcal{Q}'$ from example 4:

$$\mathsf{h} \quad \to_{\mathcal{Q}'} \quad \mathsf{g}(\mathsf{i}(l_\mathsf{a}), \mathsf{i}(l_\mathsf{a})) \quad \overset{0}{\to}_{\mathcal{Q}'} \quad \mathsf{g}(\mathsf{i}(\mathsf{a}), \mathsf{i}(l_\mathsf{a})) \quad \to_{\mathcal{Q}'} \quad \mathsf{g}(\mathsf{i}(\mathsf{b}), \mathsf{i}(l_\mathsf{a})).$$

- $\mathsf{i}(\mathsf{x}) \to \mathsf{x}$

  The *omission rule* is mandatory in all encodings and it effectively means that the next symbol to be evaluated is at a more outer position in the term or that the term, instantiated at x is in normal form.

  Continuing the sequence from above we get an example of just that:

$$\cdots \quad \to_{\mathcal{Q}'} \quad \mathsf{g}(\mathsf{i}(\mathsf{b}), \mathsf{i}(l_{\mathsf{a}})) \quad \overset{0}{\to}_{\mathcal{Q}'} \quad \mathsf{g}(\mathsf{b}, l_{\mathsf{a}})$$

- $\mathsf{i}(l_{\mathsf{dbl}}(\mathsf{x})) \to \mathsf{i}(l_{\mathsf{dbl}}(\mathsf{i}(\mathsf{x})))$

  The *propagation rules* push the i-symbol further inside the term. For now we can assume that the dbl-symbol was defined. The usage of these rules means that the next symbol to be evaluated is at a more inner position in the term. These rules have two variations: the first is as shown above (*non-terminating*) and the second omits the i-symbol at the ε position on the rhs (*terminating*). We make this distinction due to the consequences of adding both the omission rule and a propagation rule.

  The non-terminating rule above in combination with $\mathsf{i}(\mathsf{x}) \to \mathsf{x}$ for some encoded TRS $\mathcal{R}'$ creates non-terminating rewrite sequences like the following:

$$\cdots \quad \to_{\mathcal{R}'} \quad \mathsf{i}(l_{\mathsf{dbl}}(\mathsf{x})) \quad \overset{0}{\to}_{\mathcal{R}'} \quad \mathsf{i}(l_{\mathsf{dbl}}(\mathsf{i}(\mathsf{x}))) \quad \overset{0}{\to}_{\mathcal{R}'} \quad \mathsf{i}(l_{\mathsf{dbl}}(\mathsf{x})) \quad \overset{0}{\to}_{\mathcal{R}'} \quad \cdots.$$

  Such sequences could throttle the automatic complexity analysis and therefore it is best to avoid adding non-terminating rules whenever possible. An analysis on the rules can determine for which function symbols it would be viable to add the terminating version of the rule.

As seen in previous examples, the propagation rules are not mandatory in an encoding. They are only required for function symbols, which are encoded and have a non-zero arity. Without further analysis, the encoding would have to add the non-terminating propagation rules for these symbols. As discussed, this is not ideal and we should focus on eliminating as many of these non-terminating rules as possible.

The first and easiest condition to start with is checking if any nesting of encoded defined symbols occurs in any of the rules. If that is not the case, then any propagation of the i-symbol would inevitably make it encapsulate a term in normal form. Then all propagation rules would be useless and therefore be omitted. This is however rarely the case.

The very nesting of defined symbols would not be problematic, if it is of constant size i.e. there is a maximum for the number of defined symbols that can be nested in any sequence of a given TRS. In such cases the encoding adds multiple i-symbols at the location of these nestings instead of just one. This allows for the addition of terminating propagation rules, since each of the n-many i-symbols at the encoded location can be used to execute one of the n-many nested defined symbols.

Example 5.

TRS $\mathcal{M}$			encoded		TRS $\mathcal{M}'$
$\mathsf{h}(\mathsf{x})$	$\to$	$\mathsf{f}(0,\mathsf{x})$	$\mathsf{h}(\mathsf{x})$	$\to$	$\mathsf{f}(0,\mathsf{x})$
$\mathsf{f}(\mathsf{x},\mathsf{s}(\mathsf{y}))$	$\to$	$\mathsf{c}(\mathsf{dbl}(\mathsf{g}_1(\mathsf{g}_2(\mathsf{x}))),\mathsf{f}(\mathsf{x},\mathsf{y}))$	$\mathsf{f}(\mathsf{x},\mathsf{s}(\mathsf{y}))$	$\to$	$\mathsf{c}(\mathsf{dbl}(\mathsf{i}^2(l_{\mathsf{g}_1}(l_{\mathsf{g}_2}(\mathsf{x})))),\mathsf{f}(\mathsf{i}(\mathsf{x}),\mathsf{y}))$
$\mathsf{dbl}(\mathsf{x})$	$\to$	$\mathsf{d}(\mathsf{x},\mathsf{x})$	$\mathsf{dbl}(\mathsf{x})$	$\to$	$\mathsf{d}(\mathsf{i}^2(\mathsf{x}),\mathsf{i}^2(\mathsf{x}))$
$\mathsf{g}_1(\mathsf{x})$	$\to$	0	$\mathsf{g}_1(\mathsf{x})$	$\to$	0
$\mathsf{g}_2(\mathsf{x})$	$\to$	0	$\mathsf{g}_2(\mathsf{x})$	$\to$	0
				$\vdots$	
			$\mathsf{i}(l_{\mathsf{g}_1})$	$\to$	$l_{\mathsf{g}_1}(\mathsf{i}(\mathsf{x}))$
			$\mathsf{i}(l_{\mathsf{g}_2})$	$\to$	$l_{\mathsf{g}_2}(\mathsf{i}(\mathsf{x}))$

In this example we can observe the nesting of the defined symbols g_1 and g_2. The encoding measures the size of the nest and adds appropriately many i-symbols in front of it. If g_2 was to be evaluated first, a single i-symbol is to be propagated inwards, leaving one in front of g_1 for its own evaluation. If dbl is

to be evaluated first, both i-symbols get omitted and then replaced via its execution. It is important to follow the flow of the nesting, so as to add appropriately many i-symbols wherever it is needed.

In order to track the flow of these nesting a new set is to be defined, which will later be used in the encoding to add these additional i-symbols. First, we need to measure the size of a nesting. Here we are interested not in the total amount of nested defined symbols, but in the deepest nesting. The reason for this is that the propagation rules add an i-symbol in front of each argument and not just one of them.

Definition 15.

For a given TRS $\mathcal{R}$ the **nesting size** of a location $\lambda \in \mathcal{L}_\mathcal{R}$ is defined as the result of the function $nest_\mathcal{R} : \mathcal{L}_\mathcal{R} \to \mathbb{N}$. Let $max\varnothing = 0$.

$$nest_\mathcal{R}(\lambda) = \begin{cases} max\{nest_\mathcal{R}(\mu) | \mu \in loc_\mathcal{R}(\lambda)\backslash\{\lambda\}\} & \text{if} \quad root(\mathcal{R}|_\lambda) \notin \Sigma_d \\ 1 + max\{nest_\mathcal{R}(\mu) | \mu \in loc_\mathcal{R}(\lambda)\backslash\{\lambda\}\} & \text{if} \quad root(\mathcal{R}|_\lambda) \in \Sigma_d \end{cases}$$

As an example consider the location $\lambda = (\alpha_2, \mathtt{R}, 1)$ in $\mathcal{M}$. The nesting size function counts $\mathsf{dbl}, \mathsf{g_1}$ and $\mathsf{g_2}$, so it outputs $nest_\mathcal{R}(\lambda) = 3$.

However simply observing the direct position of nests is not enough. These nests could flow into some variable that by definition has 0 nesting size. The encoding has to adequately track these and make sure it adds sufficient i-symbols at these locations too.

Definition 16.

1. For some given TRS $\mathcal{R}$ let the set of tuples, where the first element is a location in $\mathcal{R}$ and the second is its nesting size, $\aleph_\mathcal{R}$ be the smallest set such that:
   - $\{(\lambda, n) \mid \lambda \in \mathcal{L}_\mathcal{R}, n = nest_\mathcal{R}(\lambda)\} \subseteq \aleph_\mathcal{R}$
   - $\{(\mu, n) \mid \mu \in \mathcal{Y}_\mathcal{R}^{\{\lambda\}}\} \subseteq \aleph_\mathcal{R}$, if $(\lambda, n) \in \aleph_\mathcal{R}$
2. We now define $\aleph'_\mathcal{R}$ in order to remove nests at the same locations with different sizes. We are only interested in the biggest one.
   - $\{(\lambda, n) \mid \forall (\lambda, m) \in \aleph_\mathcal{R}, n \geq m\}$
3. Lastly we need to update the locations of their new nesting sizes according to the values in $\aleph'_\mathcal{R}$.
   - $nest'_\mathcal{R}(\lambda) = \begin{cases} n, (\lambda, n) \in \aleph'_\mathcal{R} & \text{if} \quad loc_\mathcal{R}(\lambda)\backslash\lambda = \varnothing \\ max\{nest'_\mathcal{R}(\mu) | \mu \in loc_\mathcal{R}(\lambda)\backslash\{\lambda\}\} & \text{if} \quad root(\mathcal{R}|_\lambda) \notin \Sigma_d \\ 1 + max\{nest'_\mathcal{R}(\mu) | \mu \in loc_\mathcal{R}(\lambda)\backslash\{\lambda\}\} & \text{if} \quad root(\mathcal{R}|_\lambda) \in \Sigma_d \end{cases}$

Definition 17.

For some given TRS $\mathcal{R}$ we define the set of actual nests in $\mathcal{R}$, affected by data flows, $\mathrm{NST}_\mathcal{R}$ as

$$\mathrm{NST}_\mathcal{R} := \{(\lambda, n) \mid \lambda \in \mathcal{L}_\mathcal{R}, n = nest'_\mathcal{R}(\lambda)\}$$

The encoding takes the nesting size at each encoded location and adds that many i-symbols in front of them. This is not repeated at subsequent nested encoded locations.

This approach however only solves nestings that do not grow. If the amount of nesting depends on the starting term, then no matter how many i-symbols the encoding puts, there will always be a starting term, which results in more nested defined symbols than there are i-symbols. This means the encoded version must contain the non-terminating propagation rules for these symbols. An automated way to detect such cases is next presented in order to create a more precise encoding.

The conditions for such repeated nestings are quite straightforward. If the location of some defined symbol flows into one of its sub-locations, then it has the potential to nest repeatedly. The symbols at

these locations would be all part of a set, which will later be used in the encoding. Non-terminating propagation rules will only be added for them.

Definition 17.

For a given TRS $\mathcal{R}$ the **set of locations of defined symbols, which repeatedly nest** $\mathrm{INF}_\mathcal{R}$ is defined as

$$\mathrm{INF}_\mathcal{R} := \{\lambda \mid \exists \mu \neq \lambda, \mu \in loc_\mathcal{R}(\lambda), \mu \in \mathcal{Y}_\mathcal{R}^{\{\lambda\}}\}$$

Example 6.

The TRS $\mathcal{P}$ presented in this example contains a case of repeated nesting, which depends on the starting term.

TRS $\mathcal{P}$

$\mathsf{h(x)}$	$\to$	$\mathsf{f(0,x)}$
$\mathsf{f(x,s(y))}$	$\to$	$\mathsf{f(dbl(g(x),y)}$
$\mathsf{dbl(x)}$	$\to$	$\mathsf{d(x,x)}$
$\mathsf{g(x)}$	$\to$	$\mathsf{0}$

The two relevant nests in $\mathcal{P}$ are $((\alpha_2, \mathtt{R}, 1), 2)$ and $(\alpha_2, \mathtt{R}, 1.1), 1)$, which via the same rule create an unpredictable repeated nesting. The encoding will subsequently include the non-terminating propagation rules for the defined symbols at dbl and g.

It is also important that the encoding adds propagation rules for certain constructor symbols. While a TRS can be further analyzed to limit for which this is the case, an encoding that adds propagation rules for all constructor symbols does not in any meaningful way differ from the one that limits it. We can also reason that the propagation rule for any constructor symbol can be terminating without creating any obstacles to the automatic complexity analysis, like defined symbols do. That is because a term with a constructor at the root position can never be a redex and therefore there is no need to keep it encapsulated in an i-symbol.

For a non-ndg TRS $\mathcal{R}$ the sets $\mathrm{ENC}_\mathcal{R}, \mathrm{NST}_\mathcal{R}$ and $\mathrm{INF}_\mathcal{R}$ form the base on which the encoded version is created. The next chapter will define the encoding and prove its correctness.

Chapter 4

Encoding

Definition 18.

Let $\mathbb{T}$ be the set of all TRSs. We define **the encoding** as a function $\Phi : \mathbb{T} \to \mathbb{T}$ defined as:

$$\Phi(\mathcal{R}) = \mathcal{R}',$$

1. $\Phi(\mathcal{R}) \supseteq \{\psi(\alpha) \mid \alpha \in \mathcal{R}\}$
2. $\psi(\alpha) = \varphi(\mathcal{R}|_{(\alpha,\mathtt{L},\varepsilon)}) \to \varphi(\mathcal{R}|_{(\alpha,\mathtt{R},\varepsilon)})$
   Let $f = root(\mathcal{R}|_{(\alpha,\mathtt{X},\pi)})$ be some function symbol with arity m.
   - Location not to be encoded
     $\varphi(\mathcal{R}|_{(\alpha,\mathtt{X},\pi)}) = f(\varphi(\mathcal{R}|_{(\alpha,\mathtt{X},\pi.1)}), \cdots, \varphi(\mathcal{R}|_{(\alpha,\mathtt{X},\pi.m)}))$,
     if $(\alpha, \mathtt{X}, \pi) \notin \mathrm{ENC}_{\mathcal{R}}$
   - Location on lhs to be encoded
     $\varphi(\mathcal{R}|_{(\alpha,\mathtt{L},\pi)}) = l_f(\varphi(\mathcal{R}|_{(\alpha,\mathtt{L},\pi.1)}), \cdots, \varphi(\mathcal{R}|_{(\alpha,\mathtt{L},\pi.m)}))$,
     if $(\alpha, \mathtt{L}, \pi) \in \mathrm{ENC}_{\mathcal{R}}$
   - Location on rhs to be encoded
     $\varphi(\mathcal{R}|_{(\alpha,\mathtt{R},\pi)}) = i^k(l_f(\varphi_{\mathcal{N}}(\mathcal{R}|_{(\alpha,\mathtt{R},\pi.1)}), \cdots, \varphi_{\mathcal{N}}(\mathcal{R}|_{(\alpha,\mathtt{R},\pi.m)})))$,
     if $(\alpha, \mathtt{R}, \pi) \in \mathrm{ENC}_{\mathcal{R}}$ and $((\alpha, \mathtt{R}, \pi), k) \in \mathrm{NST}_{\mathcal{R}}$
   - Location (nested) on rhs to be encoded
     $\varphi_{\mathcal{N}}(\mathcal{R}|_{(\alpha,\mathtt{R},\pi)}) = l_f(\varphi_{\mathcal{N}}(\mathcal{R}|_{(\alpha,\mathtt{R},\pi.1)}), \cdots, \varphi_{\mathcal{N}}(\mathcal{R}|_{(\alpha,\mathtt{R},\pi.m)}))$,
     if $(\alpha, \mathtt{R}, \pi) \in \mathrm{ENC}_{\mathcal{R}}$
3. Propagation rules for constructors
   $\delta_1 = \{\mathsf{i}(f(\mathsf{x}_1, ..., \mathsf{x}_n) \to f(\mathsf{i}(\mathsf{x}_1), ..., \mathsf{i}(\mathsf{x}_n)) \mid f \in \Sigma_c^n\} \cup \{\mathsf{i}(l_f(\mathsf{x}_1, ..., \mathsf{x}_n)) \to \mathsf{i}(f(\mathsf{x}_1, ..., \mathsf{x}_n)) \mid f \in \Sigma_d^n\}$
4. Execution rules
   $\delta_2 = \{\mathsf{i}(l_f(\mathsf{x}_1, ..., \mathsf{x}_n)) \to \mathsf{i}(f(\mathsf{x}_1, ..., \mathsf{x}_n)) \mid f \in \Sigma_d^n\}$
5. Terminating propagation rules
   $\delta_3 = \{\mathsf{i}(l_f(\mathsf{x}_1, ..., \mathsf{x}_n)) \to l_f(\mathsf{i}(\mathsf{x}_1), ..., \mathsf{i}(\mathsf{x}_n)) \mid f = root(\mathcal{R}|_\lambda) \in \Sigma_d, \lambda \in \mathrm{ENC}_{\mathcal{R}} \wedge \forall \mu \in \mathrm{INF}_{\mathcal{R}}, root(\mathcal{R}|_\mu) \neq f\}$

6. Non-terminating propagation rules

   $\delta_4 = \{\mathsf{i}(l_f(\mathsf{x_1}, ..., \mathsf{x_n})) \to \mathsf{i}(l_f(\mathsf{i}(\mathsf{x_1}), ..., \mathsf{i}(\mathsf{x_n}))) \mid f = root(\mathcal{R}|_\lambda) \in \Sigma_d, \lambda \in \mathrm{ENC}_\mathcal{R} \wedge \exists \mu \in \mathrm{INF}_\mathcal{R}, root(\mathcal{R}|_\mu) = f\}$

7. $\mathcal{R}' = \bigcup_{i=1}^{4} \delta_i \cup \{\mathsf{i}(\mathsf{x}) \to \mathsf{x}\} \cup \{\psi(\alpha) \mid \alpha \in \mathcal{R}\}$

Chapter 5

Results

Bibliography

[1] G. Moser, "Proof theory at work: Complexity analysis of term rewrite systems," 2009.

[2] M. Avanzini, G. Moser, and M. Schaper, "TcT: Tyrolean Complexity Tool," in *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, vol. 9636. Springer Verlag Heidelberg, 2016, pp. 407–423.

[3] F. Frohn, J. Giesl, J. Hensel, C. Aschermann, and T. Ströder, "Lower bounds for runtime complexity of term rewriting," *Journal of Automated Reasoning*, vol. 59, pp. 1–43, 06 2017.

[4] F. Frohn and J. Giesl, "Analyzing runtime complexity via innermost runtime complexity," in *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning.* EasyChair, 2017.

[5] Termination competition. [Online]. Available: https://termination-portal.org/wiki

[6] F. Baader and T. Nipkow, *Term Rewriting and All That.* Cambridge University Press, 1998. [Online]. Available: https://www.cambridge.org/core/books/term-rewriting-and-all-that/71768055278D0DEF4FFC74722DE0D707

[7] J. Pol, van de and H. Zantema, "Generalized innermost rewriting," in *Rewriting Techniques and Applications (Proceedings 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005)*, ser. Lecture Notes in Computer Science, J. Giesl, Ed. Germany: Springer, 2005, pp. 2–16.

[8] S. Lechner, "Data flow analysis for integer term rewrite systems," *RWTH*, 2022.

[9] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke, "Mechanizing and improving dependency pairs," *J. Autom. Reasoning*, vol. 37, pp. 155–203, 10 2006.