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

RWTH Aachen University, Germany

Bachelor Thesis

Analyzing runtime complexity of non-ndg Term Rewrite Systems

submitted by

Simeon Valchev

Coming soon

Reviewers
Prof. Dr. Jürgen Giesl
TBD

Supervisor
Stefan Dollase, ...?

Statutory Declaration in Lieu of an Oath

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

1 Introduction **1**

2 Preliminaries **3**

3 Algorithm for finding non-ndg locations **9**

4 Encoding **13**

5 Encoding with terminating rules **15**

6 Results **17**

Chapter 1

Introduction

Term rewrite systems have been extensively studied, with the main problem concerning their complexity analysis. In recent decades many automated tools have been developed to infer upper bounds on runtime complexity (rc), which considers the longest rewrite sequence starting with a basic term disregarding any evaluation strategy. In contrast to rc, innermost runtime complexity (irc) makes use of an innermost evaluation strategy that can be much easier to analyse. While the analysis of irc has some uses, it is rc that is of greater interest.

Recently a technique has been developed to overapproximate some TRSs as non-dup generalized (ndg), with a corresponding proof of the equality of rc and irc for such ndg TRSs. The mentioned criteria combined with the much more powerful analysis of irc allowed inferring upper/lower bounds on rc from upper/lower bounds on irc. This method however makes no claims regarding all the TRSs that are non-ndg. It is the goal of this thesis to extend the technique to non-ndg TRSs.

Presented here is a transformation of the original non-ndg TRS into an encoded version, that shares the same rc. The goal of the encoding is to create a TRS, for which also holds that irc = rc. While the encoded version may not ndg, what is important is that we can apply the techniques used in the analysis of irc to infer the rc of the original non-ndg TRS.

As it will be shown, not all encodings are equal and some yield substantially better results. Two algorithms have been developed to improve the results of the automated analysis. The first - overapproximating the TRS positions that require encoding and a second - detecting when to alter the encoding to not add any non-terminating rules.

The structure of the thesis is as follows. Chapter 2 introduces some of the preliminary knowledge regarding TRSs. Chapter 3 contains a description of the first mentioned algorithm, which is used as the foundation, on which the encoding, described in Chapter 4, is built. Chapter 5 is dedicated to the second algorithm and the changes that come up from its result. In Chapter 6 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.

A term rewrite system (TRS) $\mathcal{R}$ is a finite set of rewrite rules $\ell \to r$, where ℓ and r are terms in the set $\mathcal{T}(\Sigma, \mathcal{V})$ over the signature Σ and the set of variables $\mathcal{V}$. In the context of a rule $\ell \to r$, ℓ refers to the left-hand side and r to the right-hand side. The rules describe how terms are rewritten, i.e. the left-hand side of a rule can be matched against some part of a term s, rewrite it to the matching right-hand side and give output t.

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 a simple TRS for the calculation of addition on natural numbers. Numbers here are represented by the so-called successor function and the base symbol 0, i.e. 1 would be $\mathsf{s}(\mathsf{0})$, 2 - $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ and so on. In this way our base case would be represented in α_1, where $x + 0 = x$. In α_2 the TRS recursively subtracts 1 from the second argument and adds 1 to the first argument. This repeats until the second argument reaches 0 and thus the first argument is output. One could intuitively imagine this as follows:

$$3 + 2 \quad = \quad 4 + 1 \quad = \quad 5 + 0 \quad = \quad 5$$

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

A **signature** Σ is a set of function symbols, where each $f \in \Sigma$ is associated with an arity $n \geq 0$. For some valid n, we denote the set of all n-ary elements of Σ as $\Sigma^{(n)}$. The elements of $\Sigma^{(0)}$ are also called constant symbols.

The signature of example 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 [1]).

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, its application $f(t_1, ..., t_n)$ is also a term.

If Σ and $\mathcal{V}$ are irrelevant or clear from the context, only $\mathcal{T}$ is written instead of $\mathcal{T}(\Sigma, \mathcal{V})$ [2] .

Given the signature $\Sigma = \{\mathsf{add}, \mathsf{s}, \mathsf{0}\}$ of example 1, we can construct terms like $\mathsf{s}(\mathsf{0})$ or $\mathsf{add}(\mathsf{x}, \mathsf{y})$ or more complex ones like $\mathsf{add}(\mathsf{add}(\mathsf{x}, \mathsf{s}(\mathsf{s}(\mathsf{0}))), \mathsf{s}(\mathsf{0}))$.

Definition 3 (Term positions [1, 2]).

1. Let Σ be a signature, $\mathcal{V}$ be a set of variables with $\Sigma \cap \mathcal{V} = \emptyset$ and $s \in \mathcal{T}(\Sigma, \mathcal{V})$. The set of **positions** of the term s is a set $pos(s)$ of strings, inductively defined as follows:
   - If $s = x \in \mathcal{V}$, then $pos(s) := \{\epsilon\}$, where ϵ denotes the empty string
   - If $s = f(s_1, ..., s_n)$, then
   $$pos(s) := \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i.\pi \mid \pi \in pos(s_i)\}$$

   The position ϵ of a term s refers to the **root position** of s and the symbol at that position is called the **root symbol** denoted by $root(s)$. The **prefix order** on term positions is defined as
   $$\pi \leq \tau \text{ iff there exists } \pi' \text{ such that } \pi.\pi' = \tau$$
   It is a partial order. 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 inductively as
   $$\begin{aligned} s|_\epsilon &:= s \\ f(s_1, ..., s_n)|_{i.\pi} &:= s_i|_\pi \end{aligned}$$
   If $\pi \neq \epsilon$, 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
   $$\begin{aligned} s[t]_\epsilon &:= t \\ f(s_1, ..., s_n)[t]_{i.\pi} &:= f(s_1, ..., s_i[t]_\pi, ..., s_n) \end{aligned}$$
5. By $\mathcal{V}ar(s)$ we denote the set of **variables occurring in** s, i.e.
   $$\mathcal{V}ar(s) := \{x \in \mathcal{V} \mid \exists \pi \in pos(s) \text{ such that } s|_\pi = x\}$$

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

Definition 4 (Substitution [1]).

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 $\sigma(x) \neq x$ for only finitely many xs. The **domain** $Dom(\sigma) := \{x \mid \sigma(x) \neq x\}$ is the set of all variables that σ does not map to themselves. The **range** $Ran(\sigma) := \{\sigma(x) \mid x \in Dom(\sigma)\}$ is the set of all terms that σ maps to. If $Dom(\sigma) = \{x_1, ..., x_n\}$, then we may write σ as

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

A term t is called an **instance** of a term s, iff there exists a substitution σ such that $\sigma(s) = t$.

Definition 5 (Term rewrite systems [2]).

A **rewrite rule** is an identity $\ell \approx r$ such that ℓ is not a variable and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$. In this case it is denoted as $\ell \to r$ instead of $\ell \approx r$. It is referred to ℓ as the left-hand side(lhs) and r as the right-hand side(rhs). A **term rewrite system** (TRS) is a finite set of such rewrite rules.

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. Given term t and $x \in \mathcal{V}$ the **number of occurrences of** x **in** 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 is an **innermost redex**, if none of its proper subterms is a redex.
   - A term is in **normal form**, if none of its subterms is a redex.

In example 1. we have $\Sigma_d = \{\mathsf{add}\}$ and $\Sigma_c = \{\mathsf{0}, \mathsf{s}\}$.
Consider the basic 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.

Definition 6 (Rewrite step [2]).

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. A reduction can be expressed as $s \to_{\mathcal{R},\pi} t$, if it holds for some $\ell \to r \in \mathcal{R}$. Subscripts such as the rule or position of reduction can be omitted, if they are irrelevant.

1. A sequence of rewrite steps(or rewrite sequence) $s \to t_1 \to t_2 \to \cdots \to t_m = t$ is denoted by $s \to^m t$.
2. A rewrite step is called **innermost**, if $s|_\pi$ is an innermost redex, and is denoted by $s \xrightarrow{i}_\pi r$

The basic term $s = \mathsf{add}(\mathsf{add}(\mathsf{0},\mathsf{0}),\mathsf{0})$ from example 1. can be used to create the following sequence:

$$\mathsf{add}(\mathsf{add}(\mathsf{0},\mathsf{0}),\mathsf{0}) \to \mathsf{add}(\mathsf{0},\mathsf{0}) \to \mathsf{0}$$

Definition 7 (Derivation height [2]).

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 longest sequence of rewrite steps starting with term t, i.e.

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

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 [2]).

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 $\mathrm{rc}_{\mathcal{R}}$. A specific class of TRSs has been previously studied in [2], for which $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$ holds. The results from the analysis on runtime complexity of such TRSs can now be used in the analysis on innermost runtime complexity and vice versa, as mentioned above.
Example 1. is sufficiently simple to see that the analysis on rc coincides with the one on irc, because every rewrite sequence starting with a basic term is also an innermost one. This TRS is a part of a class of systems, for which $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$. For now we can say that the length of rewrite sequences in example 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. Thus it holds that $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}} \in \mathcal{O}(n)$.

Definition 9 (Non-dup-generalized (ndg) rewrite step [2, 3]).

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 $\tau \in pos(\ell)\backslash\{\epsilon\}$ 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. It is proven in [2] that $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$ for all ndg TRSs $\mathcal{R}$. One can now easily observe that example 1. does fit the criteria, i.e. it has no duplicated variables on the right-hand side of rules and no nested defined symbols on left-hand sides.

Example 2.

This and the following example will present some TRSs, that do not fit one of the criteria stated above, and show why $\mathrm{rc}_{\mathcal{R}} \neq \mathrm{irc}_{\mathcal{R}}$ for them.

$$\begin{array}{rccc}
\alpha_1 : & \mathsf{g} & \to & \mathsf{f(a)} \\
\alpha_2 : & \mathsf{a} & \to & \mathsf{b} \\
\alpha_3 : & \mathsf{f(a)} & \to & \mathsf{f(a)}
\end{array}$$

The $\text{rc}_{\mathcal{R}}$ of this TRS is reached via the infinite sequence $\mathsf{g} \to \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}) \to \cdots$, while the $\text{irc}_{\mathcal{R}}$ corresponds the two step long sequence $\mathsf{g} \to \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b})$. It is the nested a in rule α_3 that makes this a non-ndg TRS with as shown non-equal rc and irc.

Example 3.

$$\begin{array}{rccc} \alpha_1: & \mathsf{f}(\mathsf{x}) & \to & \mathsf{dbl}(\mathsf{g}(\mathsf{x})) \\ \alpha_2: & \mathsf{dbl}(\mathsf{x}) & \to & \mathsf{d}(\mathsf{x},\mathsf{x}) \\ \alpha_3: & \mathsf{g}(\mathsf{s}(\mathsf{x})) & \to & \mathsf{g}(\mathsf{x}) \end{array}$$

Here rule α_2 is duplicating. Using the basic term $\mathsf{f}(\mathsf{s}(\mathsf{0}))$ we get the longest sequence $\mathsf{f}(\mathsf{s}(\mathsf{0})) \to \mathsf{dbl}(\mathsf{g}(\mathsf{s}(\mathsf{0})))$ $\to \mathsf{d}(\mathsf{g}(\mathsf{s}(\mathsf{0})), \mathsf{g}(\mathsf{s}(\mathsf{0}))) \to^2 \mathsf{d}(\mathsf{g}(\mathsf{0}), \mathsf{g}(\mathsf{0}))$, which is just one step longer than the innermost rewrite sequence $\mathsf{f}(\mathsf{s}(\mathsf{0})) \to \mathsf{dbl}(\mathsf{g}(\mathsf{s}(\mathsf{0}))) \to \mathsf{dbl}(\mathsf{g}(\mathsf{0})) \to \mathsf{d}(\mathsf{g}(\mathsf{0}), \mathsf{g}(\mathsf{0}))$.

While the technique used in [2] is helpful in inferring upper and lower bounds for rc and irc on ndg systems, it makes no statements on non-ndg systems. The examples above show that the criteria for ndg systems clearly exclude a lot of TRSs. It is the goal of this thesis to expand the mentioned technique to non-ndg systems by encoding certain positions in the TRS. For the new encoded TRS it will hold that $\text{rc}_{\mathcal{R}} = \text{irc}_{\mathcal{R}}$.

Chapter 3

Algorithm for finding non-ndg locations

Before we define how we encode a TRS it is important to establish what the goal of said encoding is. As stated previously the automated complexity analysis has access to much better techniques when dealing with innermost rewriting rather than full rewriting. So assuming we can transform a TRS $\mathcal{R}$ into a different $\mathcal{R}'$, while also sharing the same runtime complexity, would it also be possible to encode it in such a way that $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}'}$ holds? Not only is it possible, but it can be automated and used to analyze the rc of the original system.

Example 4 (Encoding example 2)**.**

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

Colored in red are the places where the encoding changed the TRS $\mathcal{R}$. We can observe the emergence of two new symbols, i.e. the defined symbol i and the constructor l_a. There also two new rules, which for the purpose of rc analysis have 0 cost or in other words: rewrite steps that are done using the new rules do not count towards the derivation height. One can be inclined to think that the encoded TRS $\mathcal{R}'$ is now an ndg TRS since the problematic nested a-symbol in the third rule is removed. However the last rewrite of the sequence

$$\mathsf{g} \to \mathsf{f}(\mathsf{i}(l_{\mathsf{a}})) \xrightarrow{0} \mathsf{f(i(a))} \xrightarrow{0} \mathsf{f(a)}$$

is notably not an ndg rewrite as a is not in normal form. It is not the goal of the encoding to turn a non-ndg system into an equivalent ndg system. In spite of that every rewrite sequence in $\mathcal{R}$ has an equivalent innermost rewrite sequence in $\mathcal{R}'$, which leads to the desired property $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}'}$.

As already shown, the longest rewrite sequence in $\mathcal{R}$ is the infinite $\mathsf{g} \to \mathsf{f(a)} \to \mathsf{f(a)} \to \cdots$, that can be represented in $\mathcal{R}'$ as

$$\mathsf{g} \to \mathsf{f}(\mathsf{i}(l_{\mathsf{a}})) \xrightarrow{0} \mathsf{f}(l_{\mathsf{a}}) \to \mathsf{f}(\mathsf{i}(l_{\mathsf{a}})) \xrightarrow{0} \mathsf{f}(l_{\mathsf{a}}) \to \cdots$$

The new defined symbol i is in a way supposed to choose which part of the term is rewritten next, and it ensures that the rewrite steps remain innermost. In order to do this, certain positions in the TRS are to be encoded, besides the addition of the new rules. The encoding of these positions usually consists of replacing a defined symbol with its new constructor counterpart, e.g. replacing a with l_{a}, or enclosing the position in i. While we do have a reason to encode the a-symbol on the lhs of $\mathsf{f(a)} \to \mathsf{f(a)}$ in $\mathcal{R}$, because it breaks ndg criteria, how do we justify the rest of the encoding?

Because we consider rewrite sequences and not just rules, it is important to also ask if such positions are reachable in a sequence and what 'flows' into them. In example 4, it is clear that the rhs of the first rule will be matched against the lhs of the third rule creating a so called **data flow**. Analogously this also holds for the rhs of the third rule. We can use the data flow analysis from [4] together with some base criteria to establish the set of positions in the TRS that need to be encoded. However it was preferred to create a separate algorithm that discovers these positions on its own, providing the benefit of a self-contained thesis.

Since positions are already defined, it is ill-suited to refer to positions in a TRS as it can create confusion. After defining them separately, we introduce a helper function, before we move onto the algorithm.

Definition 10 (Location)**.**

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

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

We denote the term at a location as $\mathcal{R}|_{(\ell \to r, \mathtt{L}, \pi)} = \ell|_\pi$ and $\mathcal{R}|_{(\ell \to r, \mathtt{R}, \pi)} = r|_\pi$.

Definition 11 (CAP$_{\mathcal{R}}$)**.**

The function $\text{CAP}_{\mathcal{R}} : \mathcal{T} \to \mathcal{T}$ replaces all proper subterms of a term t with different fresh variables. Multiple occurrences of the same subterm are replaced by pairwise different variables.

For example let $t = \mathsf{add(add(add(x, y), z), add(x, y))}$ be a term from example 1.

$$\text{CAP}_{\mathcal{R}}(t) = \mathsf{add}(\mathsf{x}_1, \mathsf{x}_2)$$

Example 5.

Due to the many steps of the algorithm, we are going to start with an example of a non-ndg system and observe the progress step by step.

TRS $\mathcal{R}_5$

α_1 :	$\mathsf{h(x)}$	$\to$	$\mathsf{f(g(x), x)}$
α_2 :	$\mathsf{f(x, s(y))}$	$\to$	$\mathsf{c(x, f(x, y)))}$
α_3 :	$\mathsf{g(s(x))}$	$\to$	$\mathsf{g(x)}$

To prove it is non-ndg one can simply look at the sequence

$$\mathsf{h(s(x))} \to \mathsf{f(g(s(x)), s(x))} \to \mathsf{c(g(s(x)), f(g(s(x)), x))} \to \cdots$$

where the second rewrite uses rule α_2 with a subterm not in normal form, namely $\mathsf{g(s(x))}$.

Algorithm 1.

Let $\mathcal{R}$ be a TRS with signature $\Sigma = \Sigma_d \cup \Sigma_c$ and a set a variables $\mathcal{V}$.

1. Let $\mathcal{X}_\mathcal{R}$ be the smallest set such that

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

     This set includes all locations of nested defined symbols on the lhs of rules. There are none in $\mathcal{R}_5$, but this case was already presented in example 4.

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

     This set includes all locations of duplicated variables on the rhs of rules. There are two such cases in $\mathcal{R}_5$ in the first two rules. Therefore it holds that

     $$\mathcal{X}_1 = \{(\alpha_1, \mathtt{R}, 1.1), (\alpha_1, \mathtt{R}, 2), (\alpha_2, \mathtt{R}, 1), (\alpha_2, \mathtt{R}, 2.1)\} \subseteq \mathcal{X}_{\mathcal{R}_5}$$

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

     This set includes all locations on lhs of rules that flow directly into non-ndg locations. By direct flow it is meant that the data flow occurs in the same rule. This is most notable when a variable is duplicated. Continuing with $\mathcal{R}_5$ we have

     $$\mathcal{X}_2 = \mathcal{X}_1 \cup \{(\alpha_1, \mathtt{L}, 1), (\alpha_2, \mathtt{L}, 1)\} \subseteq \mathcal{X}_{\mathcal{R}_5}$$

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

     This set includes all locations on rhs of rules that flow into non-ndg locations on lhs of rules. This can only occur if both the lhs and the redex at position π can be matched (i.e. have a Most General Unifier). More specifically, we take the result of the CAP-function on the redex at position π. This sacrifices some precision in determining the set of non-ndg locations, but offers much easier and intuitive the detection of potential data flows. For the purposes of this thesis, such an overapproximation is acceptable.

     The reason why the definition requires that the positions υ and ω not be parallel(i.e. $\upsilon \leq \omega$ or $\upsilon \geq \omega$), is better illustrated through example 5. Assuming the location $(\alpha_1, \mathtt{R}, 2)$ was not previously included in the final set, it can be observed that it fulfills all the criteria for this subset except the last $\upsilon \nparallel \omega$. If this criteria was not there, then the result would be that it gets marked as non-ndg, because a parallel location has a data flow to a non-ndg location. This is, of course, not acceptable and is appropriately addressed.

     In $\mathcal{R}_5$ the only location, fitting this criteria that is also not already in $\mathcal{X}_2$, is $(\alpha_1, \mathtt{R}, 1)$. This can be verified as follows. $\mathrm{CAP}_\mathcal{R}(\mathsf{f}(\mathsf{g}(\mathsf{x}), \mathsf{x})) = \mathsf{f}(\mathsf{x}_1, \mathsf{x})$, which is unified with the variable renamed $\mathsf{f}(\mathsf{y}_1, \mathsf{s}(\mathsf{y}))$ with $\sigma = \{\mathsf{x}_1 \mapsto \mathsf{y}_1, \mathsf{x} \mapsto \mathsf{s}(\mathsf{y})\}$

   - $\{(\alpha, \mathtt{R}, \tau) \mid \alpha = \ell \to r \in \mathcal{R},$
     $\beta = s \to t \in \mathcal{R}, \mathit{root}(\ell) = \mathit{root}(t|_\pi) \in \Sigma_d, \tau \in \mathit{pos}(r)\} \subseteq \mathcal{X}_\mathcal{R}$, if $(\beta, \mathtt{R}, \pi) \in \mathcal{X}_\mathcal{R}$

     This set includes all locations on rhs of rules that can be called *return positions* of other non-ndg locations containing a defined symbol. There is one such occurrence in $\mathcal{R}_5$ and it is in the third rule:

     $$\mathcal{X}_3 = \{(\alpha_3, \mathtt{R}, \epsilon), (\alpha_3, \mathtt{R}, 1)\} \cup \mathcal{X}_2 \subseteq \mathcal{X}_{\mathcal{R}_5}$$

     Consider the sequence

     $$\mathsf{h}(\mathsf{s}(\mathsf{x})) \to \mathsf{f}(\mathsf{g}(\mathsf{s}(\mathsf{x})), \mathsf{s}(\mathsf{x})) \to \mathsf{f}(\mathsf{g}(\mathsf{x}), \mathsf{s}(\mathsf{x})) \to \mathsf{c}(\mathsf{g}(\mathsf{x}), \mathsf{f}(\mathsf{g}(\mathsf{x}), \mathsf{x})) \to \cdots$$

     The second rewrite uses α_3 and then α_2. If the locations on the rhs of α_3 were not included and therefore not encoded, this could lead to problems in sequences like the one above and result in wrong analysis on the complexity of the system.

So far we have a set of locations, which if encoded would give the desired result of a TRS that shares its irc with the rc of the original TRS. However, there are ways to make this set even more precise. As stated before, the encoding creates a way to move inside the term and choose what to rewrite next with an innermost evaluation strategy. Terms starting with a constructor symbol cannot be a redex and if none of its subterms are redexes, then there is no need to move inside it via the encoding. This means, that while some variable-locations may be included in the set of non-ndg locations, they may never 'receive' a term containing a redex and so need no encoding.

In example 5, the TRS $\mathcal{R}_5$ has multiple such locations. Most obvious one is $(\alpha_1, \mathtt{L}, 1)$, which can only be reached if it is the starting term. Since we also restring the starting terms to basic term(no nested defined symbols), we can safely conclude that the variable location $(\alpha_1, \mathtt{L}, 1)$, will never 'receive' a term with a redex. This can extended to the rhs and also conclude that $(\alpha_1, \mathtt{R}, 1.2)$ and $(\alpha_1, \mathtt{R}, 2)$ also never 'receive' a term with a redex. These locations can therefore be excluded from the set and not encoded. This is what the second part of the algorithm deals with.

2. Let $\mathcal{Y}_\mathcal{R}$ be the smallest set such that

   - $\{(\alpha, \mathtt{R}, \tau) \mid \alpha = \ell \to r \in \mathcal{R}, root(r|_\tau) \in \Sigma_d\} \subseteq \mathcal{Y}_\mathcal{R}$

     This set includes all locations on the rhs of rules where a defined symbol can be found.

   -
   -

Chapter 4

Encoding

Chapter 5

Encoding with terminating rules

Chapter 6

Results

Bibliography

[1] 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

[2] 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*, ser. EPiC Series in Computing, T. Eiter and D. Sands, Eds., vol. 46. EasyChair, 2017, pp. 249–268. [Online]. Available: https://easychair.org/publications/paper/TPg

[3] 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.

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