The following full text is a preprint version which may differ from the publisher's version.

For additional information about this publication click this link. http://hdl.handle.net/2066/92375

Please be advised that this information was generated on 2021-07-15 and may be subject to change.
Strategy Independent Reduction Lengths in Rewriting and Binary Arithmetic

Hans Zantema
University of Technology, Eindhoven, The Netherlands
Radboud University, Nijmegen, The Netherlands
h.zantema@tue.nl

Abstract

In this paper we give a criterion by which one can conclude that every reduction of a basic term to normal form has the same length. As a consequence, the number of steps to reach the normal form is independent of the chosen strategy. In particular this holds for TRSs computing addition and multiplication of natural numbers, both in unary and binary notation.

1 Introduction

For many term rewriting systems (TRSs) the number of rewrite steps to reach a normal form strongly depends on the chosen strategy. For instance, in using the standard if-then-else-rules

\[
\text{if}(\text{true}, x, y) \rightarrow x \quad \text{if}(\text{false}, x, y) \rightarrow y
\]

it is a good strategy to first rewrite the leftmost (boolean) argument of the symbol if until this argument is rewritten to false or true and then apply the corresponding rewrite rule for if. In this way redundant reductions in the third argument are avoided in case the condition rewrites to true, and redundant reductions in the second argument are avoided in case the condition rewrites to false. More general, choosing a good reduction strategy is essential for doing efficient computation by rewriting. Roughly speaking, for erasing rules, that is, some variable in the left-hand side does not appear in the right-hand side, it seems a good strategy to postpone rewriting this possibly erasing argument, as is the case in the above if-then-else example. Conversely, in case of duplicating rules, that is, a variable occurs more often in the right-hand side than in the left-hand side, it seems a good strategy to first rewrite the corresponding argument before duplicating it.

Surprisingly however, there are practical examples of TRSs including both such erasing and duplicating rules, where the strategy has no influence at all on the number steps required to reach a normal form. In this paper we investigate criteria for this phenomenon. As an example, using the results of this paper, we will show that for multiplying two natural numbers by the standard TRS

\[
\begin{align*}
\text{plus}(0, x) & \rightarrow x \\
\text{mult}(0, x) & \rightarrow 0 \\
\text{plus}(s(x), y) & \rightarrow s(\text{plus}(x, y)) \\
\text{mult}(s(x), y) & \rightarrow \text{plus}(\text{mult}(x, y), y)
\end{align*}
\]

the number of steps to reach the resulting normal form is independent of the chosen strategy. Note that the third rule of this TRS is erasing, and the last rule is duplicating.

In this TRS ground terms reduce to ground normal forms of the shape \(s^n(0)\) for any natural number \(n\), representing the corresponding number \(n\) in unary notation. For large numbers \(n\) this is a quite inefficient representation. Much more efficient is the binary representation. In this paper we also give a TRS for doing addition and multiplication in binary representation. Here for every addition or multiplication of two positive integer numbers we show by the main theorem
of this paper that every reduction to normal form has the same number of steps. Moreover, this number has the same complexity as the standard binary algorithms: linear for addition and quadratic for multiplication.

Through the paper we assume familiarity with standard notions in rewriting like orthogonality and normal forms as they are introduced in e.g. [3]. For instance, a reduction is a sequence of rewrite steps, and a reduct of a term \( t \) is a term \( u \) for which there is a reduction from \( t \) to \( u \).

In Section 2 we investigate the diamond property and present our main result Theorem 3: a criterion by which for every basic term all reductions to normal form have the same length. Here a term is called basic if only its root is a defined symbol. We apply this to unary arithmetic. In Section 3 it is shown how our theorem applies to binary arithmetic. We conclude in Section 4.

2 The diamond property and the main result

We say that a binary relation \( \rightarrow \) satisfies the diamond property, if for every three elements \( s, t, u \) satisfying \( s \rightarrow t \) and \( s \rightarrow u \) and \( t \neq u \), there exists an element \( w \) such that \( t \rightarrow w \) and \( u \rightarrow w \).

This notion is slightly different from other variants of the diamond property, for instance introduced in [3] where it is typically used for reflexive relations. However, as we are interested in the exact number of steps to reach a normal form, our present version is the most natural. It is an instance of the balanced weak Church-Rosser property from [4]. The following lemma is an immediate consequence of Lemma 1 from [4].

**Lemma 1.** Let \( \rightarrow \) be a relation satisfying the diamond property. Then every element has at most one normal form, and for every element having a normal form it holds that every reduction of this element to its normal form has the same number of steps.

An orthogonal TRS is said to be variable preserving if for every rule \( \ell \rightarrow r \) every variable occurring in \( \ell \) occurs exactly once in \( r \). For instance, the TRS consisting of the two rules for plus as given in the introduction, is variable preserving.

**Lemma 2.** Let \( R \) be a variable preserving orthogonal TRS. Then the relation \( \rightarrow_R \) on the set of all terms satisfies the diamond property.

**Proof.** (sketch)

This follows by analyzing all cases as they occur in the proof of the critical pair lemma ([3], Lemma 2.7.15): for the case of disjoint redexes it is immediate, for the case of nestingness it follows from variable preservation, and the third case of overlap does not occur due to orthogonality., for the case of nestingness it follows from variable preservation, and the third case of overlap does not occur due to orthogonality.

As a direct consequence of Lemmas 1 and 2 we conclude that for all terms with respect to the two plus rules from the introduction the number of steps to reach the normal form is independent of the strategy. This does not hold any more for the full system also containing the rules for mult. For instance, the term \( \text{mult}(0, \text{plus}(0, 0)) \) admits the following two reductions to normal form having lengths one and two, respectively:

\[
\text{mult}(0, \text{plus}(0, 0)) \rightarrow_R 0, \quad \text{mult}(0, \text{plus}(0, 0)) \rightarrow_R \text{mult}(0, 0) \rightarrow_R 0.
\]

\footnote{Some texts have a weaker notion of variable preserving, but our version is more suitable for investigating reduction lengths.}
However, here the starting term \(\text{mult}(0, \text{plus}(0, 0))\) is not basic: it contains more than one defined symbol. A symbol is called a defined symbol if it occurs as the root of the left-hand side of a rule. Similar as in texts on runtime complexity like [1] we define:

A term is defined to be basic if the root is a defined symbol, and it is the only defined symbol occurring in the term.

We will prove that for basic terms like terms of the shape \(\text{mult}(s^m(0), s^n(0))\) every reduction to its normal form \(s^{mn}(0)\) has the same length.

**Theorem 3.** Let \(R\) be an orthogonal TRS over \(\Sigma\), and \(\Sigma' \subseteq \Sigma\), such that every rule \(\ell \rightarrow r\) of \(R\) is of one of the following shapes:

- \(\ell \rightarrow r\) is variable preserving, and neither \(\ell\) nor \(r\) contain symbols from \(\Sigma'\),
- the root of \(\ell\) is in \(\Sigma'\), and for every symbol in \(r\) from \(\Sigma'\) the arguments of this symbol do not contain defined symbols.

Then any two reductions of a basic term to normal form have the same length.

**Proof.** (sketch)

Due to the shape of the rules, starting from a basic term the following property remains invariant during rewriting:

For every symbol from \(\Sigma'\) in the term, the arguments of this symbol do not contain defined symbols.

So every reduct of a basic term satisfies this invariant. Next we show that the relation \(\rightarrow_R\) restricted to terms satisfying this invariant, satisfies the diamond property. To prove this, let such a term \(s\) both rewrite to \(t\) and to \(u\), \(t \neq u\). In case the redexes are parallel, then the diamond property is easily concluded. In the remaining case one redex is above the other. Due to the invariant and the shape of the rules, one of the following cases holds:

- The roots of both redexes are not in \(\Sigma'\). Then both reduction steps are with respect to variable preserving rules, and the diamond property follows from Lemma 2.
- The root of the innermost (of the two) redexes is in \(\Sigma'\), but the root of the outermost redex is not. Then the reduction with respect to the outermost redex is variable preserving, by which the diamond property can be concluded.

In all cases the diamond property can be concluded. Now the theorem follows from Lemma 1.

Indeed now by Theorem 3 we can conclude that with respect to the TRS with \(\text{plus}\) and \(\text{mult}\) as given in the introduction for every basic term of the shape \(\text{mult}(s^m(0), s^n(0))\) every reduction to its normal form \(s^{mn}(0)\) has the same length: let \(\Sigma'\) consist of the single symbol \(\text{mult}\). Then the two rules for \(\text{plus}\) satisfy the condition for rules of the first type, and the two rules for \(\text{mult}\) satisfy the condition for rules of the second type, by which the claim follows from Theorem 3.

As another example consider the Fibonacci function \(\text{fib}\) defined by the rules

\[
\begin{align*}
\text{plus}(0, x) & \rightarrow x & \text{fib}(0) & \rightarrow 0 \\
\text{plus}(s(x), y) & \rightarrow s(\text{plus}(x, y)) & \text{fib}(s(0)) & \rightarrow s(0) \\
& & \text{fib}(s(s(x))) & \rightarrow \text{plus}(\text{fib}(x), \text{fib}(s(x))).
\end{align*}
\]
Choosing \( \Sigma' \) to consist only of the symbol \( \text{fib} \), all requirements of Theorem 3 hold, so we conclude that for every \( k \) every reduction of \( \text{fib}(s^k(0)) \) to normal form has the same length.

The converse of Theorem 3 does not hold, not even for terminating orthogonal constructor systems. For instance, for the TRS consisting of the three rules

\[
\begin{align*}
    f(0) & \rightarrow 0, \\
    f(s(x)) & \rightarrow s(0), \\
    g(x) & \rightarrow f(f(x))
\end{align*}
\]

it is easily proved that any two reductions of a basic term to normal form have the same length, while the conditions of Theorem 3 do not hold.

### 3 Binary arithmetic

For more efficient computation of numbers it is natural to exploit binary notation, in which the size of the representation is logarithmic rather than linear in the value of the number. In standard binary notation positive integers can be seen to be uniquely composed from a constant 1 representing value 1, and two unary operators \( .0 \) and \( .1 \), where \( .0 \) means putting a 0 behind the number, by which its value is duplicated, and \( .1 \) means putting a 1 behind the number, by which its value \( x \) is replaced by \( 2x + 1 \).

Every positive integer has a unique representation as a ground term over these three symbols 1, \( .0 \) and \( .1 \), corresponding to the usual binary notation in which a postfix notation for \( .0 \) and \( .1 \) is used. For instance, the number 29 is 11101 in binary notation, and is written as \( .1(.0(.1(.1(1)))) \). Introducing a constant 0 would violate unicity, that’s why we restrict to positive integers.

In order to express addition and multiplication in this notation the successor \( \text{succ} \) is needed as an additional operator, having rewrite rules

\[
\begin{align*}
    \text{succ}(1) & \rightarrow .0(1) \\
    \text{succ}(.0(x_p)) & \rightarrow .1(x_p) \\
    \text{succ}(.1(x_p)) & \rightarrow .0(\text{succ}(x_p))
\end{align*}
\]

Now we can express addition:

\[
\begin{align*}
    \text{plus}(1, x) & \rightarrow \text{succ}(x) \\
    \text{plus}(.0(x), .0(y)) & \rightarrow .0(\text{plus}(x, y)) \\
    \text{plus}(.0(x), .1(y)) & \rightarrow .1(\text{plus}(x, y)) \\
    \text{plus}(.1(x), .0(y)) & \rightarrow .1(\text{plus}(x, y)) \\
    \text{plus}(.1(x), .1(y)) & \rightarrow .0(\text{succ}(\text{plus}(x, y)))
\end{align*}
\]

Indeed now for every ground term composed from 1, .0, .1, \( \text{succ}, \text{plus} \) containing at least one symbol \( \text{succ} \) or \( \text{plus} \) a rule is applicable. So the ground normal forms are the ground terms composed from 1, .0, .1, exactly being the binary representations of positive integers. Since the TRS is easily proved to be terminating, e.g., by recursive path order, every ground term will reduce to such a ground normal form, being the binary representations of a positive integer. As by every rule the numeric value of the term is preserved, this TRS serves for computing the binary value of any ground term considered so far.

Surprisingly, it is very simple to extend this system to multiplication, by using the fresh symbol \( \text{mult} \) for multiplication and introducing the following rules

\[
\begin{align*}
    \text{mult}(1, x) & \rightarrow x \\
    \text{mult}(.0(x), y) & \rightarrow .0(\text{mult}(x, y)) \\
    \text{mult}(.1(x), y) & \rightarrow \text{plus}(.0(\text{mult}(x, y)), y)
\end{align*}
\]
The above observations are also easily checked for the extended TRS consisting of all rules presented so far: all rules preserve values, and for every ground term composed from $1, 0, 1, \text{succ}, \text{plus}, \text{mult}$ containing at least one symbol $\text{succ}$ or $\text{plus}$ or $\text{mult}$ a rule is applicable. So the ground normal forms are the ground terms composed from $1, 0, 1$, exactly being the binary representations of positive integers. Also the extended TRS is easily proved to be terminating, e.g., by recursive path order, so every ground term will reduce to such a ground normal form, being the binary representations of a positive integer. So the extended TRS also serves for executing multiplication.

Choosing $\Sigma'$ to consist of the single symbol $\text{mult}$ it is easily checked that all conditions of Theorem 3 are satisfied. So by Theorem 3 we conclude that for using this TRS for computing the addition or multiplication of two binary numbers, every reduction to normal form has the same length.

In [5] it is proved that for addition this reduction length is linear in the size of the arguments, and for multiplication it is quadratic in the size of the arguments, so having the same complexity as the standard algorithms for binary addition and multiplication. The TRSs as presented in [5] have been the basis of the implementation of integer number computation in the mCRL2 tool set [2].

4 Conclusion

Basic terms are typical terms to be rewritten: a defined symbol on top and constructor terms as arguments. We gave a criterion for orthogonal TRSs by which all reductions of a basic term to normal form have the same length, showing that the reduction length is independent of the chosen strategy. This applies to both addition and multiplication, both in unary and binary notation.

In unary notation the Fibonacci function still satisfies our criteria. However, for more complicated user defined functions experiments based on a simple implementation show that reduction lengths of basic terms are typically not strategy independent any more.

Acknowledgment. We want to thank Evans Kaigaje for fruitful discussions on this topic and for doing some experiments.

References

[1] Martin Avanzini and Georg Moser. Closing the gap between runtime complexity and polytime computability. In Christopher Lynch, editor, Proceedings of the 21st International Conference on Rewriting Techniques and Applications, volume 6 of Leibniz International Proceedings in Informatics (LIPIcs), pages 33–48, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
[2] J. F. Groote et al. The mCRL2 tool set. http://mcr12.org/mcr12/wiki/index.php/Home.
[3] Terese. Term Rewriting Systems. Cambridge University Press, 2003.
[4] Y. Toyama. Reduction strategies for left-linear term rewriting systems. In A. Middeldorp, V. van Oostrom, F. van Raamsdonk, and R. de Vrijer, editors, Processes, Terms and Cycles: Steps on the Road to Infinity: Essays Dedicated to Jan Willem Klop on the Occasion of His 60th Birthday, volume 3838 of Lecture Notes in Computer Science, pages 198–223. Springer, 2005.
[5] H. Zantema. Basic arithmetic by rewriting and its complexity. Available at http://www.win.tue.nl/~hzantema/aritm.pdf, 2003.