Separability and translatability of sequential term rewrite systems into the lambda calculus

Sugwoo Byun* Richard Kennaway† Vincent van Oostrom‡ Fer-Jan de Vries§

Abstract

Orthogonal term rewrite systems do not currently have any semantics other than syntactically-based ones such as term models and event structures. For a functional language which combines lambda calculus with term rewriting, a semantics is most easily given by translating the rewrite rules into lambda calculus and then using well-understood semantics for the lambda calculus. We therefore study in this paper the question of which classes of TRS do or do not have such translations.

We demonstrate by construction that forward branching orthogonal term rewrite systems are translatable into the lambda calculus. The translation satisfies some strong properties concerning preservation of equality and of some inequalities. We prove that the forward branching systems are exactly the systems permitting such a translation which is, in a precise sense, uniform in the right-hand sides.

Connections are drawn between translatability, sequentiality and separability properties. Simple syntactic proofs are given of the non-translatability of a class of TRSs, including Berry's F and several variants of it.

*Address: Kyungsung University, Korea. Email `swbyun@star.kyungsung.ac.kr`

†Address: School of Information Systems, University of East Anglia, Norwich, NR4 7TJ, UK. Email: `jrk@sys.uea.ac.uk`

‡Address: Faculteit Wijsbegeerte, Universiteit Utrecht, P.O. Box 80.126 3508 TC Utrecht, the Netherlands. Email `oostrom@phil.uu.nl`

§Address: Department of Mathematics and Computer Science, University of Leicester, Leicester, LE1 7RH, UK. Email `ferjan@mcs.le.ac.uk`

Contents

1 Introduction

Our previous work [BKS95, BKS96] on translation of TRSs into lambda calculus has considered only constructor TRSs, that is, those in which a symbol which appears at the root of a left-hand side does not appear anywhere else in any left-hand side. In this paper we consider a larger class of orthogonal systems which can be translated into constructor systems. We generalise the notion of separability from [BKS95, BKS96], and show by construction that all strongly separable orthogonal systems have a translation into a constructor system of a very simple form, which in turn has a translation to lambda calculus. The translation is, in a sense we will define, uniform in the right-hand sides of the original system. We also obtain simple syntactic proofs of the non-translatability of several systems, such as variants of Berry's F.

The translation problem can be motivated from the viewpoint of functional programs as TRSs, to be given meaning by translation to lambda calculus, or implementation via implementations of lambda calculus such as in [AG98]. One can also consider functional programs as lambda expressions (usually in some form of typed lambda calculus), and define implementations by translation into rewrite systems, such as in [Oho99]. Reverse compilation can be seen as translating such a rewrite system back into lambda calculus.

2 Definitions

2.1 Basics

We assume familiarity with the basic concepts and notations of term rewrite systems.

The function symbols of a TRS can be divided into two classes: *operators* or *defined symbols*, which appear at the root of some left-hand side, and *constructors*, symbols which do not appear at the root of any left-hand side. A *constructor* TRS is one in which no operator appears in any left-hand side except at the root. We mention this class primarily to make it clear that we do not restrict attention to it: in the systems we consider, operator symbols may appear anywhere in a left-hand side. We will primarily be concerned with orthogonal TRSs, although some of our results apply to the wider class of left-linear TRSs.

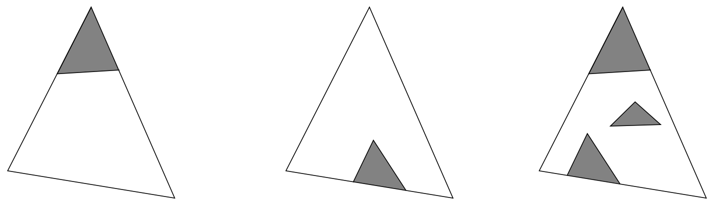

Figure 1: Kinds of parts of terms

t is a *prefix* of t' if t' is an instance of t, and for every position u of a variable x in t such that $t'|_u \neq x$, x occurs in t only at u. A prefix which is just a variable is *empty*. Thus $F(x,x,y)$ is not a prefix of $F(A,A,B)$, $F(x,y,B)$, or $F(A,x,B)$, but it is a prefix of $F(x,x,B)$. In practice, we will only be considering prefixes of linear terms, i.e. terms containing no repeated variables. These are always linear.

In a given TRS, a *redex prefix* is a non-empty prefix of a left-hand side. A *component* of a term t is a non-empty prefix of a subterm of t. A *constructor schema* is either a non-variable proper subterm of a left-hand side, or a term $C(\overline{x})$ where C is a symbol not occurring in any left-hand side. A *constructor normal form* is any term which can be obtained from a constructor schema by replacing any of its variables by variables or constructor normal forms. See Figure 1.

Constructor schemas generalise the concept of constructor symbols. If t is a term having some constructor schema t' as a prefix, then every reduct of t has t' as a prefix. Furthermore, no reduct of t can contain a redex whose root lies within t'. Thus t' is a "fixed part" of t. In particular, t is root-stable.

In a TRS having just the left-hand side $F(F(A(B(x))))$, the redex prefixes are $F(F(A(x)))$, $F(F(x))$, and $F(x)$. The constructor schemas are $F(A(B(x)))$, $A(B(x))$, and $B(x)$. The components of $F(F(A(B(x))))$ are the redex prefixes and the constructor schemas, plus $F(A(x))$ and $A(x)$. Notice that $F(x)$ occurs twice as a component of $F(F(A(B(x))))$.

2.2 Translations

A *translation* of a signature Σ into lambda calculus is a mapping $[\![\,]\!]$ from Σ to closed lambda expressions. This can be extended to all terms by defining:

$$[\![F(t_1,\ldots,t_n)]\!] = [\![F]\!]\,[\![t_1]\!]\ldots[\![t_n]\!] \tag{1}$$

$$[\![x]\!] = x \tag{2}$$

Given a TRS R over Σ, a *translation* of R is a translation of Σ which satisfies certain conditions. What these conditions should be is not obvious. Many different concepts of a translation have appeared in the literature. At a minimum, it should certainly map each rule of R to an equality of lambda calculus, and thus map all equalities between terms to equalities of lambda calculus. If this were the only condition, many TRSs would have a trivial translation which mapped all terms other than variables to the same lambda expression, simply by defining the translation of each n-ary symbol F to be $\lambda x_1 \ldots x_n x.x$. It is not enough to require preservation of equality: some inequalities should also be preserved. However, if we have two rules $A(x) \to x$ and $B(x) \to x$, it does not seem reasonable to disallow translating both A and B by $\lambda x.x$. The terms whose inequality we will require to be preserved are the constructor schemas. In an orthogonal TRS, constructor schemas that differ other than in the names of their variables are never equal. We will require that they are translated into lambda expressions which are not merely unequal, but in a sense provably unequal, as defined by the notion of strong separability.

DEFINITION 2.1 A tuple of lambda expressions $t_1,\ldots,t_n$ is *separable* if for any tuple of lambda expressions $s_1,\ldots,s_n$ there is a context $C[\,]$ such that $C[t_i] = s_i$ for all i.

It is *strongly separable* if $C[\,]$ can be chosen so as to contain (whether free or bound) none of the free variables of $t_1,\ldots,t_n$.

$C[\,]$ is called a *(strongly) separating context.*

In the above definition, we can without loss of generality take $s_1,\ldots,s_n$ to be a tuple of distinct new variables.

Separability is as defined in [Bar84], although we find it technically more convenient to speak of tuples rather than sets of expressions. Note that a tuple containing repeated elements is never separable. Strong separability is our extension of the concept. The terms x and y are separable (by the context $(\lambda xy.[\,])s_1 s_2$) but they are not strongly separable. In this paper we are primarily concerned with strong separability.

We will also require that the free variables of a constructor schema be "exposed" in its translation, in the following sense.

DEFINITION 2.2 An occurrence at position u of a free variable x of a lambda-expression t is *exposed* if there is a context $C[\,]$ containing none of the free variables of t such that $C[t]$ reduces to x, and the final occurrence of x is a descendant of u. x is exposed in t if some occurrence of x is.

The property of x being exposed in t is equivalent to that of $\lambda x\overline{y}.t$ being *left-invertible* [BDC74], where $\overline{y}$ is a tuple of all the free variables of t excluding x. Left-invertibility of a lambda expression M means that there is a lambda expression N such that for a new variable x, $N(Mx) = x$.

We can now present our notion of a translation.

DEFINITION 2.3 A *translation* of a TRS R is a translation of its signature (and hence of all its terms) which has the following properties.

(1) It maps every rule of R to an equality of lambda calculus.

(2) It maps every instance of every constructor schema to an expression having a head normal form.

The translation is *proper* if in addition:

(3) It maps every constructor schema to an expression in which every variable occurring in the schema is exposed.

(4) It maps every tuple of distinct constructor schemas to a strongly separable tuple of lambda expressions.

Note that condition (3) implies condition (2) for constructor schemas having at least one free variable, since no variables are exposed in a term having no head normal form. Our reason for distinguishing translations from proper translations is that some of our results will demonstrate that certain systems have no translation, not even an improper one.

LEMMA 2.4 *In lambda calculus, if x is exposed at u in s and y is exposed at v in t then y is exposed at $u \cdot v$ in $s[u := t]$.*

PROOF. Let $C[\ ]$ expose x in s at u and $C'[\ ]$ expose y in t at v. Then $C'[C[\ ]]$ exposes y at $u \cdot v$ in $s[u := t]$. $\square$

LEMMA 2.5 *In lambda calculus, if x is exposed at u in t then x is exposed at u in any term t' obtained by substituting any terms for any occurrences of free variables in t other than at u. (Different occurrences of the same variable need not be substituted by the same terms, and occurrences of x other than at u may be substituted.)*

PROOF. Suppose we are given a reduction of $C[t]$ to x, where the final x is a residual of the occurrence of x at u in t. This reduction can clearly also be performed on any such term t' and yields the same result. $\square$

LEMMA 2.6 *Let t be a constructor schema of the form $t'[\overline{x} := \overline{t}]$, where t' is also a constructor schema, $\overline{x}$ consists of all the free variables of t, and the terms $\overline{t}$ are not variables (which implies that they are constructor schemas). If condition (3) of Definition 2.3 holds for t' and $\overline{t}$, it holds for t.*

PROOF. Let x be a free variable of t. It must occur in some $t'' \in \overline{t}$ which is substituted for a variable y in t'. By condition (3) applied to t' and t'', there are contexts $C'[\ ]$ and $C''[\ ]$ such that $C'[[\![t']\!]] = y$ and $C''[[\![t'']\!]] = x$. We then have $C''[C'[t]] = x$. $\square$

In the constructor TRSs which arise in practice, this lemma usually allows properness of a translation to be established by checking condition (3) of Definition 2.3 just for the terms $C(\overline{x})$, for constructor symbols C.

2.3 Uniform translations

The left-hand sides of a TRS specify what is a redex. The right-hand sides specify how to reduce each type of redex. We are interested in translations which use as little information as possible about the right-hand sides. The lambda expressions to which such translations map the function symbols embody a strategy for finding redexes which is determined solely by the left-hand sides of the system. We call these uniform translations.

DEFINITION 2.7 Let $\mathcal{F}$ be a set of function symbols, and $(l_1, \ldots, l_n)$ be a tuple of terms over $\mathcal{F}$. Let $\mathcal{R}$ be the set of all orthogonal TRSs having $\mathcal{F}$ as its set of function symbols and $(l_1, \ldots, l_n)$ as its set of left-hand sides. Suppose that a (proper) translation $[\![\,]\!]_R$ is given for every system R in $\mathcal{R}$. We call this a *uniform* (proper) translation of $\mathcal{R}$ if for each symbol F there is a lambda expression $[\![F]\!]_u$ such that for each system R with rules $\{l_i \rightarrow r_i | i : 1 \ldots n\}$ the following equations hold in lambda-calculus:

$$[\![F]\!]_R \;=\; [\![F]\!]_u(\lambda\overline{x}_1.[\![r_1]\!]_R)\ldots(\lambda\overline{x}_n.[\![r_n]\!]_R) \tag{3}$$

$$[\![l_i]\!]_u\overline{a} \;=\; a_i\overline{x} \tag{4}$$

In both equations, $\overline{x}_i$ is the tuple of all variables occurring in the left-hand side of the ith rule, from left to right. In equation (4), $\overline{a}$ is a tuple of n variables $a_1 \ldots a_n$ and $[\![l_i]\!]_u$ is defined by extending $[\![\,]\!]_u$ to all terms by the rules:

$$[\![F(t_1, \ldots, t_k)]\!]_u \;=\; \lambda\overline{a}.[\![F]\!]_u\overline{a}([\![t_1]\!]_u\overline{a})\ldots([\![t_k]\!]_u\overline{a}) \tag{5}$$

$$[\![x]\!]_u \;=\; \lambda\overline{a}.x \tag{6}$$

Equations (5) and (6) are equations (1) and (2) with everything parameterised by the right-hand sides.

We call a TRS *uniformly translatable* if there is a uniform translation of the set of all TRSs having the same left-hand sides.

LEMMA 2.8 *With the notation of Definition 2.7, for every term t,*

$$[\![t]\!]_R \;=\; [\![t]\!]_u(\lambda\overline{x}_1.[\![r_1]\!]_R)\ldots(\lambda\overline{x}_n.[\![r_n]\!]_R)$$

PROOF. For brevity, we write ρ_i for $\lambda\overline{x}_i.[\![r_i]\!]_R$ and $\overline{\rho}$ for $\rho_1 \ldots \rho_n$. We are to prove that $[\![t]\!]_R = [\![t]\!]_u\overline{\rho}$.

$$
\begin{array}{rcll}
[\![x]\!]_R & = & x & \text{definition} \\
 & = & (\lambda\overline{x}.x)\overline{\rho} & \text{beta expansion} \\
 & = & [\![x]\!]_u\overline{\rho} & \text{definition}
\end{array}
$$

$$
\begin{array}{rcll}
[\![F(t_1,\ldots,t_k)]\!]_R & = & [\![F]\!]_R\ [\![t_1]\!]_R \ldots [\![t_k]\!]_R & \text{definition} \\
 & = & [\![F]\!]_R\ ([\![t_1]\!]_u\overline{\rho}) \ \ldots\ ([\![t_1]\!]_u\overline{\rho}) & \text{induction} \\
 & = & [\![F]\!]_u\overline{\rho}\ ([\![t_1]\!]_u\overline{\rho}) \ \ldots\ ([\![t_1]\!]_u\overline{\rho}) & \text{equation 3} \\
 & = & (\lambda\overline{x}.[\![F]\!]_u\overline{x}\ ([\![t_1]\!]_u\overline{x}) \ \ldots\ ([\![t_k]\!]_u\overline{x}))\overline{\rho} & \text{beta expansion} \\
 & = & [\![F(t_1,\ldots,t_k)]\!]_u\overline{\rho} & \text{equation 5}
\end{array}
\qquad \square
$$

Notice that this proof does not use equation 4, and proves a special case of that equation:

$$
[\![l_i]\!]_u(\lambda\overline{x}_1.[\![r_1]\!]_R) \ \ldots\ (\lambda\overline{x}_n.[\![r_n]\!]_R) \quad = \quad \lambda\overline{x}_i.[\![r_i]\!]_R
$$

2.4 Sequentiality

We recall here some results from [Mid90] on sequentiality, which we will use in section 4 to demonstrate connections with separability.

Definition 2.9 We adjoin two new nullary function symbols Ω and $\bullet$. An Ω-*instance* of a term t is a term obtainable from t by replacing some occurrences of Ω by arbitrary terms. Define the reduction rule $t \rightarrow_\Omega t'$ if $t' = t[u := \Omega]$, where $t|_u$ is not Ω, and has an Ω-instance which is a redex (of the ordinary rules). NF_s is the set of terms which contain at least one occurrence of Ω and no redexes, and which reduce to Ω by Ω-reduction. An *index* of a term t in NF_s is a position u such that $t|_u = \Omega$, and $t[u := \bullet] \not\rightarrow^*_\Omega \Omega$.

An index of t is *transitive* if for every $t' \in NF_s$ and every index v of t', $u \cdot v$ is an index of $t[u := t']$.

An orthogonal TRS *has transitive indexes* if every index of every term in NF_s is transitive.

This notion of index is related to the notion of an index of a predicate, which is fundamental to definitions of various forms of sequentiality. Middeldorp has shown ([Mid90], Lemma 6.2.7) that the indexes of the predicate nf' defined by Huet and Lévy ([HL91]) coincide with the indexes of the above definition. As the latter definition is technically easier to work with, and we do not in this paper make any use of indexes of any other predicate, we use Middeldorp's characterisation as the definition.

THEOREM 2.10 *([Mid90], Proposition 6.2.9.) If $u \cdot v$ is an index of t then u is an index of $t[u := \Omega]$ and v is an index of $t|_u$.* □

THEOREM 2.11 *([Mid90], Proposition 6.2.16.) An orthogonal TRS is strongly sequential if and only if every term in NF_s has an index.*

3 Separability

For TRSs we can define concepts of separability which relate to those for lambda calculus.

DEFINITION 3.1 Let $\mathcal{L}$ be a set of linear terms. Let t be a proper prefix of a member of $\mathcal{L}$. Let x be a free variable of t at position u. u is an *extension site* of t (for $\mathcal{L}$) if for every occurrence v of t as a component of any member l of $\mathcal{L}$, $l|v \cdot u$ is not a variable.

If every proper prefix of every member of $\mathcal{L}$ has an extension site, then we say that $\mathcal{L}$ is *strongly separable.*

A tuple of terms is strongly separable if it contains no repetitions, and it is strongly separable considered as a set.

A tuple of terms $t_1, \ldots, t_n$ is *separable* if some tuple $t'_1, \ldots, t'_n$ of respective prefixes of the given terms is strongly separable.

When the tuple of left-hand sides of a TRS is (strongly) separable, we say that the TRS is (strongly) separable.

As an example, let $\mathcal{L} = \{F(A,x), G(F(C,D))\}$. The term $F(x,y)$ is a proper prefix of one member of $\mathcal{L}$, and occurs one more time as a component of a member of $\mathcal{L}$. The position 1 of $F(x,y)$ is instantiated by both occurrences of $F(x,y)$ in $\mathcal{L}$, but the position 2 is only instantiated by one of them. Therefore 1 is an extension site of $F(x,y)$ but 2 is not.

The set $\{H(G(A,A,x),A), H(G(A,x,A),B), G(B,B,B)\}$ is strongly separable, but $\{H(G(A,x),A), H(G(x,A),B), G(B,B)\}$ is not. The latter tuple is separable, since $\{H(x,A), H(x,B), G(B,B)\}$ is strongly separable. The tuple $F(x,A,B), F(B,x,A), F(A,B,x)$ is not separable.

Notice that in the first example (taken from [SS97]), $G(A, x, y)$ occurs as a component twice, but neither of its variables is instantiated by both occurrences. However, since $G(A, x, y)$ is not a prefix of any member of the set, this does not imply non-separability. In the second example, $G(x, y)$ is a prefix with no extension site, therefore the set is not strongly separable. It is separable, because it has the strongly separable tuple of prefixes $\{H(x, A), H(x, B), G(B, B)\}$. If we add $H(C, A)$ and $H(D, B)$ to the set, it becomes non-separable.

LEMMA 3.2 *Let $C[\ ]$ be an n-ary context, and let there be given terms $s_{i,j}$ ($1 \leq i \leq m$, $1 \leq j \leq n$), such that $\forall j.\exists i, i'.s_{i,j}$ is a variable and $s_{i',j}$ is not. Let $t_i = C[s_{i,1}, \ldots, s_{i,n}]$. Then for any translation, $[\![\overline{t}]\!]$ is not strongly separable.*

PROOF. Suppose that $[\![\overline{t}]\!]$ is strongly separable. Choose m new variables $\overline{x}$. Then there must be a context C' containing none of the variables of any t_i such that $C'[t_i] = x_i$ for $i \leq m$. $C'[t_i] = C'[C[s_{i,1}, \ldots, s_{i,n}]]$. Now consider the term $C'[C[\overline{z}]]$, where $\overline{z}$ is a tuple of n new variables. This must have a head normal form.

If its head variable is some z which is not in $\overline{z}$, then z must be the head variable of $C'[C[s_{i,1}, \ldots, s_{i,n}]]$ as well, contradicting the fact that $C'[C[s_{i,1}, \ldots, s_{i,n}]] = x_i$.

Therefore its head variable must be some $z_j \in \overline{z}$. Let i and i' be such that $s_{i,j}$ is a variable, say x, and $s_{i',j}$ is not a variable. Then x is the head variable of the head normal form of $C'[C[s_{i,1}, \ldots, s_{i,n}]]$, contradicting the fact that $C'[C[s_{i,1}, \ldots, s_{i,n}]] = x_i$. □

THEOREM 3.3 *In a given TRS, let t be a redex-prefix having no extension site. Let $\overline{r}$ be the tuple of right-hand sides of the rules whose left-hand sides t is a prefix of. Let $\overline{s}$ be the tuple of distinct instances of t which occur as proper subterms of left-hand sides. Let $\overline{t}$ be the concatenation of $\overline{s}$ and $\overline{r}$. Then for any translation, $[\![\overline{s}]\!]$ is not strongly separable.*

PROOF. The maximal common prefix of $\overline{t}$ is some n-ary context $C[\ ]$. Let $t_i = C[s_{i,1}, \ldots, s_{i,n}]$. The absence of an extension site implies that the terms $s_{i,j}$ satisfy the hypothesis of Lemma 3.2. This establishes the theorem. □

This immediately establishes the non-translatability for certain TRSs.

THEOREM 3.4 *The following set of rules ("Berry's F") has no proper translation:*

$$\begin{aligned} F(x, A, B) &\to 0 \\ F(B, x, A) &\to 1 \\ F(A, B, x) &\to 2 \end{aligned}$$

PROOF. The prefix $F(x, y, z)$ has no extension site. The tuple $\bar{t}$ considered by Theorem 3.3 is $(0, 1, 2)$. The theorem implies that any translation must map these to a tuple which is not strongly separable. But it is a tuple of distinct constructors, so the translation cannot be proper. □

Another way of putting this theorem is that if 0, 1, and 2 are taken to be the Church numerals (or the numerals of any other numeral system in the lambda calculus) then the equations have no solution in lambda calculus. This has previously only been proved by domain-theoretic arguments.

Note that the system has many improper translations. For example, let $[\![F]\!] = \lambda xyz.t$, where t is closed, and define $[\![0]\!] = [\![1]\!] = [\![2]\!] = t$. $[\![A]\!]$ and $[\![B]\!]$ are arbitrary.

THEOREM 3.5 *Every orthogonal TRS with a uniform translation is strongly separable.*

PROOF. Let there be given an orthogonal TRS which is not strongly separable, and a uniform translation $[\![\,]\!]$ of it. We will derive a contradiction.

By non-strong separability there is a proper redex prefix t with no extension site. Let x be a free variable of t. Choose a left-hand side l_x which contains t as a component and does not instantiate x. (This is possible by the absence of an extension site for t.) Take a TRS R in which the corresponding right-hand side is x. Then $\mathcal{B}([\![l_x]\!]_R) = x$. Because l_x is an instance of t, it follows that $\mathcal{B}([\![t]\!])$ must contain x free. Therefore every free variable of t is free in $\mathcal{B}([\![t]\!])$.

Let x be the variable having the leftmost-outermost occurrence in $\mathcal{B}([\![t]\!])$. Let l_x be as above. Choose a TRS R in which x is not free in r_x. There exist lambda expressions $M_1, \ldots, M_k$ in which x is not free such that $[\![t]\!]M_1, \ldots, M_k = x$. Therefore $[\![l_x]\!]M_1, \ldots, M_k = x$. Therefore $[\![r_x]\!]M_1, \ldots, M_k = x$. But x is not free in r_x, contradiction. □

4 Connections between separability and sequentiality

4.1 Strong sequentiality and index transitivity

Suppose that t and t' are terms in NF_s and that t has an index at u. The property of strong sequentiality implies that for at least one index v of t', $u \cdot v$ is an index of t. Strong separability makes the additional requirement that an index v can be chosen independently of t and u. v is what we have called an extension site of t'. We could also call it a *separating index* of t'. Index transitivity makes the even stronger requirement that for every index v of t', $u \cdot v$ is an index of t, i.e. that every index of every redex-prefix is separating. These remarks establish the next theorem.

THEOREM 4.1 *If an orthogonal TRS is strongly sequential and has transitive indexes it is strongly separable. If an orthogonal TRS is strongly separable it is strongly sequential.* □

THEOREM 4.2 *Neither of the implications of Theorem 4.1 can be reversed.*

PROOF. The following system is strongly separable but does not have transitive indexes.

$$\begin{aligned} F(A, B) &\rightarrow \ldots \\ G(F(C, x)) &\rightarrow \ldots \end{aligned}$$

The proper redex prefixes are $F(x, y)$, $F(A, x)$, $F(x, B)$, $G(x)$, and $G(F(x, y))$. We have chosen the variables such that in each prefix, x is instantiated in every occurrence of that prefix in a left-hand side. Therefore the system is strongly separable. Consider the terms $G(\Omega)$, $F(\Omega, \Omega)$, and $G(F(\Omega, \Omega))$. The first two have indexes at 1 and 2 respectively, but $1 \cdot 2$ is not an index of the third. Therefore the system does not have transitive indexes.

Now consider the system obtained by adding the following rule to the above:

$$H(F(x, D)) \rightarrow \ldots$$

This system is still strongly sequential, but the redex-prefix $F(x, y)$ does not have any variable which is instantiated by every occurrence of $F(x, y)$ in a left-hand side. $F(x, y)$ thus has no extension site, and the system is not strongly separable. $\square$

THEOREM 4.3 *Let there be given an orthogonal TRS. For each redex prefix t, let U be the set of positions of free variables of t which are instantiated by every occurrence of t as a prefix of a left-hand side. Then the system is strongly sequential with transitive indexes if and only if for every proper redex prefix t, the set U is nonempty, and every member of U is an extension site of t.*

PROOF. Let t be a proper redex prefix and U be as described. Then $t_\Omega \in NF_s$ and U is the set of indexes of t_Ω. If U is empty, the system is not strongly sequential. Supose that U is nonempty, but some member u of U is not an extension site of t. Then there is a left-hand side t' in which t occurs as a redex-component at some position v, such that $t' \mid v \cdot u$ is a variable. Let $t'' = t'_\Omega[v := \Omega]$. Then v is an index of t'', u is an index of t_Ω, and $v \cdot u$ is not an index of $t''[v := t_\Omega]$. This demonstrates that the system does not have transitive indexes.

For the converse, suppose that the condition on U holds for all redex prefixes. Let $t \in NF_s$. We will define a set $I(t)$ of positions of Ω in t by an inductive construction, and prove that these are the indexes of t.

For any position u of t, define $\Omega_u(t)$ to be the normal form of t by Ω-reduction subject to the constraint that no reduction is performed at any prefix of u. Let $t' = \Omega_{\langle\rangle}(t)$. Then t' must be a redex-prefix. Let U be the positions of t' which are instantiated by every occurrence of t' as a redex prefix. U is nonempty and every member is an extension site of t'.

If $t' = t$ (the base case of the induction) then every member of U is an index of t. Define $I(t) = U$ and $I'(t) = U$.

Otherwise, consider the terms $t_u = t|u$ for all $u \in U$. Each of these is smaller than t, so we apply this construction to obtain for each one a set $I(t_u)$. Define $I(t) = \{u \cdot v | v \in I(t_u)\}$ and $I'(t) = U \cup I(t)$.

It is immediate from this construction that $I(t)$ is nonempty. The auxiliary set $I'(t)$ will be technically useful in the proof. It has the important property that for all $u \in I(t)$, and all $v < u$ such that $t|v \to^*_\Omega \Omega$, $v \in I'(t)$.

We can now prove that $I(t)$ consists of indexes of t. Suppose that for some t, some member of $I(t)$ were not an index. Take t to be of minimal size such that this is so, and let $u \in I(t)$ be a non-index. Let $t' = \Omega_u(t)[u := \bullet]$. This must contain an Ω-redex at one or more prefixes of u. Consider the innermost such redex, at position v. Let $u = v \cdot w$. We know that $v \in I'(t)$ and $w \in I(t'|v)$. By induction, if $t'|v$ is smaller than t, then w is an index of $t'|v$, and therefore $t'[u := \bullet]$ cannot have an Ω-redex at v, contradiction. Therefore $t'|v$ must be t itself. As a member of $I(t)$, u is constructed as a chain $v \cdot w \cdot \ldots$, where each of v, $v \cdot w$, etc. is in $I'(t)$. If there were exactly one segment, then $t[u := \bullet]$ would not be an Ω-redex. Otherwise, the tail $u' = w \cdot \ldots$ is not only an index of $I(t|v)$, but an extension site of it. This implies that in this case also, $t[u := \bullet]$ cannot be an Ω-redex.

This establishes that every member of $I(t)$ is an index of t. If all other occurrences of Ω in t are replaced by $\bullet$, the construction of $I(t)$ allows one to Ω-reduce the resulting term to Ω. Therefore $I(t)$ contains all the indexes of t.

It remains to prove that indexes are transitive. Suppose that t and t' are in NF_s, $u \in I(t)$, and $v \in I(t')$. Then $u \in I'(t[u := t'])$, and $u \cdot v \in I(t[u := t'])$. $\square$

For constructor systems, all strongly sequential systems are index-transitive ([Mid90]), and therefore for constructor systems, the strongly separable systems coincide with both of the other classes.

4.2 Forward branching and inductive sequentiality

Inductive sequentiality is defined by [HLM98], which proves it equivalent to the forward branching systems defined by [Str89, Dur94]. We shall prove both equivalent to strong separability for orthogonal systems.

The definition of forward branching systems is easier to work with. These are defined in terms of the notion of an index tree for an orthogonal TRS. (Despite the name, it is actually a type of graph.)

An index tree for an orthogonal TRS is a finite state automaton. Its non-terminal states are pairs (t, u), where t is a redex prefix and u is an index of t, and its terminal states are the left-hand sides of the system. Its initial state is the pair $(\Omega, \langle\rangle)$. Its transition function is in two parts: the transfer function and the failure function. The transfer function maps pairs

$((t,u),A)$, where (t,u) is a state and A is a function symbol, to states of the index tree. If $((t,u),A)$ is mapped to state s, this should be interpreted as a transition labelled A from (t,u) to s. s always has the form (t',u') or t', where $t' = t[u := A(\overline{x})]$. The failure function defines a set of unlabelled transitions from some non-terminal states to states.

An index tree defines a reduction algorithm for the TRS. We will not go into detail, but the intuition is that when one is at a state (t,u), one has established that t is a prefix of the term being evaluated, and u is the position of the node of the term to be inspected next. The transfer function defines what state to go to for each of the symbols that can extend t at u and still have a redex prefix. The failure function defines what to do if some other symbol is found. In that case, some subterm of t may have to be evaluated instead, and the failure transition will go to a state labelled with that subterm.

A *forward branching* index tree is one in which every node is accessible from the initial state without failure transitions. What this means is that if a failure transition from (t,u) goes to a state labelled with $t' = t|v$ (either a terminal or a non-terminal state), then if we had begun by using the index tree to evaluate a term of which t' was a prefix, we would have arrived at that same state. The evaluation strategy proceeds independently of the context in which the term with prefix t' occurred.

Note that for forward branching index trees, in each non-terminal state (t,u), u is determined by t, so we can consider the state to be redex prefixes and the chosen index to be given by a function U from non-terminal states to indexes.

An orthogonal TRS is defined to be *forward branching* if there is a forward branching index tree whose terminal states are its left-hand sides. The example of a non-strongly separable system in section 3 is also not forward branching. This is not a coincidence. Notice how the decision of which argument of a term $G(\ldots,\ldots)$ to evaluate next depends on the context it was encountered: if as part of a term $H(G(\ldots,\ldots),A)$, then the first, but if as part of $H(G(\ldots,\ldots),B)$, then the second. This is an example of a general connection between strong separability and forward branching.

THEOREM 4.4 *An orthogonal TRS is strongly separable if and only if it is forward branching.*

PROOF.

Only if: The definition of strong separability immediately suggests a construction of a forward branching index tree. We construct it node by node, starting with just the initial state (Ω). If t is a non-terminal state, we write $U(t)$ for the position u associated with that state, and $T(t, A)$ for the transition function at that state applied to symbol A.

We will endeavour to maintain the following invariant of the partial index tree at each stage in the construction. If t is one of the non-terminal states, and there is a v such that $\langle\,\rangle < v < U(t)$ and $t|v$ is a redex-prefix, then $t|v$ is a state that has already been added to the index tree and $U(t) = v \cdot U(t|v)$. This is trivially satisfied initially.

At each stage, we select a proper redex-prefix t already in the set of states such that $U(t)$ is not yet defined, and choose t to be of minimal depth such that this is so. Strong separability ensures t has an extension site. Let u be any extension site of t. If there is no v such that $\langle\,\rangle < v < u$ and $t|v$ is a redex-prefix, then define $U(t) = u$. Otherwise, choose the shortest such v. By the invariant, $U(t|v)$ is defined. Define $U(t) = v \cdot U(t|v)$ (which may be different from u).

It remains to define the transitions from t. For each symbol A, such that $t' = t[U(t) := A(\overline{x})]$ is a redex-prefix, define $T(t, A) = t'$.

If there is a v such that $\langle\,\rangle < v < U(t)$ and $t|v$ is a redex-prefix, then take the shortest such v and define $F(t) = t|v$. By the invariant, $t|v$ is already in the set of states.

This construction can terminate exactly when every left-hand side appears as a terminal state of the tree. The invariant ensures that failure transitions only go to nodes that were already reachable without failure transitions. Therefore the tree is forward branching.

If: Suppose that we have a forward branching index tree for a TRS. We must find an extension site for every proper redex-prefix. Let t is a proper redex-prefix. Let t' be a maximal prefix of t which is a state (necessarily non-terminal) of the index tree. Let $u = U(t')$. If $t|u$ was a term with root symbol A, then $t'[u := A(\overline{x})]$ would be a larger prefix of t occurring as a state of the index tree. Therefore $t|u = \Omega$. Since u is an extension site of t', it is also an extension site of t. $\square$

This result also shows why strong separability is the relevant concept, rather than separability. Strong separability implies that the whole of redex pattern can be matched by an algorithm which at each node, chooses which descendant to next inspect based only on the last operator symbol seen. Separability only guarantees that such an algorithm is capable of distinguishing the different redex patterns, without necessarily exploring the entire pattern. Consider the earlier example of a separable but not strongly separable tuple: $\{H(G(A, x), A), H(G(x, A), B), G(B, B)\}$. A forward branching index tree can match the following strongly separable tuple of prefixes: $\{H(x, A), H(x, B), G(B, B)\}$. If one assumes that the term that one is matching is indeed an instance of one of the three given terms, then this is sufficient to determine which one, but it does not explore the entire pattern.

5 Non-translatable systems

Some orthogonal systems have no translation, not even a non-uniform one. By establishing some sequentiality properties of the lambda calculus, we will demonstrate the non-translatability of a class of systems and give some examples, including another variant of Berry's F. The proofs are purely syntactic and quite straightforward.

We adjoin to lambda calculus the symbol Ω, and extend the notion of head normal form by stipulating that lambda expressions of the form $\Omega t_1 \dots t_n$ are not head normal forms. Positions of a lambda expression are defined by regarding "$\lambda x.$" as a unary operator for each x, and application as a binary operator that is left unwritten in the usual notation. Thus $\lambda w.x(yy)(zx)|1 \cdot 2 \cdot 1 = z$. The *left spine* of a lambda expression t is the set of all its positions which are of the form $1 \dots 1$ (including the empty position $\langle\,\rangle$).

LEMMA 5.1 *Let $C[\dots]$ be an n-ary context of lambda calculus. For $i : 1 \dots m$ and $j : 1 \dots n$, let t_{ij} be a lambda expression, such that $\forall j \exists i.\, t_{ij} = \Omega$. If for all i, $C[t_{i1}, \dots, t_{in}]$ has a head normal form, then $C[t_1, \dots, t_n]$ has a head normal form for any lambda expressions $t_1, \dots, t_n$.*

PROOF. $C[\overline{x}]$ must have a head normal form (otherwise its instances $C[t_{i1}, \dots, t_{in}]$ could not). So without loss of generality we may assume that $C[\dots]$ is already in head normal form. Let the head variable of $C[\overline{x}]$ be x. If x is not in $\overline{x}$, then x must be the head variable of $C[\overline{t}]$ for any tuple of lambda expressions $\overline{t}$. Therefore $C[\overline{t}]$ has a head normal form.

Otherwise, x is some $x_j \in \overline{x}$. But there is an i such that $t_{ij} = \Omega$. So for that i, $C[t_{i1}, \dots, t_{in}]$ reduces to a lambda expression $\lambda\overline{z}.\Omega \dots$, which has no head normal form, contradiction. □

Lemma 5.2 *Let C be a function symbol and $n > 1$. When $1 \leq i \leq m$ and $1 \leq j \leq n$, let $A_{i,j}$ be either a function symbol or a variable. Consider the set of equations*

$$C A_{i,1} \ldots A_{i,n} = x_i \quad (1 \leq i \leq n)$$

where x_i is a variable occurring in $A_{i,1} \ldots A_{i,n}$. Suppose that $\forall j. \exists i. A_{i,j} = x_i$, and that $\forall j. \exists i'. A_{i',j} \neq x_{i'}$. Then the equations have no solution in the lambda calculus.

Proof. Suppose that there is a solution, which maps C to a closed lambda expression C', and each $A_{i,j}$ which is not a variable to a closed lambda expression $A'_{i,j}$. When $A_{i,j}$ is a variable, define $A'_{i,j} = A_{i,j}$. Then each of the lambda expressions $C' A'_{i,1} \ldots A'_{i,n}$ must rewrite to x_i, and must do so by head reduction. The argument used in the previous lemma can be applied here. We take the longest common initial segment of the head reductions of these lambda expressions. Let t_i be the resulting reduct of $C' A'_{i,1} \ldots A'_{i,n}$. If for some i and j, a residual of $A'_{i,j}$ lies on the left spine of t_i, then this must be so for that j and all i. But for some i, $A_{i,j} = x_i$, so for that i, t_i must be x_i, otherwise it cannot reduce to x_i. Therefore for all other i', t'_i is a residual of $A_{i',j}$, and therefore cannot reduce to $x_{i'}$ unless $A_{i',j} = x_{i'}$. But this contradicts the condition that for some i', $A_{i',j} \neq x_{i'}$. □

Lemma 5.2 immediately yields the untranslatability of several TRSs.

Corollary 5.3 *None of the following rule-sets has any solution in the lambda calculus.*

1. *A variant of Berry's F:*

$$\begin{aligned} F\,x\,A\,B &\rightarrow x \\ F\,B\,x\,A &\rightarrow x \\ F\,A\,B\,x &\rightarrow x \end{aligned}$$

2. *Disjunction:*

$$\begin{aligned} or\,F\,x &\rightarrow x \\ or\,x\,F &\rightarrow x \end{aligned}$$

3. A Mal'cev operator:

$$\begin{aligned} m\,x\,x\,y &\to y \\ m\,x\,y\,y &\to x \end{aligned}$$

$\square$

Note that in the last example, the non-left-linearity of the rules is not the problem, since if the right-hand sides were x and y respectively, there would be a solution: $m = \lambda abc.b$. Also, sequentiality does not seem relevant to these examples. The first two examples of Corollary 5.3 are sequential, while the following non-sequential definition of parallel-or is translatable (but not properly translatable):

$$\begin{aligned} por\,T\,x &\to T \\ por\,x\,T &\to T \\ por\,F\,F &\to F \end{aligned}$$

One translation maps por to $\lambda xyz.z$ and T and F to $\lambda x.x$.

A small change to the first example of Corollary 5.3 makes it translatable: replace the right-hand side of the first rule by $C(x)$. It can then be translated by mapping F to $\lambda xyz.xyz$, A to $A' = \lambda xy.y$, B to $B' = \lambda xy.x$, and C to $C' = \lambda x.xA'B'$. This is not a proper translation, since we have $[\![C(A)]\!] = [\![B]\!]$ and $[\![C(B)]\!] = [\![A]\!]$. These equalities must hold for any translation of this system. This can be proved by arguments similar to those in the previous non-translatability results. The second and third equalities for F imply that $[\![F(x,y,z)]\!]$ must have a head normal form of the form $x\overline{C}[x,y,z]$, where $\overline{C}$ is a list of 3-ary contexts. Those equalities then state that $B'\overline{C}[B',x,A'] = A'$ and $A'\overline{C}[A',B',x] = B'$. The first equality states that $x\overline{C}[x,A',B'] = C'x$ and hence that $B'\overline{C}[B,A',B'] = C'B'$. But $B'\overline{C}[B,A',B'] = A'[x := A'] = A'$, therefore $C'B' = A'$. $C'A' = B'$ is proved similarly.

This also provides an example of a system which is translatable but not uniformly translatable, since example 1 of Corollary 5.3 has the same left-hand sides but has no translation.

If we replace the first right-hand side by C, then the system does not even have a translation.. As before, one shows that the second and third equations imply that $[\![F(x,y,z)]\!]$ has a head normal form $x\overline{C}[x,y,z]$. Hence

$[\![F(x,A,B)]\!] = x\overline{C}[x,A',B']$. But x is not free in $[\![C]\!]$, and so translation cannot preserve the first equality.

Translatability is a much stronger property of a rule-set than consistency. It is possible to add to the lambda calculus a new function symbol F and the rules for Berry's F, and the resulting system will be consistent (as demonstrated by the fact that it has a model, see [Bar84]). Parallel-or is also consistent with lambda calculus. The Mal'cev operator is not: adding such an operator allows proving $x = y$ (by a non-trivial argument).

6 A uniform translation of strongly separable systems

We now demonstrate that every strongly separable orthogonal TRS has a uniform proper translation into lambda calculus.

6.1 Flattening

To define our translation, we will first transform strongly separable orthogonal TRSs into simpler term rewrite systems by a process we shall call *flattening*. This is a transformation of a certain class of TRSs (which we shall see is exactly the strongly separable TRSs) which eliminates occurrences of operators in left-hand sides except at the roots, and also reduces the size of left-hand sides.

Flattening changes the semantics (informally speaking) of the system in one significant way: it imposes a particular reduction strategy, in effect by nominating a favoured extension site in every redex-prefix that has more than one. This eliminates a certain amount of non-determinism from the system. Only a very restrictive class of systems have translations to lambda calculus which do not eliminate non-determinism: in effect, the systems which do not have any to begin with. See [Ken94] for a discussion of such systems.

Assume we are given an orthogonal TRS. Choose, if possible, an operator symbol F, an integer i, and a constructor symbol C, such that

1. i is an extension site of $F(\overline{w})$.
2. $F(\overline{x}, C(\overline{y}), \overline{z})$ (where the subterm $C(\overline{y})$ is the ith argument) is a redex-component but not a redex.

Add a new symbol F_C and a new rule:

$$F(\overline{x}, C(\overline{y}), \overline{z}) \quad \rightarrow \quad F_C(\overline{x}, \overline{y}, \overline{z})$$

This rule is called the *introduction* rule for F_C. In addition, replace every occurrence in every left-hand side of a redex-component of the form $F(\overline{x}, C(\overline{y}), \overline{z})$ by $F_C(\overline{x}\,\overline{y}\,\overline{z})$. The resulting rules are the *elimination rules* for F_C. This constitutes one *flattening step.*

The flattening transformation consists of repeating flattening steps as long as possible.

THEOREM 6.1 *Flattening terminates.*

PROOF. Consider the multiset of the sizes of the left-hand sides of a given orthogonal TRS, measuring size as the number of occurrences of function symbols.

Each flattening step adds a new rule whose left-hand side contains 2 function symbols. The flattening step also reduces the size of at least one of the existing left-hand sides by 1 symbol, and any such left-hand side must contain at least 3 function symbols. Therefore a flattening step replaces some values $n \geq 3$ by $n-1$, and adds a 2. This transforms the multiset into one which is smaller in the standard well-ordering of multisets of natural numbers. Therefore it can be repeated only finitely often. □

THEOREM 6.2 *Flattening preserves and reflects orthogonality.*

PROOF. It is sufficient to prove this for a single flattening step. Let F and C be the function symbols involved, with the extension side of $F(\overline{x})$ being i.

The transformed system is clearly left-linear. Suppose there were a conflict between two transformed left-hand sides t_1 and t_2. By replacing every occurrence in these terms of a subterm $F_C(\overline{t}_1\overline{t}_2\overline{t}_3)$ by $F(\overline{t}_1, C(\overline{t}_2), \overline{t}_3)$, we obtain a conflicting pair of left-hand sides of the original system. This proves that flattening preserves orthogonality.

For reflection, suppose there is an overlap between two left-hand sides t_1 and t_2 of the original system. Flattening will modify these left-hand sides by replacing $F(\overline{t}_1, C(\overline{t}_2), \overline{t}_3)$ by $F_C(\overline{t}_1\overline{t}_2\overline{t}_3)$ throughout them. For this to remove the given conflict, that conflict can only be between $F(\overline{t}_1, C(\overline{t}_2), \overline{t}_3)$ and

$F(\overline{t'}_1, x, \overline{t'}_3)$, where these are unifiable terms, one of them is a left-hand side, the other is a subterm of a left-hand side, and the x is the ith argument. But no left-hand side can contain the latter as a subterm, since i is an extension site of $F(\overline{x})$. $\square$

THEOREM 6.3 *Flattening preserves and reflects strong separability.*

PROOF. It is sufficient to prove this for a single flattening step. Let a trs R be transformed to R' by a flattening step applied to F, i, and C.

Assume R is strongly separable. Let t' be a proper prefix of a left-hand side l' of R'. If l' is the left-hand side of the introduction rule for F_C, then t' must be $F(\overline{x})$, and i is an extension site of t' in R'. Otherwise, l' arises from some left-hand side l of R by substitution of $F_C(\overline{t}_1\overline{t}_2\overline{t}_3)$ for $F(\overline{t}_1, C(\overline{t}_2), \overline{t}_3)$. Applying the reverse substitution to t' gives a proper prefix t of l. t must have an extension site u in R. The variable at u in t occurs at some position u' in t'; u' is an extension site of t' in R'. Therefore R' is strongly separable.

Assume R' is strongly separable. Let t be a proper prefix of a left-hand side l of R. Replace every occurrence of a subterm $F(\overline{t}_1, C(\overline{t}_2), \overline{t}_3)$ of t $F_C(\overline{t}_1\overline{t}_2\overline{t}_3)$, giving a term t'. If t' is a proper prefix of a left-hand side of R', then in a similar manner to the previous argument, every extension site of t in R gives rise to an extension site of t' in R'. t' may fail to be such a prefix, however, if t contains a subterm of the form $F(\overline{x})$, at some position u. But in this case, since i is an extension site of $F(\overline{x})$, $u \cdot i$ must be an extension site of t. Therefore R is strongly separable. $\square$

DEFINITION 6.4 A left-linear TRS is *flat* if every left-hand side has the form $F(\overline{x}, C(\overline{y}), \overline{z})$ or $F(\overline{x})$; equivalently, if every left-hand side has size at most 2.

The flat orthogonal systems are similar to Berarducci and Böhm's *canonical* systems [BB92], except that we do not require the constructor C to appear as the first argument of F. For flat systems, the test for orthogonality is very simple: each operator symbol must either have a single rule with left-hand side $F(\overline{x})$, or a set of rules with left-hand sides of the form $F(\overline{x}, C(\overline{y}), \overline{z})$ in which the constructors are all different, and all occur at the same position.

THEOREM 6.5 *Every flat orthogonal TRS is strongly separable.*

PROOF. Every proper redex-prefix has the form $F(\overline{w})$, and orthogonality implies that every left-hand side of the form $F(\overline{x}, C(\overline{y}), \overline{z})$ must have the constructor occurring as the same argument of F. This argument position is an extension site of $F(\overline{w})$. □

THEOREM 6.6 *Let R be an OTRS. R is strongly separable if and only if the result of flattening R is flat.*

The result of flattening a strongly separable orthogonal TRS is flat.

PROOF. If: By Theorem 6.5, flat OTRSs are strongly separable, so by Theorem 6.3, R is strongly separable.

Only if: Suppose that R is strongly separable and some left-hand side has size 3 or more. If R is not a constructor system, choose an operator symbol F such that there is a proper subterm of some left-hand side of the form $F(\overline{t})$ where the terms $\overline{t}$ contain no operator symbols. (The finiteness of left-hand sides implies that this must be possible.) Then $F(\overline{x})$ has an extension site, and a flattening step can be performed.

If R is a constructor system and not flat, chose any left-hand side $F(\overline{t})$ of size 3 or more. A flattening step can be applied to $F(\overline{x})$.

Therefore when flattening terminates, every left-hand side must have size at most 2. □

To illustrate this theorem, we flatten the examples that were presented following Definition 3.1. Consider the following TRS: $H(G(A, A, x), A) \to t_0$, $H(G(A, x, A), B) \to t_1$, $G(B, B, B) \to t_2$. The redex-prefix $G(x, y, z)$ has one extension site, at the position 1. $G(A, x, y)$ is a redex-component which is not a redex, and A is a constructor. Thus the conditions for flattening hold. We replace the first two rules by $H(G_A(A, x)) \to t_0$ and $H(G_A(x, A)) \to t_1$, and add a rule $G(A, x, y) \to G_A(x, y)$. We can perform a second flattening step based on the redex-component $G(B, x, y)$. The original rule for G is replaced by $G_B(B, B) \to t_2$, and we add a rule $G(B, x, y) \to G_B(x, y)$. Next, we consider the redex-prefix $H(x, y)$. Both 1 and 2 are extension sites. Taking 1, in two flattening steps we replace the rules for H by $H(G_A(x, y)) \to H_{G_A}(x, y)$, $H(G_B(x, y)) \to H_{G_B}(x, y)$, $H_{G_A}(A, x) \to t_0$, and $H_{G_A}(x, A) \to t_1$. The only remaining non-flat left-hand side is $G_B(B, B)$. $G_B(x, y)$ has extension sites at 1 and 2. If we choose position 2, we replace this rule by

$G_B(x, B) \to G_{B_B}(x)$ and $G_{B_B}(B) \to t_2$. All of the left-hand sides are now flat.

Now consider the similar system $H(G(A, x), A) \to t_0$, $H(G(x, A), B) \to t_1$, $G(B, B) \to t_2$. The only possible flattening step takes the extension site 2 of $H(x, y)$. The first two rules are replaced by $H(x, A) \to H_A$, $H(x, B) \to H_B$, $H_A(G(A, x)) \to t_0$, and $H_B(G(x, A)) \to t_1$. There are no remaining possibilities for flattening. $G(x, y)$ has no extension site. $H_A(x)$ and $H_B(x)$ have extension sites at position 1, but the symbol there is not a constructor. However, the system is not flat.

We now show how the reduction relations of a strongly separable OTRS and its flattening are related.

THEOREM 6.7 *Let R be strongly separable and let R' be a flattening of R. Let t and t' be terms of R. Then (1) $t \to_R t' \Rightarrow t \to^+_{R'} t'$, and (2) $t \to^*_R t' \iff t \to^*_{R'} t'$.*

PROOF. It is sufficient to prove these when R' results from R by a single flattening step for symbols F and C and position i.

Suppose that $t \to_R t'$. If this is by a rule of R', then $t \to'_R t'$. Otherwise, it is by a rule $l \to r$ which flattening transforms to an elimination rule of R'. The same reduction can be performed in R' by first applying the introduction rule for F_C as often as $F(\ldots, C(\ldots), \ldots)$ occurs in l, then applying the elimination rule derived from $l \to r$. This establishes (1), and hence also the forwards implication of (2).

For the reverse direction of (2), consider a single step $t \to_{R'} t'$, where t and t' are any terms of R'. Let s and s' be the normal forms of t and t' respectively by the reverse of the F_C-introduction rule. That is, they are obtained by replacing every occurrence of $F_C(\ldots)$ by $F(\ldots, C(\ldots), \ldots)$. If $t \to_I t'$, then $s = s'$. If $t \to_E t'$ by a rule r, then $s \to_R s'$ by the rule which the flattening step replaced by r. If $t \to_R t'$ by a rule r, then $s \to_R s'$ by the same rule, since F_C cannot occur in the left-hand side of r. So in all cases, $s \to^*_R s'$. It follows that if $t \to^*_{R'} t'$, then $s \to^*_R s'$, where s and s' are defined as before. But if t and t' are in R, then $t = s$ and $t' = s'$. $\square$

6.2 A uniform proper translation of flat OTRSs

The translation we shall present is in effect an algorithm for compiling strongly separable OTRSs into lambda calculus. We begin by giving lambda calculus definitions of some simple building blocks: tupling, numerals, etc.

DEFINITION 6.8

$$
\begin{array}{rcl}
\langle t, t' \rangle & = & \lambda z.ztt' \\
[\overline{t}] \;=\; [t_1, \ldots, t_n] & = & \langle t_1, \langle t_2 \ldots, \langle t_{n-1}, \langle t_n, \mathbf{I} \rangle \rangle \rangle \rangle \\
\mathbf{T} & = & \lambda xy.x \\
\mathbf{F} & = & \lambda xy.y \\
\mathbf{left} & = & \lambda x.x\mathbf{T} \\
\mathbf{right} & = & \lambda x.x\mathbf{F} \\
\underline{0} & = & \mathbf{I} \\
\underline{n+1} & = & \langle \mathbf{F}, \underline{n} \rangle \\
\mathbf{Succ} & = & \lambda x.\langle \mathbf{F}, x \rangle \\
\mathbf{Pred} & = & \mathbf{right} \\
\mathbf{Zero} & = & \mathbf{left} \\
\mathbf{if} & = & \lambda xyz.xyz \\
\mathbf{Y} & = & \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) \\
\mathbf{Eq} & = & \mathbf{Y}\,(\lambda exy.\mathbf{if}\ (\mathbf{Zero}\ x)(\mathbf{Zero}\ y)(e(\mathbf{Pred}\ x)(\mathbf{Pred}\ y))) \\
x^i y & = & x(x(x \ldots (xy) \ldots)) \qquad (i \text{ occurrences of } x)
\end{array}
$$

For a lambda expression t and a non-negative integer i:

$$t \downarrow i \;=\; \mathbf{left}\,(t\,\mathbf{F}\ \ldots\ \mathbf{F}) \qquad i \text{ occurrences of } \mathbf{F}$$

Instead of **if** $t\ u\ v$ we may write **if** t **then** u **else** v for clarity.

case t **of** $t_1 : s_1; \ldots ; t_n : s_n$ **otherwise** s **end** means **if** $(\mathbf{Eq}\ t\ t_1)\ s_1\ (\ldots$ $(\mathbf{if}\ (\mathbf{Eq}\,t\,t_n)\ s_n\ s) \ldots)$.

The details of these definitions are less important than the properties they satisfy. $\langle t, t' \rangle$ is a pairing construction, and **left** and **right** select the components. $[\overline{t}]$ uses pairing to build lists of any length terminated by **I**. For integers n the expressions $\underline{n}$ behave as numerals with successor **Succ** and predecessor **Pred**. **Zero** is a test for equality with zero, and **Eq** is equality of arbitrary pairs of numerals. **if-then-else** is a conditional test, and **case-of-otherwise-end** is a case switch.

Let there be given some flat OTRS. Consider the set whose members are the constructor symbols of the system and the pairs (F, C) for which there is a rule whose left-hand side has the form $F(\overline{x}_1, C(\overline{x}_2), \overline{x}_3)$. Let this set be enumerated in some order, and let $ord(C)$ and $ord(F, C)$ be the positions of C and (F, C) in this sequence.

Each defined symbol F has either a single rule of the form

$$F(\overline{x}) \quad = \quad r$$

or a set of one or more rules of the form

$$F(\overline{x}, C_i(\overline{y}_i), \overline{z}) \quad = \quad r_i$$

for $i : 1 \ldots n$. (By renaming of variables we can assume without loss of generality that the variables $\overline{x}$ and $\overline{z}$ in the latter case are the same in each of the rules for F.)

In the first case, let the rule be the ith rule of the system. We define:

$$[\![F]\!]_u \;=\; \lambda \overline{a}\, \overline{x}. a_i\, \overline{x}$$

In the second case, let the rules be rules $i_1, \ldots, i_n$. Let $\overline{a}$ be a tuple of n variables. We define:

$$\begin{array}{lll} [\![F]\!]_u & = & \lambda \overline{a}\, \overline{x}\, y\, \overline{z}. \\ & & \quad \textbf{case left}(y)\ \textbf{of} \\ & & \qquad \underline{ord(C_1)} :\ a_1 \overline{x}(y \downarrow 1) \ldots (y \downarrow arity(C_1))\overline{z} \\ & & \qquad \quad \ldots \\ & & \qquad \underline{ord(C_n)} :\ a_n \overline{x}(y \downarrow 1) \ldots (y \downarrow arity(C_n))\overline{z} \\ & & \qquad \textbf{otherwise}\ :\ [\underline{ord(F, C)}, \overline{x}\,(y \downarrow 1) \ldots (y \downarrow arity(C_1))\overline{z}] \\ & & \quad \textbf{end} \end{array}$$

For each constructor symbol F, define

$$[\![F]\!]_u \;=\; \lambda \overline{a}\, \overline{x}. [\underline{ord(F)}, \overline{x}]$$

where the length of $\overline{x}$ is the arity of F.

This will be the constant part of a uniform translation. To construct the variable part for a given system R, we take equation 2.3(3):

$$[\![F]\!]_R \;=\; [\![F]\!]_u(\lambda\overline{x}_1.[\![r_1]\!]_R)\ldots(\lambda\overline{x}_n.[\![r_n]\!]_R)$$

and use equations 2.2(1), 2.2(2), and 2.3(3) to replace the notations $[\![r_i]\!]_R$ by lambda expressions in which notations $[\![G]\!]_R$ may occur for the function symbols G of R. This can then be read as a set of equations specifying $[\![F]\!]_R$ for all F:

$$[\![F_i]\!]_R \;=\; C_i[[\![F_1]\!]_R,\ldots,[\![F_n]\!]_R]$$

where C_i is a lambda context. If there are circular dependencies, the equations can be solved by means of a standard fixed point construction.

6.3 Correctness of the translation

THEOREM 6.9 *The above is a uniform proper translation of flat OTRSs.*

PROOF. The definition of $[\![F]\!]_R$ in terms of $[\![F]\!]_u$ makes equation 2.3(3) immediate.

A simple calculation establishes equation 2.3(4). For a left-hand side of the form $l_i = F(\overline{x})$, we have:

$$\begin{aligned}[\![l_i]\!]_u\overline{a} &= [\![F(\overline{x})]\!]_u\overline{a}\\ &= [\![F]\!]_u\overline{a}([\![x_1]\!]_u\overline{a})\ldots([\![x_k]\!]_u\overline{a})\\ &= a_i\overline{x}\end{aligned}$$

For a left-hand side of the form $l_i = F(\overline{x}, C(\overline{y}), \overline{z})$, we have:

$$\begin{aligned}[\![l_i]\!]_u\overline{a} &= [\![F]\!]_u\overline{a}\,\overline{X}([\![C]\!]_u\overline{a}\,\overline{Y})\overline{Z}\\ &= a_i\,\overline{X}\,\overline{Y'}\,\overline{Z}\\ &= a_i\,\overline{x}\,\overline{y}\,\overline{z}\end{aligned}$$

where

$$X_i \;=_{df}\; [\![x_i]\!]\overline{a}$$

$$
\begin{aligned}
&= x_i \\
Y_i &=_{df} [\![y_i]\!]\overline{a} \\
&= y_i \\
Y'_i &=_{df} [\underline{ord(C)}, \overline{y}] \downarrow i \\
&= y_i \\
Z_i &=_{df} [\![z_i]\!]\overline{a} \\
&= z_i
\end{aligned}
$$

Therefore this is a uniform translation.

To prove properness, note that the constructor prefixes are just the terms of the form $C(\overline{x})$ for constructor symbols C. Such a term is translated to $[\underline{ord(C)}, \overline{x}]$, from which x_i is exposed by the context **left**(**right**^{i}[ ]). A separating context for any tuple of these terms is easily constructed. □

THEOREM 6.10 *Let R be a strongly separable OTRS. Let R_F be its flattening. The translation of R_F, restricted to the function symbols of R, is a uniform proper translation of R.*

PROOF. Every equality of R is an equality of R_F, and hence translates to an equality of lambda calculus. Uniformity is immediate from the fact that the flattening transformation is independent of right-hand sides.

To prove properness, let t be a constructor prefix of R. Then in R_F, t either is or reduces to a term $C(\overline{x})$, where in the latter case C is one of the new symbols introduced in R_F, and $\overline{x}$ is the tuple of free variables of t from left to right. A context for exposing x_i in $[\![C(\overline{x})]\!]_{R_F}$ will also expose x_i in $[\![t]\!]_R$.

The flattening transformation maps each constructor schema of R to a term of R_F consisting entirely of constructors and variables. Distinct constructor schemas of R are mapped to terms of R_F which are not unifiable with each other (otherwise R would not be orthogonal). Since each constructor of R_F is mapped to an operator that tuples its arguments together with a numeral identifying the constructor, it is routine to construct a separating context for any tuple of constructor terms of R_F which contains no unifiable pairs. □

Finally, we demonstrate that the result of a computation in the lambda calculus can be translated back to a result in the original TRS. As the correspondence between a forward branching TRS and its flattening is so direct, we shall prove only the relationship between the end result of a computation in a flat OTRS and the result of its lambda calculus translation.

THEOREM 6.11 *Let t be a term in a flat OTRS R which has a normal form t' consisting only of constructors. Then for the translation which we gave in section 6.2, $[\![t]\!]_R$ has a normal form s from which t' can be computed.*

Let t be a term in a flat OTRS R which reduces to a term of the form $C[t_1, \ldots, t_n]$, where the prefix $C[\,]$ consists only of constructors. Then from $[\![t]\!]$ we can compute $C[\,]$, and lambda terms equal to $[\![t_1]\!], \ldots, [\![t_n]\!]$.

PROOF. Since the translation preserves equality, $[\![t]\!]_R = [\![t']\!]_R$ in lambda calculus. Therefore without loss of generality, we can assume that in the first part, t is already a constructor term, and in the second part, t is already in the form $C[t_1, \ldots, t_n]$.

Thus for the first part, we must prove that for any constructor term t, $[\![t]\!]_R$ has a normal form from which t may be computed.

Let $t = C(\overline{t})$. Then

$$[\![t]\!]_R = [\underline{ord(C)}, \overline{T}]$$

where

$$T_i = [\![t_i]\!]_R$$

But we then have

$$\begin{aligned} [\![t]\!]_R \downarrow 0 &= \underline{ord(C)} \\ [\![t]\!]_R \downarrow i &= \overline{T_i} \end{aligned}$$

Thus the index of the constructor of t can be recovered from $[\![t]\!]_R$, as can the translations of the immediate subterms of t. By induction, all of t can be computed from $[\![t]\!]_R$.

For the second part, by induction it is sufficient to consider the case where the context $C[\,]$ is a single constructor C, and $t = C(t_1, \ldots, t_n)$. Then as in the first part we have

$$\begin{aligned} [\![t]\!]_R \downarrow 0 &= \underline{ord(C)} \\ [\![t]\!]_R \downarrow i &= [\![t_i]\!]_R \end{aligned}$$

□

6.4 Complexity of the translation

It is routine to verify that flattening results in only a linear growth of the size of the system. Given a flat system, the uniform part of the translation of an operator symbol has size proportional to the total number of free variables occurring in the left-hand sides of the rules for that symbol — that is, proportional to the total size of those left-hand sides. The uniform part of the translation of a constructor has size proprtional to its arity.

The variable part $[\![F]\!]_R$ can be larger by a factor proportional to the number of variables in a left-hand size for F. However, some optimisation can be performed, as equation 2.3(3) allows a large number of trivial beta-reductions to be performed. If the right hand side of the equation is reduced to normal form (without expanding any of the symbols $[\![G]\!]_R$), then the size of the resulting specification of $[\![F]\!]_R$ becomes proportional to the total size of all the rules for $[\![F]\!]_R$. The subsequent fixed-point extraction adds only a constant amount.

It is more difficult to obtain a meaningful comparison of the cost of reductions in the original system and in the transformed system. Any such comparison would have to consider such things as the implementation of the **case** operation (we have defined it as a linear search, but other definitions are possible), and the mechanism for pattern-matching in an implementation of the original system. This gets into details of implementation which cannot be realistically be studied in this abstract setting.

6.5 An example of the translation

We take the rules:

$$\begin{array}{rcl} F(A,B) & \to & t_0 \\ G(F(x,C(y,z))) & \to & t_1 \end{array}$$

A flattening of these rules is:

$$F(x,B) \;\to\; F_B(x)$$

$$
\begin{array}{rcl}
F(x, C(y, z)) & \rightarrow & F_C(x, y, z) \\
F_B(A) & \rightarrow & t_0 \\
G(F_C(x, y, z)) & \rightarrow & t_1
\end{array}
$$

This results in the following definitions of the constant part of a uniform translation.

$$
\begin{array}{rll}
[\![F]\!]_u = & \lambda a_1 a_2 a_3 a_4 x y. & \textbf{case } (\textbf{left } y) \textbf{ of} \\
 & & \quad \underline{ord(B)} : a_1 x \\
 & & \quad \underline{ord(C)} : a_2 x (y \downarrow 1)(y \downarrow 2) \\
 & & \quad \textbf{otherwise} \ : \langle \underline{ord(F)}, x \rangle \\
 & & \textbf{end}
\end{array}
$$

$$
\begin{array}{rll}
[\![F_B]\!]_u = & \lambda a_1 a_2 a_3 a_4 x. & \textbf{case } (\textbf{left } x) \textbf{of} \\
 & & \quad \underline{ord(A)} : a_3 \\
 & & \quad \textbf{otherwise} \ : \langle \underline{ord(F_B)}, x \rangle \\
 & & \textbf{end}
\end{array}
$$

$$
\begin{array}{rll}
[\![G]\!]_u = & \lambda a_1 a_2 a_3 a_4 x. & \textbf{case } (\textbf{left } x) \textbf{ of} \\
 & & \quad \underline{ord(F_C)} : a_3 (x \downarrow 1)(x \downarrow 2)(x \downarrow 3) \\
 & & \quad \textbf{otherwise} \ : \langle \underline{ord(G)}, x \rangle \\
 & & \textbf{end}
\end{array}
$$

$$
\begin{array}{rcl}
[\![F_C]\!]_u & = & \lambda a_1 a_2 a_3 a_4 x y z.[\underline{ord(F_C)}, x, y, z] \\
[\![A]\!]_u & = & \lambda a_1 a_2 a_3 a_4.[\underline{ord(A)}] \\
[\![B]\!]_u & = & \lambda a_1 a_2 a_3 a_4.[\underline{ord(B)}] \\
[\![C]\!]_u & = & \lambda a_1 a_2 a_3 a_4 x y.[\underline{ord(C)}, x, y]
\end{array}
$$

The translations of the symbols for the particular rewrite system can be computed from Definition 2.7. As an example, we write out $[\![F]\!]_R$, simplifying by beta reduction where possible:

$$
\begin{array}{rl}
[\![F]\!]_R = & \lambda xy.\ \textbf{case}\ (\textbf{left}\ y)\ \textbf{of} \\
& \qquad \underline{ord(B)} : \textbf{case}\ (\textbf{left}\ x)\textbf{of} \\
& \qquad\qquad \underline{ord(A)} : [\![t_0]\!]_R \\
& \qquad\qquad \textbf{otherwise} : \langle \underline{ord(F_B)}, x \rangle \\
& \qquad\quad \textbf{end} \\
& \qquad \underline{ord(C)} : [\underline{ord(F_C)}, x(y \downarrow 1)(y \downarrow 2)] \\
& \qquad \textbf{otherwise} : \langle \underline{ord(F)}, x \rangle \\
& \textbf{end}
\end{array}
$$

7 Acknowledgments

We wish to thank Mariangiola Dezani-Ciancaglini for useful discussions and for drawing some of the references to our attention.

References

[AG98] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages.* Cambridge University Press, 1998. Cambridge Tracts in Theoretical Computer Science, No. 45.

[Bar84] H.P. Barendregt. *The Lambda Calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* Elsevier Science Publishers B.V., P.O. Box 1991, 1000 BZ Amsterdam, The Netherlands, 2nd edition, 1984.

[BB92] A. Berarducci and C. Böhm. A self-interpreter of lambda calculus having a normal form. In *Computer Science Logic'92*, pages 85–99. Springer, October 1992.

[BDC74] C. Böhm and M. Dezani-Ciancaglini. Combinatorial problems, combinator equations and normal forms. In *2nd International Colloquium on Automata, Languages, and Programming*, pages 189–199, 1974. LNCS 14.

[BKS95] S. Byun, J.R. Kennaway, and M.R. Sleep. Transformation of orthogonal term rewriting systems. In *1995 Asian Computing Science Conference*, pages 73–87. Springer-Verlag, 1995. LNCS 1023.

[BKS96] S. Byun, J.R. Kennaway, and M.R. Sleep. Lambda-definable term rewriting systems. In *Proc. 1996 Asian Computing Science Conference*, pages 106–115. Springer-Verlag, 1996. LNCS 1179.

[Dur94] I. Durand. Bounded, strongly sequential, and forward branching term rewrite systems. *J. Symb. Comp.*, 18:319–352, 1994.

[HL91] G. Huet and J.-J. Lévy. Computations in orthogonal rewrite systems I and II. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson*, pages 395–444. MIT Press, 1991.

[HLM98] M. Hanus, S. Lucas, and A. Middeldorp. Strongly sequential and inductively sequential term rewriting systems. *Information Processing Letters*, 67(1):1–8, 1998.

[Ken94] J.R. Kennaway. A conflict between call-by-need computation and parallelism. In N. Dershowitz and N. Lindenstrauss, editors, *Proc. Workshop on Conditional and Typed Term Rewriting Systems (CTRS '94)*, pages 247–261, Jerusalem, 1994. Springer. LNCS 968.

[Mid90] A. Middeldorp. *Modular properties of term rewriting systems.* PhD thesis, Vrije Universiteit, Amsterdam, 1990.

[Oho99] A. Ohori. The logical abstract machine: A Curry-howard isomorphism for machine code. In *4th Fuji International Symposium on Functional and Logic Programming (FLOPS'99)*, pages 300–318. Springer, November 1999.

[SS97] B. Salinier and R. Strandh. Simulating forward-branching systems with constructor systems. In *Proc. CAAP'97.* Springer-Verlag, 1997. Lecture Notes in Computer Science.

[Str89] R. Strandh. Classes of equational programs that compile into efficient machine code. In *Proc. 3rd Int. Conf. on Rewriting Techniques and Applications*, pages 449–461. Springer-Verlag, 1989. Lecture Notes in Computer Science 355.