Term Rewriting with Sequences

Makoto Hamana
Doctoral Program in Engineering, University of Tsukuba,
Tsukuba 305, Japan
E-mail: hamana@score.is.tsukuba.ac.jp

Abstract

This paper gives an extension of term rewriting systems, which treat not only terms but also "sequences". We present a definition of confluent rewrite relation for this systems and matching algorithms used in rewriting. Application to rewriting semantics of Mathematica programming language is also discussed.

1 Introduction

We can regard term rewriting systems[Klo92] (TRSs for short) as rewrite systems on tree structure. Thus, to treat linear data structure (e.g. arrays) in TRSs, we must use recursive data structure (e.g. lists) for it. If we can treat such linear data structure directly, the expressive power of TRS will be more increased.

The idea of this paper to introduce sequences into TRSs comes from Mathematica[Wol97] programming language. Mathematica has linear data structure called "sequences" and two kinds of variables, usual variables and "sequence variables". The structure of "sequences" essentially differs from that of tuples in other functional languages, like ML or Haskell. The important property of sequences is *flatness*, thus we cannot construct "sequence of sequences", which is automatically flattened into one sequence. This property is contrast to the case of tuples, where we can define nested tuples.

To introduce tuples into TRSs is just done by defining constructors for them. But to formalize flatness of sequences in TRSs cannot be so directly done because sequences as abstract data structure must satisfy equational axioms expressing flatness and other properties of sequences.

It is well-known that the property of sequences can be equationally axiomized by monoid's axioms. Thus we formalize TRSs with sequences as TRSs modulo monoid axioms (Section 3). This is just a special case of equational TRSs which are TRSs modulo arbitrary set of equations[JK86]. But in the case of monoid, the induced rewrite relation by the theory of equational TRSs is not generally confluent. In order to use term rewriting with sequences as an operational semantics of functional programming languages, we try to obtain sufficient conditions of confluent rewrite relation. We restrict the rewrite relation by taking minimal M-matcher (M is a set of axioms of monoid) and innermost strategy (Section 4). Although, E-matching in equational rewriting is undecidable in general, it is decidable in the case of monoid. We present explicitly algorithms to obtain minimal M-matcher and prove its correctness (Section 5,6). Finally, we illustrate an application of this rewrite systems to rewriting semantics of Mathematica language (Section 7).

2 Language

We assume familiarity with the basic notions of order-sorted algebras [Wal92] and term rewriting[Klo92].

sort. The sort set S used in this paper is the set $\{\mathsf{Word}, \mathsf{Elm}\}$ with the order $\mathsf{Elm} \sqsubseteq \mathsf{Word}$.

signature. A S-sorted signature Σ is the $S^* \times S$-sorted family $\{\Sigma_{w,s} \mid w \in S^*, s \in S\}$ of signatures. The signature Σ used in this paper must satisfy $\Sigma = \mathcal{F} \cup \Sigma_{\epsilon,\mathsf{Word}} \cup \Sigma_{\mathsf{Word}\,\mathsf{Word},\mathsf{Word}}$ where

- $\mathcal{F} = \{\mathcal{F}_{\epsilon,s} \mid s \in S\} \cup \{\mathcal{F}_{w,s} \mid w \in S, s \in S\}$ for user-defined function symbols,
- $\Sigma_{\epsilon,\mathsf{Word}} = \{\mathsf{e}\}$ for the empty word,
- $\Sigma_{\mathsf{Word\ Word},\mathsf{Word}} = \{\mathsf{.}\}$ for the concatenation operator.

variables. A S-sorted family V of variables is the disjoint union of $\mathsf{V}_{\mathsf{Word}}$ and $\mathsf{V}_{\mathsf{Elm}}$. We write elements of $\mathsf{V}_{\mathsf{Elm}}$ as $x, y, \cdots$ and of Word as over-lined forms $\bar{x}, \bar{y}, \cdots$.
terms. A set $\mathsf{T}_\Sigma(\mathsf{V})$ of terms is that of well-formed order-sorted terms built from Σ and $\mathcal{V}$. We call an elements of $\mathsf{T}_\Sigma(\mathsf{V})_{\mathsf{Elm}}$ an *elements* and or $\mathsf{T}_\Sigma(\mathsf{V})_{\mathsf{Word}}$ *word*. Notice that we only allow 0 or 1-arities for user-defined function symbols in $\mathcal{F}$. So we use a term $f(t_1 \mathsf{.} \cdots \mathsf{.} t_n)$ instead of usual many arity term $f(t_1, \cdots, t_n)$. We write $t : s$ for a term t of the sort s. The length of a word w is denoted by $\#w$. The size of a term t is denoted by $|t|$. Notice that $|\mathsf{e}| = 1$. The set of variables occurring in a term t is denoted by $\mathcal{V}(t)$.
substitution. An *assignment* ϕ is a mapping from a S-sorted set of variables X to an S-sorted Σ-algebra $\mathcal{A}$ and often presented as $\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$, where $\sigma(x_i) = t_i$ for each $x_i \in X, i = 1, \cdots, n$. An assignment from X to $\mathsf{T}_\Sigma(\mathsf{V})$ of terms is called *substitution*. The assignment ϕ is uniquely extended to a homomorphism $\phi^\#$ from $\mathsf{T}_\Sigma(\mathsf{V})$ to $\mathcal{A}$.
equations, rewrite rules. An *equation* is denoted by $\forall x_1, \cdots, x_n(s = t)$ where s is an element, t is a term and $x_1, \cdots, x_n$ are variables occurring in the terms s and t. We will often omit the quantified variables if these are clear from context. A rewrite rule is denoted by $l \to r$ where l is an element and r is a term and satisfies the condition $\mathcal{V}(l) \supseteq \mathcal{V}(r)$.
STRSs. An order-sorted term rewriting system (TRS for short) is a triple (S, Σ, R) where R is a set of rewrite rules between terms in $\mathsf{T}_\Sigma(\mathsf{V})$. We have already noticed that the sort set S is fixed and a signature must satisfy the conditions described above. We call an order-sorted TRS on such sort and signature a STRS, which stands for TRS with sequences. A STRS R induces a order-sorted rewrite relation $\to_R$ on $\mathsf{T}_\Sigma(\mathsf{V})$ as follows[Wal92]:

- There exists a rewrite rule $l \to r \in R$.
- There exists a position $p \in \mathcal{P}os(t)$.
- There exists a substitution $\sigma : \mathsf{V} \to T_\Sigma(X)$ such that $l\sigma = t_{|_p}$.
- $t' = t[r\sigma]_p$.
- there exists a sort $s \in S$ such that $t[x : s]_p$ is a well-formed term and $l\sigma, r\sigma \in T_\Sigma(X)_s$.

3 Rewriting modulo monoid

We define a set I, A and M of equations for identity, associativity and monoid respectively.

$$\mathsf{I} = \left\{ \begin{array}{l} \forall(\bar{x} \mathsf{.} \mathsf{e} = \bar{x}) \\ \forall(\mathsf{e} \mathsf{.} \bar{x} = \bar{x}) \end{array} \right\}$$
$$\mathsf{A} = \left\{ \forall((\bar{x} \mathsf{.} \bar{y}) \mathsf{.} \bar{z} = \bar{x} \mathsf{.} (\bar{y} \mathsf{.} \bar{z})) \right\}$$
$$\mathsf{M} = \mathsf{I} \cup \mathsf{A}$$

The equivalence relation $=_\mathsf{I}$, $=_\mathsf{A}$ and $=_\mathsf{M}$ denotes induced equalities from I, A and M. By regarding I, A and M as left-to-right oriented rewrite rules, we can treat that I, A and M are STRSs. Thus the usual rewrite relation $\to_\mathsf{I}$, $\to_\mathsf{A}$ and $\to_\mathsf{M}$ are defined.

We define rewriting modulo monoid for STRSs.

Definition 3.1
Let R be a STRS. The rewrite relation $\to_{R/\mathsf{M}}$ is defined as follows:

$$\to_{R/\mathsf{M}} \stackrel{def}{=} =_{\mathsf{M}} \circ \to_R \circ =_{\mathsf{M}} .$$

The relation $\to_{R/\mathsf{M}}$ is an instance of the equational rewrite relation in equational term rewriting systems[JK86] where the set of equation is M.

Example 3.2
We illustrate that some uses of words (or we often say sequences) and the relation $\to_{R/\mathsf{M}}$ formalizes term rewriting with sequences. Consider a STRS

$$R = \left\{ \begin{array}{ll} \mathsf{first}(x \,.\, \bar{y}) & \to x \\ \mathsf{last}(\bar{x} \,.\, y) & \to y \\ \mathsf{reverse}(\mathsf{e}) & \to \mathsf{e} \\ \mathsf{reverse}(x \,.\, \bar{y}) & \to \mathsf{reverse}(\bar{y}) \,.\, x \\ \mathsf{member}(c \,.\, (\bar{x} \,.\, (c \,.\, \bar{y}))) & \to \mathsf{true} \end{array} \right\} .$$

The STRS R axiomatizes well-know functions appearing in modern functional languages like Haskell. However we notice that the functions in R are on *sequences*, not on *lists*. Lists in functional languages and sequences seem similar, but the difference is the structure. Namely,

- words are flat structure, but
- lists are recursive structure.

The effect of the flatness of sequences appears in R. The function first is the same form as the case of functional languages, but last can take the last element directly due to the flatness of sequences. In the second rule of the function $\mathsf{reverse}$, the appending x to the last of $\mathsf{reverse}(\bar{y})$ is simply done by concatenation operator "." in contrast to the use of "append" function in functional languages. The function member checks whether a given sequence contains the constant c of the first argument. In functional languages, this is defined by recursive checking, but the use of sequences enables to describe it more directly, namely, $\bar{x} \,.\, (c \,.\, \bar{y})$ expresses a sequence which contain the constant c.

The following are the examples of rewriting by $\to_{R/\mathsf{M}}$.

$\mathsf{first}((1 \,.\, 2) \,.\, 3) =_{\mathsf{M}} \mathsf{first}(1 \,.\, (2 \,.\, 3)) \to_R 1$
$\mathsf{last}(1 \,.\, ((2 \,.\, 3) \,.\, 4)) =_{\mathsf{M}} \mathsf{last}((1 \,.\, (2 \,.\, 3)) \,.\, 4) \to_R 4$
$\mathsf{reverse}(1 \,.\, (2 \,.\, 3)) \to_R \mathsf{reverse}(2 \,.\, 3) \,.\, 1 \to_R (\mathsf{reverse}(3) \,.\, 2) \,.\, 1 =_{\mathsf{M}} (\mathsf{reverse}(3 \,.\, \mathsf{e}) \,.\, 2) \,.\, 1$
$\to_R ((\mathsf{reverse}(\mathsf{e}) \,.\, 3) \,.\, 2) \,.\, 1 \to_R (\mathsf{e} \,.\, 3) \,.\, 2) \,.\, 1 =_{\mathsf{M}} (3 \,.\, 2) \,.\, 1$
$\mathsf{member}(c \,.\, (1 \,.\, (2 \,.\, (c \,.\, 3)))) \to_{R/\mathsf{M}} \mathsf{true}$

4 Confluent Rewriting

The relation $\to_{R/\mathsf{M}}$ is not confluent. The following example shows this. Consider a STRS

$$R = \{ \; f(\bar{x} \,.\, 2 \,.\, \bar{y}) \to \bar{x} \; \} .$$

There are two different derivations to normal forms starting from $f(1 \,.\, 2 \,.\, 1)$:

$$f(1 \,.\, 2 \,.\, 1) \to_{R/\mathsf{M}} 1, \tag{1}$$
$$f(1 \,.\, 2 \,.\, 1) \to_{R/\mathsf{M}} 1 \,.\, 2, \tag{2}$$

but $1 \neq_{\mathsf{M}} 1.2$.

In order to use term rewriting with sequences as an operational semantics of functional programming languages, we want to get confluent rewriting for it. Completion techniques in equational term rewriting systems [JK86][BD89] will make $1.2 \rightarrow_{R/\mathsf{M}} 1$ or reverse. But we do not take this approach because this kind of identification $1 =_{R \cup \mathsf{M}} 1.2$ is against our intuition of sequences, namely, we want to distinguish the sequences 1 and 1.2.

Our approach is to introduce a strategy for $\rightarrow_{R/\mathsf{M}}$ to make it confluent. The strategy introduced here discards the rewriting (2) in the above example, and only takes the rewriting (1). More precisely, the strategy chooses a rewriting which takes a *minimal* M*-matcher* for the matching part of rewrite steps. For example, the matchers in (1) and (2) are

$$\theta_{(1)} = \{\bar{x} \mapsto 1, \bar{y} \mapsto 2.1\} \text{ and } \theta_{(2)} = \{\bar{x} \mapsto 1.2, \bar{y} \mapsto \mathsf{e}\}.$$

The minimal M-matcher in this case is $\theta_{(1)}$ because the length of $\bar{x}\theta_{(1)}$ is smaller than that of $\theta_{(2)}$. The general definition of the minimal M-matcher is as follows:

Definition 4.1 [**Minimal M-matcher**]
A substitution σ is a *minimal* M*-substitution of a term* l if

$$\sigma = \mathsf{min}_{\leq_{\mathrm{m}}} \{\theta \mid \text{a substitution } \theta : \mathsf{V} \rightarrow \mathsf{T}_{\Sigma}(\mathsf{V}),\ l\sigma =_{\mathsf{M}} l\theta\},$$

where

- The ordering $\leq_{\mathrm{m}}$ on substitutions is defined as:

$$\alpha \leq_{\mathrm{m}} \beta \overset{def}{\Longleftrightarrow} |\bar{x_1}\alpha|.\cdots.|\bar{x_n}\alpha| \leq_{\mathsf{lex}} |\bar{x_1}\beta|.\cdots.|\bar{x_n}\beta|,$$

- $\bar{x_1}, \cdots, \bar{x_n}$ are variables of the sort Word occurred in l which are indexed corresponding to the order of the occurrences of them starting from the leftmost-innermost position, (i.e. $\bar{x_1}$ is the leftmost-innermost variable in l and $\bar{x_n}$ is the rightmost-innermost variable in l),

- $|w|$ is the length of a word w,

- $\leq_{\mathsf{lex}}$ is the lexicographic ordering on $\mathbb{N}^*$ which compares from left to right.

Note: σ is an I-normalized substitution. Moreover, we call σ a *minimal* M*-matcher of a term* l *for a term* t if

- $l\sigma =_{\mathsf{M}} t$,

- σ is a minimal M-substitution of l.

Notice that a minimal element in the set $\{\theta \mid \text{a substitution } \theta : \mathsf{V} \rightarrow \mathsf{T}_{\Sigma}(\mathsf{V}),\ l\sigma =_{\mathsf{M}} l\theta\}$ always exists because the order $\leq_{\mathsf{lex}}$ is a total order on $\mathbb{N}^*$. Hence we can immediate get the following proposition.

Proposition 4.2
If l has a M-matcher for t, then l has a minimal M-matcher for t.

Imposing minimal M-matcher restriction is not sufficient to make $\rightarrow_{R/\mathsf{M}}$ confluent. The following example illustrates this.

Example 4.3
Consider a STRS

$$R = \left\{ \begin{array}{ll} f(\bar{x} \,.\, 2 \,.\, \bar{y}) & \to \bar{x} \\ g & \to 2 \end{array} \right\}.$$

There are two possible derivations to normal forms starting from $f(1 \,.\, g \,.\, 2)$:

$$f(1 \,.\, g \,.\, 2) \to_{R/\mathsf{M}} f(1 \,.\, 2 \,.\, 2) \to 1, \tag{3}$$

$$f(1 \,.\, g \,.\, 2) \to_{R/\mathsf{M}} 1 \,.\, g \to 1 \,.\, 2, \tag{4}$$

but $1 \neq_{\mathsf{M}} 1 \,.\, 2$. This example shows if a term is rewritten when there are redexes in the inner part of it, then a separator (in this case "2") in the lhss of rewrite rules may not be correctly matched in minimal M-matching step.

Again, we impose a restriction on $\to_{R/\mathsf{M}}$ for confluent rewriting. We discard the rewriting (4) and take (3). Our intention is that parameters of a redex should be sufficiently rewritten to detect correct separators in the lhss of rewrite rules. To do so, we simply take a fixed *innermost* strategy. If a rewrite relation is not terminating, the innermost strategy may fail to reach a normal form and into infinite computation. Non-left linear and overlapping rules may also cause problems for confluent rewriting. So we assume that the rewrite relation is terminating and rules in STRSs are left-linear and non-overlapping. The formal definition of this strategy is as follows. First we define overlapping in STRSs.

Definition 4.4
A STRS R is *overlapping* if the following properties are satisfied:

- $l_1 \to r_1$ and $l_2 \to r_2$ are variants of different rewrite rules of R without common variables,
- non-variable position $p \in Pos(l_1)$,
- $l_{1|_p}$ and l_2 are M-unifiable, i.e., there exists a substitution σ such that $l_{1|_p}\sigma =_{\mathsf{M}} l_2\sigma$.
- if $p = \epsilon$ then $l_1 \to r_1$ and $l_1 \to r_1$ are not variants.

Definition 4.5
Let R be a left-linear and non-overlapping STRS. Choose leftmost, rightmost or parallel innermost strategy. In the case of leftmost or rightmost innermost strategy, the rewrite relation $\to_{R\mathsf{M}}$ is defined as follows: $t \to_{R\mathsf{M}} t'$ if

- There exists a rewrite rule $l \to r \in R$.
- There exists a leftmost (or rightmost) innermost redex position $p \in \mathcal{P}os(t)$.
- There exists a minimal M-matcher σ of l for $t_{|_p}$ (thus $l\sigma =_{\mathsf{M}} t_{|_p}$).
- $t' = t[r\sigma]_p$ (Notice: not $=_{\mathsf{M}}$).
- there exists a sort $s \in S$ such that $t[x : s]_p$ is a well-formed term and $l\sigma, r\sigma \in T_{\Sigma}(X)_s$.

In the case of parallel innermost strategy, $\to_{R\mathsf{M}}$ is similarly defined.

Proposition 4.6
$\to_{R\mathsf{M}}$ is confluent.
Proof By the definition of $\to_{R\mathsf{M}}$, for any term t there exists exactly one term t' such that $t \to_{R/\mathsf{M}} t'$. Hence $\to_{R\mathsf{M}}$ is trivially confluent.

Some restriction in $\rightarrow_{RM}$ may be weaken. We have not exactly checked that left-linearity is needed to ensure confluence. But this assumption makes consideration of matching algorithm easier. Actually, the matching algorithm presenting next two sections depends on left linearity. Moreover we may use more sophisticated strategy under non-terminating assumption instead of the fixed innermost strategy. But it will be difficult and future work.

5 Minimal M-matching algorithm

For decidable rewrite relation, we define matching algorithm modulo monoid M by Haskell-like functional program named Mmatch. The function Mmatch takes a lhs l of rule and a term s to be rewritten and returns a minimal M-matcher of l for s.

$$
\begin{aligned}
&\mathsf{Mmatch}(c,c) = \varnothing \\
&\mathsf{Mmatch}(c,d) = \varnothing \\
&\mathsf{Mmatch}(\mathsf{e},\mathsf{e}) = \varnothing \\
&\mathsf{Mmatch}(\mathsf{e},s) = \bot \\
&\mathsf{Mmatch}(x,s:\mathsf{Elm}) = \{x \mapsto s\} \\
&\mathsf{Mmatch}(\bar{x},s:\mathsf{Elm}) = \{\bar{x} \mapsto s\} \\
&\mathsf{Mmatch}(\bar{x},s:\mathsf{Word}) = \{\bar{x} \mapsto s\} \\
&\mathsf{Mmatch}(x,s:\mathsf{Word}) = \bot \\
&\mathsf{Mmatch}(f(t),f(s)) = \mathsf{Mmatch}(t,s) \qquad (5) \\
&\mathsf{Mmatch}(f(t),g(s)) = \bot \text{ if } f \neq g \\
&\mathsf{Mmatch}(t_1 \,.\, t_2, s:\mathsf{Elm}) = \mathsf{Mmatch}(t_1 \,.\, t_2, s \,.\, \mathsf{e}) \parallel \mathsf{Mmatch}(t_1 \,.\, t_2, \mathsf{e} \,.\, s) \qquad (6) \\
&\mathsf{Mmatch}(t_1 \,.\, t_2, s:\mathsf{Word}) = \|_{(s_1,s_2)\in S} \, (\mathsf{Mmatch}(t_1,s_1) \cup \mathsf{Mmatch}(t_2,s_2)) \qquad (7) \\
&\quad \text{where } S = \{(s_1,s_2) \mid s =_{\mathsf{A}} s_1 \,.\, s_2\} \text{ (Note: } \#S < \infty)
\end{aligned}
$$

Here, we assume that the union operator "$\cup$" on substitutions is strict for each parameter, i.e., $A \cup \bot = \bot \cup A = \bot$. The binary operator "$\parallel$" which computes M-matchers parallel way and returns minimal one is defined as follows.

$$
\begin{aligned}
\bot \parallel \bot &= \bot \\
\theta \parallel \bot &= \theta \\
\bot \parallel \theta &= \theta \\
\theta_1 \parallel \theta_2 &= \min_{\leq_{\mathrm{m}}}\{\theta_1,\theta_2\} \text{ if } (\bar{x_1} \,.\, \cdots \,.\, \bar{x_n})\theta_1 = (\bar{x_1} \,.\, \cdots \,.\, \bar{x_n})\theta_2
\end{aligned}
$$

where $\bar{x_1}, \cdots, \bar{x_n}$ are variables of the sort Word occurring in l which are indexed corresponding to the order of the occurrences of them starting from the leftmost-innermost position.

Theorem 5.1
Let l and s are I-normalized terms. Suppose l is linear. $s' \rightarrow_{\mathsf{I}}^{!} s$. Then $\mathsf{Mmatch}(l,s)$ returns a unique substitution in finite time and has following properties:
[soundness]

(i) $\mathsf{Mmatch}(l,s) \neq \bot \;\Rightarrow\; \mathsf{Mmatch}(l,s)$ is a minimal M-matcher of l for s.

(ii) $\mathsf{Mmatch}(l,s) = \bot \;\Rightarrow\; l$ is not M-matchable for s.

[completeness]

(i) l has M-matcher θ for $s \;\Rightarrow\; \mathsf{Mmatch}(l,s) = \theta$.

(ii) l is not M-matchable for $s \;\Rightarrow\; \mathsf{Mmatch}(l,s) = \bot$.

Proof See Appendix A.

6 Efficient minimal M-matching algorithm

In this section, we give another version of minimal M-matching algorithm. The algorithm Mmatch is clearly inefficient because it try to match for all possible concatenation of a word by associativity. To avoid such useless computation, first we assume that l and s in minimal M-matching problem are M-normalized words. Due to this assumption, we can use flat notation for M-normalized words[Sti81]. M-matching problem between flat words leads to more natural and direct linear-way checking. So it may become more efficient than Mmatch.

In the following the notation t denotes an element in $\mathsf{T}_\Sigma(\mathsf{V})_{\mathsf{Elm}}$ and $\bar{t}$ denotes an word $\mathsf{T}_\Sigma(\mathsf{V})_{\mathsf{Word}}$. Also, we use a flat notation $t_1 \,.\, t_2 \,.\, \cdots \,.\, t_{n-1} \,.\, t_n$ where $t_i : \mathsf{Elm} (1 \leq i \leq n)$ for M-normalized word $t_1 \,.\, (t_2 \,.\, \cdots \,.\, (t_{n-1} \,.\, t_n) \cdots)$. If $n = 0$, $t_1 \,.\, \cdots \,.\, t_n$ denotes e. We allow the over-lined notation in the flat notation, e.g., $t_1 \,.\, \bar{u} \,.\, t_2$ where $\bar{u} = u_1 \,.\, u_2 \,.\, u_3$, denotes A-normalized word $t_1 \,.\, (u_1 \,.\, (u_2 \,.\, (u_3 \,.\, t_2)))$.

Definition 6.1
An minimal M-matching algorithm MA consists of the following inference rules.

$$\frac{\bar{s_1} \,.\, c(\bar{s_1}) \,.\, \bar{s_3} \approx \bar{t_1} \,.\, c(\bar{t_2}) \,.\, \bar{t_3}, E}{\bar{s_1} \approx \bar{t_1}, \bar{s_2} \approx \bar{t_2}, \bar{s_3} \approx \bar{t_3}, E}$$

where $\bar{s_1}$ and $\bar{t_1}$ do not contain a term $c(s)$ (s is arbitrary term)
and $\bar{s_1}$ is a sequence of variables.

$$\frac{x \,.\, \bar{s} \approx t_1 \,.\, \bar{t_2}, E}{\bar{s} \approx \bar{t_2}, E} \quad x \mapsto t_1$$

$$\frac{\bar{x} \,.\, y \,.\, \bar{s} \approx t_1 \,.\, \cdots \,.\, t_n, E}{\bar{s} \approx t_{k+2} \,.\, \cdots \,.\, t_n, E} \quad \bar{x} \mapsto t_1 \,.\, \cdots \,.\, t_k,\ y \mapsto t_{k+1}$$

where $n \geq 1$ and try each k such that $0 \leq k \leq n-1$

$$\frac{\bar{x} \,.\, \bar{y} \,.\, \bar{s} \approx \bar{t}, E}{\bar{y} \,.\, \bar{s} \approx \bar{t}, E} \quad \bar{x} \mapsto \mathsf{e}$$

$$\frac{\bar{x} \approx \bar{t}, E}{E} \quad \bar{x} \mapsto \bar{t}$$

$$\frac{\mathsf{e} \approx \mathsf{e}, E}{E}$$

These rules operate on finite sequence of equations of the form $s \approx t$. The empty sequence is denoted by $\Box$. There are overlapping between the first rule, and third and fourth rules. If we encounter a case that two rules are applicable, we apply the first rule and discard the other possibility. Note that the first rule is the key rule of minimal M-matching, where $c(\bar{s_1})$ is a separator.

We write $s \rightarrowtail_\sigma t$ for one-step application of an inference rule where σ is the obtained substitution and $s \rightarrowtail_\sigma^* t$ for zero or more step application, where σ is the union of the obtained substitutions. The correctness of MA has not been proven but it will be done by similar way as the case of Theorem Mmatch.

Conjecture 6.2
Let l and s be M-normalized terms and l is linear.

- $\rightarrowtail$-steps starting from $l \approx s$ always terminates,
- $l \approx s \rightarrowtail_\sigma^* \Box \quad \Leftrightarrow \quad \sigma$ is a minimal M-matcher of l for s,
- $l \approx s$ cannot reach $\Box$ by $\rightarrowtail \quad \Leftrightarrow \quad \sigma$ does not have matcher.

7 Application to Semantics of Mathematica Programming Language

Mathematica[Wol97] is a well-known commercial mathematical software system. Moreover the programming language in Mathematica is basically a functional language or rewrite language[Buc97]. We concentrate our attention to its rewriting part of Mathematica language called Mathematica/R (Mathematica restricted to Rewriting)[Buc97]. Mathematica/R can actually be considered as a pure functional language, but the presence of *sequence variables* is a new feature as a pure functional language constructs.

In the following, we illustrate a rough sketch of rewriting semantics of Mathematica/R by rewriting in STRSs. We treat Mathematica terms as STRS terms as follows:

Mathematica/R	STRSs
`x_`	x
`x___`	$\bar{x}$
$f[t_1, \cdots, t_n]$	$f(t'_1 \centerdot \cdots \centerdot t'_n)$
`Sequence[]`	`e`
`Sequence`$[t_1, \cdots, t_n]$	$t'_1 \centerdot \cdots \centerdot t'_n$

An STRS term t' denotes the recursively translated term from Mathematica/R term t. A function definition l `:=` r of Mathematica/R is just translated into a rewrite rule $l' \rightarrow r'$ of a STRS R. Evaluation of Mathematica/R starting from a term s and ending with t are STRS rewriting

$$s' \rightarrow_{R\mathsf{M}}^! t'$$

where $\rightarrow_{R\mathsf{M}}^!$ denotes many step rewriting to a normal form. Minimal M-matching and innermost strategy in the definition of $\rightarrow_{R\mathsf{M}}$ corresponds to the evaluation strategy in Mathematica/R.

This rough sketch of semantics dropped many part of Mathematica/R, e.g.,

- conditional part in function definitions
- no linearity restriction on lhss
- priority between function definitions
- curried higher-order terms
- built-in function that takes special evaluation strategy, e.g., `If` function

These features will be formalized by various extensions of TRSs, namely conditional[Kap84], priority[BBKW89], applicative[NMI95], higher-order[Klo80] and context-sensitive[Luc96] term rewriting systems. However, properties of combination of these extensions of TRSs has not been well studied. Hence, to give complete semantics of Mathematica/R, more work is needed.

References

[BBKW89] J. C. M. Baeten, J. A. Bergstra, J. W. Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, (67):283–301, 1989.

[BD89] Leo Bachmair and Nachum Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, (67):173–201, 1989.

[Buc97] B. Buchberger. Mathematica as a rewrite language. pages 1–13. World Scientific, 1997. Proceedings of the second Fuji International Workshop on Functional and Logic Programming.

[JK86] J. P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15:1155–1194, 1986.

[Kap84] S. Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33(2):175–193, 1984.

[Klo80] J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, Mathematical Centre Tracts Nr. 127. CWI, Amsterdam, 1980.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992.

[Luc96] S. Lucas. Context-sensitive computations in confluent programs. In H. Kuchen and D. Swierstra, editors, *Proc. of the International Symposium on Programming Languages, Implementations, Logics, and Programs, PLILP'96*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1996. To appear.

[NMI95] K. Nakahara, A. Middeldorp, and T. Ida. A complete narrowing calculus for higher-order functional logic programming. In *Proceedings of Seventh International Conference on Programming Languages: Implementations, Logics, and Programming 95 (PLILP'95), Lecture Notes in Computer Science 982*, pages 97–114, 1995.

[Sti81] M. E. Stickel. A unification algorithm for associative-commutative functions. *Journal of Association for Comuting Machinery*, 28(3):423–434, 1981.

[Wal92] Uwe Waldmann. Semantics of order-sorted specifications. *Theoretical Computer Science*, 94:1–35, 1992.

[Wol97] S. Wolfram. *The Mathematica Book.* Wolfram Media, 1997.

A Proof of theorem 5.1

Since the definition of Mmatch is not overlapped, uniqueness of the returned values of Mmatch is clear.

[termination of Mmatch **]**. In the definition of Mmatch, The size of the term of the first argument of Mmatch is decreasing except for the equation (9) and (7). The one-step unfolding (9) yields

$$\begin{aligned}&\mathsf{Mmatch}(t_1\,.\,t_2, s\,.\,\mathsf{e}) \parallel \mathsf{Mmatch}(t_1\,.\,t_2, \mathsf{e}\,.\,s)\\ &= (\|_{(s_1,s_2)\in S}\ (\mathsf{Mmatch}(t_1,s_1)\cup\mathsf{Mmatch}(t_2,s_2))) \parallel (\|_{(s_1,s_2)\in S'}\ (\mathsf{Mmatch}(t_1,s_1)\cup\mathsf{Mmatch}(t_2,s_2)))\end{aligned}$$

where

$$S = \{(s_1, s_2) \mid s \,.\, \mathsf{e} =_A s_1 \,.\, s_2\}$$
$$S' = \{(s_1, s_2) \mid \mathsf{e} \,.\, s =_A s_1 \,.\, s_2\}.$$

Then we see that for every component of $\|$, the size of the term of the first argument of Mmatch is also decreasing. Since $\#S, S' \leq \infty$, the number of branches by $\|$ is finite. Hence Mmatch terminates.
[soundness]. Define $\theta \triangleq \mathsf{Mmatch}(l, s)$. We prove

(i) $\theta \neq \bot \;\Rightarrow\; l\theta =_{\mathsf{M}} s$ and θ is a minimal matcher of l for s,

(ii) $\theta = \bot \;\Rightarrow\; \forall\sigma. l\sigma \neq_{\mathsf{M}} s$

by induction on (l, s).

- For the cases of $(l,s) = (c,c), (c,d), (\mathsf{e},\mathsf{e}), (\mathsf{e},s), (x, s : \mathsf{Elm}), (\bar{x}, s : \mathsf{Elm}), (x, s : \mathsf{Word}), (\bar{x}, s : \mathsf{Word}), (f(t), f(s)), (f(t), g(s))$, the proposition is clear.
- Case $\theta = \mathsf{Mmatch}(t_1 \,.\, t_2, s), s : \mathsf{Elm}$. By definition, the following equation holds:

$$\begin{aligned}\mathsf{Mmatch}(t_1 \,.\, t_2, s) &= \mathsf{Mmatch}(t_1 \,.\, t_2, s \circ \mathsf{e}) \parallel \mathsf{Mmatch}(t_1 \,.\, t_2, \mathsf{e} \,.\, s) \\ &= (\mathsf{Mmatch}(t_1, s) \cup \mathsf{Mmatch}(t_2, \mathsf{e})) \parallel (\mathsf{Mmatch}(t_1, \mathsf{e}) \cup \mathsf{Mmatch}(t_2, s)).\end{aligned}$$

Define $\sigma_1 \triangleq \mathsf{Mmatch}(t_1, s)$, $\sigma_2 \triangleq \mathsf{Mmatch}(t_2, \mathsf{e})$, $\sigma \triangleq \sigma_1 \uplus \sigma_2$ and $\sigma' \triangleq \mathsf{Mmatch}(t_1, \mathsf{e}) \cup \mathsf{Mmatch}(t_2, s)$. Thus $\theta = \sigma \parallel \sigma'$.

(i) Case $\theta \neq \bot$.

– Case $(\sigma \neq \bot, \sigma' = \bot)$ or $(\bot \neq \sigma \leq_{\mathrm{m}} \sigma' \neq \bot)$. In this case, $\theta = \sigma$. By the induction hypothesis,

$$t_1\sigma_1 =_{\mathsf{M}} s \text{ and } \sigma_1 \text{ is a minimal M-matcher of } t_1 \text{ for } s \tag{8}$$
$$t_2\sigma_2 =_{\mathsf{M}} s \text{ and } \sigma_2 \text{ is a minimal M-matcher of } t_2 \text{ for } s \tag{9}$$

Thus

$$(t_1 \,.\, t_2)\theta = (t_1 \,.\, t_2)\sigma = t_1\sigma_1 \,.\, t_2\sigma_2 s \,.\, \mathsf{e} =_{\mathsf{M}} s.$$

Next we prove θ is a minimal M-matcher of $t_1 \,.\, t_2$ for s. In (8) and (9), the all possible values of (t_1, σ_1) and (t_2, σ_2) are

t_1	σ_1
x	$x \mapsto s$
s	ϵ

t_2	σ_2
$\bar{y}$	$\bar{y} \mapsto \mathsf{e}$
e	ϵ

From the all possible combination of (t_1, σ_1) and (t_2, σ_2) in the above table, we know that $\sigma_1 \uplus \sigma_2$ is a minimal M-matcher of $t_1 \,.\, t_2$ for s.

– Case $(\sigma = \bot, \sigma' \neq \bot)$ or $(\bot \neq \sigma \geq_{\mathrm{m}} \sigma' \neq \bot)$. Similar to the previous case.

(ii) Case $\theta = \bot$. In this case, $\sigma = \sigma' = \bot$. By the induction hypothesis,

$$(\forall\rho. t_1\rho \neq_{\mathsf{M}} s \text{ or } \forall\rho. t_2\rho \neq_{\mathsf{M}} \mathsf{e}) \text{ and } (\forall\rho. t_1\rho \neq_{\mathsf{M}} \mathsf{e} \text{ or } \forall\rho. t_2\rho \neq_{\mathsf{M}} s)$$

We want to show $\forall\rho.(t_1 \,.\, t_2)\rho \neq_{\mathsf{M}} s$. We take the contrapositive

$$\exists\rho.(t_1 \,.\, t_2)\rho =_{\mathsf{M}} s \;\Rightarrow\; (\exists\rho. t_1\rho =_{\mathsf{M}} s \text{ and } t_2\rho =_{\mathsf{M}} \mathsf{e}) \text{ or } (\exists\rho. t_1\rho =_{\mathsf{M}} \mathsf{e} \text{ and } t_2\rho =_{\mathsf{M}} s).$$

This clearly holds.

- Case $\theta = \mathsf{Mmatch}(t_1 \,.\, t_2, s)$, $\#s > 1$.

  (i) Case $\theta \neq \bot$. In this case, there exist $a_1, a_2 \in S$ such that

  $$\begin{aligned}\mathsf{Mmatch}(t_1 \,.\, t_2, s) &= \|_{(s_1,s_2)\in S}\ (\mathsf{Mmatch}(t_1, s_1) \cup \mathsf{Mmatch}(t_2, s_2)) \\ &= \mathsf{Mmatch}(t_1, a_1) \cup \mathsf{Mmatch}(t_2, a_2).\end{aligned}$$

  Define $\theta_1 \triangleq \mathsf{Mmatch}(t_1, a_1), \theta_2 \triangleq \mathsf{Mmatch}(t_2, a_2)$. By the induction hypothesis,

  $$t_1\theta_1 =_{\mathsf{M}} a_1,\ t_2\theta_2 =_{\mathsf{M}} a_2.$$

  Hence

  $$(t_1 \,.\, t_2)\theta = (t_1 \,.\, t_2)(\theta_1 \uplus \theta_2) =_{\mathsf{M}} a_1 \,.\, a_2 =_{\mathsf{M}} s.$$

  Next we prove θ is a minimal M-matcher of $t_1 \,.\, t_2$ for s. Now from (i) and the induction hypothesis,

  $$\begin{aligned}\forall s_1, s_2 \in \mathsf{T}_\Sigma(\mathsf{V}), \alpha_1 : \mathcal{V}(s_1) \to \mathsf{T}_\Sigma(\mathsf{V}), \alpha_2 : \mathcal{V}(t_2) \to \mathsf{T}_\Sigma(\mathsf{V}), \\ t_1\alpha_1 =_{\mathsf{M}} s_1 \text{ and } t_2\alpha_2 =_{\mathsf{M}} s_2 \ \Rightarrow\ \alpha_1 \uplus \alpha_2 \geq_{\mathrm{m}} \theta.\end{aligned}$$

  Let β be a arbitrary M-matcher of $t_1 \,.\, t_2$ for s. Then there exists $\beta_1 : \mathcal{V}(t_1) \to \mathsf{T}_\Sigma(\mathsf{V}), \beta_2 : \mathcal{V}(t_2) \to \mathsf{T}_\Sigma(\mathsf{V})$ such that $\beta = \beta_1 \uplus \beta_2$. Thus

  $$t_\beta = (t_1 \,.\, t_2)(\beta_1 \uplus \beta_2) = t_1\beta \,.\, t_2\beta.$$

  Since β is a M-matcher,

  $$t_1\beta \,.\, t_2\beta =_{\mathsf{M}} s.$$

  Since we assumed that s is I-normalized,

  $$t_1\beta \,.\, t_2\beta (\to_{\mathsf{I}}^! \circ =_{\mathsf{A}}) s$$

  This means there exists terms s_1, s_2 such that

  $$\begin{aligned}&s =_{\mathsf{A}} s_1 \,.\, s_2, \\ &t_1\beta \to_{\mathsf{I}}^! s_1,\ t_2\beta \to_{\mathsf{I}}^! s_2.\end{aligned}$$

  Hence by applying (), we obtain

  $$\beta = \beta_1 \uplus \beta_2 \geq_{\mathrm{m}} \theta.$$

  (ii) Case $\theta = \bot$. In this case, $\mathsf{Mmatch}(t_1 \,.\, t_2, s) = \|_{(s_1,s_2)\in S}\ (\mathsf{Mmatch}(t_1, s_1) \cup \mathsf{Mmatch}(t_2, s_2))$ where $S = \{(s_1, s_2) \mid s =_{\mathsf{A}} s_1 \,.\, s_2\}$. Clearly,

  $$\forall (s_1, s_2) \in S. \mathsf{Mmatch}(t_1, s_1) = \mathsf{Mmatch}(t_2, s_2) = \bot.$$

  Therefore by the induction hypothesis,

  $$\forall (s_1, s_2) \in S. (\forall \sigma. t_1\sigma \neq_{\mathsf{M}} s_1) \text{ and } (\forall \sigma. t_2\sigma \neq_{\mathsf{M}} s_2).$$

  Hence $\forall \sigma. (t_1 \,.\, t_2)\sigma \neq_{\mathsf{M}} s$.

[completeness]. The part (i) of the completeness is direct consequence of the contrapositive of (i) of the soundness.

We prove

$$l\theta =_{\mathsf{M}} s \text{ and } \theta \text{ is a minimal M-matcher of } l \text{ for } s \ \Rightarrow\ \theta = \mathsf{Mmatch}(l, s)$$

by induction on (l, s).

- For the cases of $(l,s) = (c,c),(c,d),(\mathsf{e},\mathsf{e}),(\mathsf{e},s),(x,s : \mathsf{Elm}),(\bar{x},s : \mathsf{Elm}),(x,s : \mathsf{Word}),(\bar{x},s : \mathsf{Word}),f(t)\theta = f(s),(f(t),g(s))$, the proposition is clear.

- Case $(t_1 \,.\, t_2)\theta =_{\mathsf{M}} s$ and $s : \mathsf{Elm}$. There exists $\theta_1 : \mathcal{V}(t_1) \to \mathsf{T}_\Sigma(\mathsf{V})$, $\theta_2 : \mathcal{V}(t_2) \to \mathsf{T}_\Sigma(\mathsf{V})$ such that $s =_{\mathsf{I}} s_1 \,.\, s_2$, $t_1\theta_1 =_{\mathsf{M}} s_1$ and $t_2\theta_2 =_{\mathsf{M}} s_2$. Since $s : \mathsf{Elm}$, the possible combination of the decomposed terms s_1, s_2 are

$$(s_1 = \mathsf{e}, s_2 = s) \text{ or } (s_1 = s, s_2 = \mathsf{e}). \tag{10}$$

  By the induction hypothesis,

$$\mathsf{Mmatch}(t_1, s_1) = \theta_1 \text{ and } \mathsf{Mmatch}(t_2, s_2) = \theta_2.$$

  Now we notice the following:

$$\begin{aligned}
\mathsf{Mmatch}(t_1 \,.\, t_2, s) &= \mathsf{Mmatch}(t_1 \,.\, t_2, s \,.\, \mathsf{e}) \parallel \mathsf{Mmatch}(t_1 \,.\, t_2, \mathsf{e} \,.\, s) \\
&= \|_{(s_1,s_2)\in S} \left(\mathsf{Mmatch}(t_1, s_1) \cup \mathsf{Mmatch}(t_2, s_2)\right) \\
&\quad \parallel \|_{(s_1,s_2)\in S} \left(\mathsf{Mmatch}(t_1, s_1) \cup \mathsf{Mmatch}(t_2, s_2)\right) \\
&\quad \text{where } S = \{(s_1, s_2) \mid s \circ \mathsf{e} =_{\mathsf{A}} s_1 \,.\, s_2\} \\
&\qquad\quad S' = \{(s_1, s_2) \mid \mathsf{e} \circ s =_{\mathsf{A}} s_1 \,.\, s_2\} \\
&= (\mathsf{Mmatch}(t_1, s) \cup \mathsf{Mmatch}(t_2, \mathsf{e})) \parallel (\mathsf{Mmatch}(t_1, \mathsf{e}) \cup \mathsf{Mmatch}(t_2, s))
\end{aligned}$$

  Then by the equation (10),

$$\begin{aligned}
\theta &= \theta_1 \cup \theta_2 \\
&= \mathsf{Mmatch}(t_1, s_1) \cup \mathsf{Mmatch}(t_2, s_2) \\
&= (\mathsf{Mmatch}(t_1, s) \cup \mathsf{Mmatch}(t_2, \mathsf{e})) \parallel \mathsf{Mmatch}(t_1, \mathsf{e}) \cup \mathsf{Mmatch}(t_2, s) \\
&= \mathsf{Mmatch}(t_1 \,.\, t_2, s)
\end{aligned}$$

- Case $(a_1 \,.\, a_2)\theta =_{\mathsf{M}} s$ and $\#s > 1$. Let $\theta = \{x_1 \mapsto a_1, x_2 \mapsto a_2, \cdots, x_n \mapsto a_n\} = \theta_1 \uplus \theta_2$ where $\theta_1 = \{x_1 \mapsto a_1\}, \theta_2 = \{x_2 \mapsto a_2, \cdots, x_n \mapsto a_n\}$.

  Choose minimal (with respect to the length of word) word l_1 and corresponding l_2 such that

  - $t_1 \,.\, t_2 =_{\mathsf{A}} l_1 \,.\, l_1$,
  - $\mathcal{V}(l_1) = \{x_1\}$.

  Then the form of l_1 is either the following two cases:

  - $l_1 = b_1 \,.\, \cdots \,.\, b_k \,.\, a_1$,
  - $l_1 = b_1 \,.\, \cdots \,.\, b_k \,.\, f(t)$,

  where $b_1, \cdots, b_k (k$ may be $0)$ are ground terms, $f \in \mathcal{F}$, and t is a term such that $x_1 \in \mathcal{V}(t)$.

  - Case $l_1 = b_1 \,.\, \cdots \,.\, b_k \,.\, x_1$. Since a substitution $\{x_1 \mapsto a_1\}$ is a minimal matcher of $b_1 \,.\, \cdots \,.\, b_k \,.\, x_1$ for $b_1 \,.\, \cdots \,.\, b_k \,.\, t_1$, θ_1 is a minimal matcher of l_1.

    Let $\rho : \{x_2, \cdots, x_k\} \to \mathsf{T}_\Sigma(\mathsf{V})$ be a arbitrary substitution satisfying $l_2\theta_2 =_{\mathsf{M}} l_2\rho$. Now we define

$$\theta \triangleq \theta_1 \uplus \theta_2 \text{ and } \sigma \triangleq \theta_1 \uplus \rho.$$

Then

$$l\sigma = (l_1 \,.\, l_2)(\theta_1 \uplus \rho) = l_1\theta_1 \,.\, l_2\rho =_{\mathsf{M}} l_1\theta_1 \,.\, l_2\theta_2 = (l_1 \,.\, l_2)(\theta_1 \uplus \theta_2) = l\theta$$

Since θ is a minimal matcher of t,

$$|x_1\theta| \,.\, |x_2\theta| \,.\, \cdots \,.\, |x_n\theta| \leq |x_1\sigma| \,.\, |x_2\sigma| \,.\, \cdots \,.\, |x_n\sigma|.$$

Namely,

$$|x_1\theta_1| \,.\, |x_2\theta_2| \,.\, \cdots \,.\, |x_n\theta_2| \leq |x_1\theta_1| \,.\, |x_2\rho| \,.\, \cdots \,.\, |x_n\rho|.$$

Thus

$$|x_2\theta_2| \,.\, \cdots \,.\, |x_n\theta_2| \leq |x_2\rho| \,.\, \cdots \,.\, |x_n\rho|.$$

This means θ_2 is a minimal matcher of l_2. Therefore θ_1 and θ_2 are minimal matchers for l_1 and l_2 respectively.

– Case $l_1 = b_1 \,.\, \cdots \,.\, b_k \,.\, f(t)$. Clearly, θ_1 is a minimal matcher of l_1 and θ_2 is a minimal matcher of l_2.

By the induction hypothesis, $\mathsf{Mmatch}(l_1, l_1\theta_1) = l_1\theta_1$ and $\mathsf{Mmatch}(l_2, l_2\theta_2) = l_2\theta_2$. By the definition of Mmatch,

$$\mathsf{Mmatch}(t_1 \,.\, t_2, s) = \mathsf{Mmatch}(l_1 \,.\, l_2, s) = \|_{(s_1,s_2)\in S} \, (\mathsf{Mmatch}(l_1, s_1) \cup \mathsf{Mmatch}(l_2, s_2)) \tag{11}$$

By the soundness of Mmatch, each component $(\mathsf{Mmatch}(l_1, s_1) \cup \mathsf{Mmatch}(l_2, s_2))(\neq \bot)$, in (11) are matchers of $l_1 \,.\, l_2$ for s. Since θ is the minimal matcher of $l_1 \,.\, l_2$ for s, $(l_1\theta_1, l_2\theta_2) \in S$, and

$$\begin{aligned}\theta &= l_1\theta_1 \uplus l_2\theta_2 = \mathsf{Mmatch}(l_1, l_1\theta_1) \cup \mathsf{Mmatch}(l_2, l_2\theta_2) \\ &= \|_{(s_1,s_2)\in S} \, (\mathsf{Mmatch}(l_1, s_1) \cup \mathsf{Mmatch}(l_2, s_2)).\end{aligned}$$

Hence $\theta = \mathsf{Mmatch}(l_1 \,.\, l_2, s)$. ■