Flat Matching

Temur Kutsia

Research Institute for Symbolic Computation, Johannes Kepler University,
A-4040 Linz, Austria

Abstract

We study matching in flat theories both from theoretical and practical points of view. A flat theory is defined by the axiom $f(\overline{x}, f(\overline{y}), \overline{z}) \doteq f(\overline{x}, \overline{y}, \overline{z})$ that indicates that nested occurrences of the function symbol f can be flattened out. From the theoretical side, we design a procedure to solve a system of flat matching equations and prove its soundness, completeness, and minimality. The minimal complete set of matchers for such a system can be infinite. The procedure enumerates this set and stops if it is finite. We identify a class of problems on which the procedure stops. From the practical point of view, we look into restrictions of the procedure that give an incomplete terminating algorithm. From this perspective, we give a set of rules that, in our opinion, describes the precise semantics for the flat matching algorithm implemented in the Mathematica system.

1. Introduction

This paper pursues two major goals. The first one is to study theoretical properties of matching in flat theories, to design a complete procedure to solve flat matching problems, and to investigate terminating restrictions. The second goal is to formally characterize one of such restrictions, Mathematica's flat matching algorithm, give its precise semantics, and compare it to the theoretically complete and minimal procedure. A flat theory is defined by the axiom $f(\overline{x}, f(\overline{y}), \overline{z}) \doteq f(\overline{x}, \overline{y}, \overline{z})$ that indicates that nested occurrences of the function symbol f can be flattened out. Function symbols with this property are called flat function symbols. Their arity is not fixed. The variables $\overline{x}$, $\overline{y}$, and $\overline{z}$ are sequence variables. They can be instantiated by finite, possibly empty, sequences of terms. Flat symbols appear in the programming language of the Mathematica system, by assigning to certain symbols the attribute `Flat`. This property affects both evaluation and pattern matching in Mathematica.

Matching in flat theories is interesting per se, from the unification theory point of view, without relating it to any particular implementation. In this paper we study theories with flexible arity function symbols where some of those symbols can be flat, and variables

Email address: `Temur.Kutsia@risc.uni-linz.ac.at` (Temur Kutsia).

are of three kinds: individual variables (can be instantiated by a single term), function variables (can be instantiated by a function symbol or a function variable), and sequence variables (can be instantiated by a term sequence). Interest to a flat theory is caused by the fact that it is an example of a theory, not "cooked artificially", that has infinitary decidable matching.

Flatness is often confused with associativity. Although these properties are similar, they are not the same. Even more, flat and associative theories belong to different classes in the unification/matching hierarchy: associative matching is finitary (minimal complete set of matchers always exists and is finite), while flat matching is infinitary (minimal complete set of matchers always exists and for some problems it may be infinite). Similarity can be found if one restricts flat theories to have only individual variables (i.e., forbids sequence and function variables), and instead of associative matching considers associative matching with the unit element (AU-matching). Even in this case they are just very similar, not exactly the same. The reason is hidden in the fact that flat function symbols have flexible arity, while associative functions are binary. The matching problem $f(x, y) \ll f(a, b)$ underlines this: The minimal complete set of matchers for it, when f is associative, is a proper subset of the minimal complete set of matchers for flat f, e.g. the substitution $\{x \mapsto f(a), y \mapsto f(b)\}$ is a matcher for flat f, but not for associative f. In the latter case $f(a)$ and $f(b)$ are not well-formed terms, because f is binary.

This paper investigates flat matching in details. We introduce rules to solve flat matching equations and impose a control on these rules that gives a solving procedure. We prove that the procedure enumerates the minimal complete set of matchers, and terminates if this set if finite. There are flat matching problems that have infinite minimal complete set of matchers, which implies that flat matching is infinitary. We identify classes of flat matching problems on which the procedure terminates. We also show how to obtain an incomplete terminating algorithm for arbitrary flat matching problems, slightly modifying one of the rules in the procedure.

Searching for practically useful terminating restrictions of the flat matching procedure is the motivation behind the second part of the paper. It represents an attempt to give a precise semantics for the implementation of flat matching in Mathematica. We describe Mathematica's flat matching algorithm and compare it to the theoretically complete procedure. Obviously, a practically useful method that solves flat matching equations should be terminating and, hence, incomplete (unless it provides a finite description of the infinite complete set of flat matchers). Therefore, it is natural that Mathematica's flat solving method is incomplete. Interesting questions are what its semantics is, what the rules behind it are, and how it works. These questions, as far as we know, have not been formally answered. Informal explanations can be found elsewhere, see, e.g. [37, 25, 16, 36]. The MathGroup Archive [1] contains more than 600 postings that discuss and try to clarify the flat attribute. It seems that for many people who program in Mathematica the behavior of flat matching is quite confusing. Understanding proper semantics of programming constructs is very important to program correctly, and we hope that the last part of this paper will contribute into clarifying the semantics of Mathematica's flat matching. To the best of our knowledge, it gives the first formal account of the corresponding mechanism implemented in the system.

The paper consists of the following sections: Section 1 is the introduction. In Section 2 the basic definitions are given. In Section 3 the flat matching procedure is defined, its properties are proved, and some of the terminating restrictions are introduced. Section 4 gives a detailed formal account of the flat matching algorithm of Mathematica. Related work is briefly surveyed in Section 5. The paper ends with concluding remarks given in Section 6.

2. Preliminaries

We assume some familiarity with the standard notions of the unification theory [3] and with programming in Mathematica.

First we start with studying flat theories. The alphabet we are using consist of mutually disjoint countable sets of individual variables $\mathcal{V}_{\mathrm{Ind}}$, sequence variables $\mathcal{V}_{\mathrm{Seq}}$, function variables $\mathcal{V}_{\mathrm{Fun}}$, and function symbols $\mathcal{F}$. All the symbols in $\mathcal{F}$ have flexible arity. We will use x, y, z for individual variables, $\overline{x}, \overline{y}, \overline{z}$ for sequence variables, F, G, H for function variables, and a, b, c, f, g, h for function symbols. The set of variables $\mathcal{V}_{\mathrm{Ind}} \cup \mathcal{V}_{\mathrm{Seq}} \cup \mathcal{V}_{\mathrm{Fun}}$ is denoted by $\mathcal{V}$. *Terms* (over $\mathcal{F}$ and $\mathcal{V}$) are defined by the following grammar:

$$t ::= x \mid \overline{x} \mid f(t_1, \dots, t_n) \mid F(t_1, \dots, t_n).$$

An *individual term* is a term that is not a sequence variable. When it is not ambiguous, we write f for the term $f()$ where $f \in \mathcal{F}$. In particular, we will always write a, b, c for $a(), b(), c()$. Terms are denoted with s, t, r.

The *set of variables* of a term t is denoted by $\mathcal{V}(t)$. We can use the subscripts Ind, Seq, and Fun to indicate the sets of individual, sequence, and function variables of a term, respectively. A *ground* term is a term without variables. These definitions are generalized for any syntactic object throughout the paper. The *head* of a term is its root symbol. The size of a term t, denoted $\mathit{size}(t)$, is the number of symbols in it.

A *substitution* is a mapping from individual variables to individual terms, from sequence variables to finite term sequences, and from function variables to function variables and symbols such that all but finitely many variables are mapped to themselves. (We do not distinguish between a singleton term sequence and its sole element.) We will use lower case Greek letters for substitutions, with ε for the empty substitution.

For a substitution σ, the domain is the set of variables

$$\mathit{dom}(\sigma) = \{v \in \mathcal{V} \mid \sigma(v) \neq v\}.$$

A substitution can be represented explicitly as a function by a finite set of bindings of variables in its domain: $\{v \mapsto \sigma(v) \mid v \in \mathit{dom}(\sigma)\}$. For readability, we put term sequences in parentheses. For instance, the set $\{x \mapsto f(a, \overline{y}), \overline{x} \mapsto (), \overline{y} \mapsto (a, F(f(b)), x), F \mapsto g\}$ is such a representation of a substitution.

Substitutions are extended to terms:

$$x\sigma = \sigma(x). \qquad (f(t_1, \dots, t_n))\sigma = f(t_1\sigma, \dots, t_n\sigma).$$
$$\overline{x}\sigma = \sigma(\overline{x}). \qquad (F(t_1, \dots, t_n))\sigma = \sigma(F)(t_1\sigma, \dots, t_n\sigma).$$

In a similar way substitution are extended to term sequences:

$$(t_1, \dots, t_n)\sigma = (t_1\sigma, \dots, t_n\sigma).$$

We call $t\sigma$ and $(t_1, \dots, t_n)\sigma$ *instances* of respectively t and $(t_1, \dots, t_n)$ under σ. We use $\tilde{s}$, $\tilde{t}$, and $\tilde{r}$ to denote finite, possibly empty, term sequences.

Composition of two substitutions σ and ϑ, written $\sigma\vartheta$, is defined by $\tilde{s}(\sigma\vartheta) = (\tilde{s}\sigma)\vartheta$ and $F(\sigma\vartheta) = (F\sigma)\vartheta$.

An *equation* (over $\mathcal{F}$ and $\mathcal{V}$) is a pair of individual terms $\langle s, t\rangle$, written $s \doteq t$. Substitutions are extended to equations in the usual way. The notion of size extends to sequences, substitutions and equations:

$$\begin{aligned}
size(t_1, \dots, t_n) &= \sum_{i=1}^{n} size(t_i),\\
size(\sigma) &= \sum_{x \in dom(\sigma)} size(x\sigma) + \sum_{\overline{x} \in dom(\sigma)} size(\overline{x}\sigma) + \sum_{F \in dom(\sigma)} size(F()\sigma),\\
size(s \doteq t) &= size(s) + size(t).
\end{aligned}$$

Given a set E of equations over $\mathcal{F}$ and $\mathcal{V}$, we denote by $\doteq_E$ the least congruence relation on the set of finite sequences of terms (over $\mathcal{F}$ and $\mathcal{V}$) that is closed under substitution application and contains E. The set $\doteq_E$ is called an *equational theory* defined by E. Slightly abusing the terminology, we will also call the set E an equational theory or an E-theory. The *signature* of E, denoted $sig(E)$, is the set of all function symbols occurring in E. A function symbol is called *free* with respect to E if it does not occur in $sig(E)$.

A substitution σ is *more general* than a substitution ϑ on a set of variables $\mathcal{X}$ modulo an equational theory E, denoted $\sigma \leq_E^{\mathcal{X}} \vartheta$, if there exists a φ such that $v\sigma\varphi \doteq_E v\vartheta$ for all individual and sequence variables $v \in \mathcal{X}$ and $F()\sigma\varphi \doteq_E F()\vartheta$ for all function variables $F \in \mathcal{X}$.

Solving equations in an equational theory E is called *E-unification.* The fact that the equation $s \doteq t$ has to be solved in an equational theory E is written as $s \doteq_E^? t$. If one of the sides of an equation is ground, then it is called a *matching equation*, and solving such equations in a theory E is called *E-matching.* We write matching equations as $s \ll t$, where t is ground, and indicate that it has to be solved in an E-theory by writing $s \ll_E^? t$.

Let E be an equational theory with $sig(E) \subseteq \mathcal{F}$. An *E-matching problem* over $\mathcal{F}$ is a finite set of matching equations over $\mathcal{F}$ and $\mathcal{V}$:

$$\Gamma = \{s_1 \ll_E^? t_1, \dots, s_n \ll_E^? t_n\}.$$

An *E-matcher* of Γ is a substitution σ such that $s_i\sigma \doteq_E t_i$ for all $1 \leq i \leq n$. The set of all E-matchers of Γ is denoted by $match_E(\Gamma)$. Γ is *E-matchable*, or *E-solvable*, if $match_E(\Gamma) \neq \emptyset$.

A *minimal complete set of E-matchers* of Γ is a set S of substitutions with the following three properties:

(1) (Correctness) $S \subseteq match_E(\Gamma)$, i.e., each element of S is an E-matcher of Γ;
(2) (Completeness) For each $\vartheta \in match_E(\Gamma)$ there exists $\sigma \in S$ such that $\sigma \leq_E^{\mathcal{V}(\Gamma)} \vartheta$.
(3) (Minimality) The set S is *minimal* with respect to $\mathcal{V}(\Gamma)$ modulo E, i.e., if there exist $\sigma, \vartheta \in S$ such that $\sigma \leq_E^{\mathcal{V}(\Gamma)} \vartheta$ then $\sigma = \vartheta$.

The equality $f(\overline{x}, f(\overline{y}), \overline{z}) \doteq f(\overline{x}, \overline{y}, \overline{z})$ specifies the property called *flatness* for the function symbol f that is called a *flat* function symbol. A *flat theory*, or shortly an F-theory, is defined by a set of equalities that express flatness of function symbols. Below we consider general F-matching, i.e., besides flat symbols we can have also arbitrary free function symbols in matching problems. A term or an equation is in the *flattened form* if all nested occurrences of flat function symbols are flattened out.

Flat matching is decidable. It is easy to observe that if a flat matching problem Γ is solvable, then it has a solution in the flattened form (i.e. where all terms are flattened) whose size is bounded by the size of Γ. There are finitely many flattened substitutions whose size does not exceed the size of Γ, which map variables in Γ to terms, finite term sequences, and function symbols occurring in Γ. Hence, we can simply check whether any of these substitutions is a solution of Γ.

An interesting property of flat matching is that some problems may have an infinite minimal complete set of matchers:

Example 1. The minimal complete set of matchers for the flat matching problem $\{f(\overline{x}) \ll^?_{\mathsf{F}} f(a)\}$ with flat f is $\{\{\overline{x} \mapsto a\}, \{\overline{x} \mapsto f(a)\}, \{\overline{x} \mapsto (a, f())\}, \{\overline{x} \mapsto (f(a), f())\}, \{\overline{x} \mapsto (f(), a)\}, \{\overline{x} \mapsto (f(), f(a))\}, \{\overline{x} \mapsto (f(), a, f())\}, \{\overline{x} \mapsto (f(), f(a), f())\}, \ldots\}$.

3. Flat Matching Procedure

We describe a procedure that enumerates the minimal complete set of matchers for flat matching problems. The procedure will be defined in a rule-based manner, in the spirit of [21]. Its inference system $\mathfrak{R}$ consists of the rules presented below. Rules operate on systems. A *system* is either the symbol $\bot$ (failure) or a pair $\Gamma; \sigma$. It is assumed that equations are kept in the flattened form.

T: **Trivial**

$\{s \ll^?_{\mathsf{F}} s\} \cup \Gamma;\ \sigma \Longrightarrow \Gamma;\ \sigma.$

S: **Solve**

$\{x \ll^?_{\mathsf{F}} t\} \cup \Gamma;\ \sigma \Longrightarrow \Gamma\vartheta;\ \sigma\vartheta,$ where $\vartheta = \{x \mapsto t\}$.

FVE: **Function Variable Elimination**

$\{F(\tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma\vartheta;\ \sigma\vartheta,$ where $\vartheta = \{F \mapsto f\}$.

Dec: **Decomposition**

$\{f(s, \tilde{s}) \ll^?_{\mathsf{F}} f(t, \tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{s \ll^?_{\mathsf{F}} t,\ f(\tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma;\ \sigma,$ if $s \notin \mathcal{V}$.

IVE: **Individual Variable Elimination**

$\{f(x, \tilde{s}) \ll^?_{\mathsf{F}} f(t, \tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma\vartheta;\ \sigma\vartheta$ where $\vartheta = \{x \mapsto t\}$.

SVP: **Sequence Variable Projection**

$\{f(\overline{x}, \tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma\vartheta;\ \sigma\vartheta$ where $\vartheta = \{\overline{x} \mapsto ()\}$.

SVW: **Sequence Variable Widening**

$\{f(\overline{x}, \tilde{s}) \ll^?_{\mathsf{F}} f(t, \tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\overline{x}, \tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma\vartheta;\ \sigma\vartheta$ where $\vartheta = \{\overline{x} \mapsto (t, \overline{x})\}$.

IVE-FH: **Individual Variable Elimination under Flat Head**

$\{f(x, \tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t}_1, \tilde{t}_2)\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t}_2)\} \cup \Gamma\vartheta;\ \sigma\vartheta$

where f is flat and $\vartheta = \{x \mapsto f(\tilde{t}_1)\}$.

SVW-FH: **Sequence Variable Widening under Flat Head**

$$\{f(\overline{x}, \tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t}_1, \tilde{t}_2)\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\overline{x}, \tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t}_2)\} \cup \Gamma\vartheta;\ \sigma\vartheta$$

where f is flat and $\vartheta = \{\overline{x} \mapsto (f(\tilde{t}_1), \overline{x})\}$.

We call the substitutions computed at transformation steps (the ϑ's in the rules in $\mathfrak{R}$) the *local* substitutions. We may write $\Gamma_1; \sigma_1 \Longrightarrow_{\mathsf{R},\vartheta} \Gamma_2; \sigma_2$ to indicate that the system $\Gamma_1; \sigma_1$ was transformed into $\Gamma_2; \sigma_2$ by applying the rule $\mathsf{R} \in \mathfrak{R}$ with the local substitution ϑ. A *derivation* is a sequence of system transformations $\Gamma_1; \sigma_1 \Longrightarrow_{\mathsf{R}_1,\vartheta_1} \Gamma_2; \sigma_2 \Longrightarrow_{\mathsf{R}_2,\vartheta_2} \cdots$. Some of the subscripts will be omitted if they are not relevant for the context. A *selection strategy* $\mathcal{S}$ is a function which given a derivation $\Gamma_1; \sigma_1 \Longrightarrow_{\mathsf{R}_1,\vartheta_1} \cdots \Longrightarrow_{\mathsf{R}_{n-1},\vartheta_{n-1}} \Gamma_n; \sigma_n$ returns a matching equation in Γ_n. A derivation is *via* a selection strategy $\mathcal{S}$ if in the derivation all choices of selected equations are performed according to $\mathcal{S}$. We will use the abbreviation $\Gamma_1; \sigma_1 \Longrightarrow^+_{\vartheta} \Gamma_n; \sigma_n$ for the derivation $\Gamma_1; \sigma_1 \Longrightarrow_{\vartheta_1} \Gamma_2; \sigma_2 \Longrightarrow_{\vartheta_2} \cdots \Longrightarrow_{\vartheta_{n-1}} \Gamma_n; \sigma_n$, where $\vartheta = \vartheta_1 \cdots \vartheta_{n-1}$.

Definition 2. A *flat matching procedure* $\mathfrak{F}$ is any program that takes a system $\Gamma; \varepsilon$ and a selection strategy $\mathcal{S}$ as input, where Γ is a flat matching problem, and uses the rules in $\mathfrak{R}$ to generate a complete tree of derivations via $\mathcal{S}$ in the following way:

(1) The root of the tree is labeled with $\Gamma; \varepsilon$.
(2) Each branch of the tree is a derivation via $\mathcal{S}$. The nodes in the tree are systems.
(3) If a system in a node is unsolvable, the branch is extended with $\Gamma; \varepsilon \Longrightarrow \bot$. Otherwise, the system is transformed by the rules in $\mathfrak{R}$. If several rules, or different instances of the same rule are applicable to the selected equation, they are applied concurrently.

The leaves of such a tree are labeled either with $\bot$ (failed branches) or with systems of the form $\emptyset; \sigma$ (successful branches). Since each selected equation can be transformed by finitely many different ways, the tree is finitely branching. A substitution σ is called an *answer of* Γ *computed by* $\mathfrak{F}$, or just a *computed answer* of Γ if $\emptyset; \sigma$ is the leaf of a successful branch of the solving tree for Γ. We denote by $\mathit{comp}_{\mathfrak{F}}(\Gamma)$ the set of answers of Γ computed by $\mathfrak{F}$.

To illustrate how $\mathfrak{F}$ works, we give an example below. (More examples can be found in Section 4 in the context of comparing $\mathfrak{F}$ to flat matching in Mathematica.)

Example 3. We show a successful derivation for a flat matching problem $\{f(\overline{x}, g(\overline{x})) \ll^?_{\mathsf{F}} f(a, g(f(), a))\}$, where f is flat and g is free. (Remember that equations are flattened after each application of substitutions.)

$$\begin{aligned}
&\{f(\overline{x}, g(\overline{x})) \ll^?_{\mathsf{F}} f(a, g(f(), a))\};\ \varepsilon \\
&\quad\Longrightarrow_{\mathsf{SVW\text{-}FH}} \{f(\overline{x}, g(f(), \overline{x})) \ll^?_{\mathsf{F}} f(a, g(f(), a))\};\ \{\overline{x} \mapsto (f(), \overline{x})\} \\
&\quad\Longrightarrow_{\mathsf{SVW}} \{f(\overline{x}, g(f(), a, \overline{x})) \ll^?_{\mathsf{F}} f(g(f(), a))\};\ \{\overline{x} \mapsto (f(), a, \overline{x})\} \\
&\quad\Longrightarrow_{\mathsf{SVP}} \{f(g(f(), a)) \ll^?_{\mathsf{F}} f(g(f(), a))\};\ \{\overline{x} \mapsto (f(), a)\} \\
&\quad\Longrightarrow_{\mathsf{T}} \emptyset;\ \{\overline{x} \mapsto (f(), a)\}.
\end{aligned}$$

The procedure $\mathfrak{F}$ is sound, complete, and enumerates a minimal complete set of matchers for a given flat matching problem. It follows from the theorems we prove below.

Theorem 4 (Soundness)**.** *Let Γ be a flat matching problem and $\sigma \in \mathit{comp}_{\mathfrak{F}}(\Gamma)$. Then $\sigma \in \mathit{match}_{\mathsf{F}}(\Gamma)$.*

Proof. The theorem follows from the fact that each rule in $\mathfrak{R}$ is sound: If $\Gamma_1; \sigma_1 \Longrightarrow_{\mathsf{R},\vartheta} \Gamma_2; \sigma_2$ by a rule $\mathsf{R} \in \mathfrak{R}$, and $\varphi \in \mathit{match}_{\mathsf{F}}(\Gamma_2)$, then $\vartheta\varphi \in \mathit{match}_{\mathsf{F}}(\Gamma_1)$. Correctness of this fact is easy to establish: Inspection of the rules in $\mathfrak{R}$ is sufficient. $\Box$

Theorem 5 (Completeness). *Let Γ be a flat matching problem and $\sigma \in \mathit{match}_{\mathsf{F}}(\Gamma)$. Then $\sigma|_{\mathcal{V}(\Gamma)} \in \mathit{comp}_{\mathfrak{F}}(\Gamma)$.*

Proof. By well-founded induction on the size of σ. We shall construct the derivation from Γ (via a given selection strategy $\mathcal{S}$) that ends with $\emptyset; \sigma|_{\mathcal{V}(\Gamma)}$. Let $s \ll^?_{\mathsf{F}} t$ be an equation in Γ selected by $\mathcal{S}$. Depending on the shape of s and t we may have different cases. Here we consider only the case when $s = f(\overline{x}, \tilde{s})$ and $t = f(t, \tilde{t})$, where f is flat. The other cases are similar. For $\overline{x}\sigma$ we have one of the following alternatives: (i) $\overline{x}\sigma = (t, \tilde{r})$, (ii) $\overline{x}\sigma = ()$, (iii) $\overline{x}\sigma = (f(), \tilde{r})$, or (iv) $\overline{x}\sigma = (f(t, \tilde{r}_1), \tilde{r}_2)$. In (i) $\tilde{r}$ is an initial subsequence of $\tilde{t}$ and we extend the derivation by the rule SVW and the substitution $\vartheta = \{\overline{x} \mapsto (t, \overline{x})\}$ arriving at $\Delta = \{f(\overline{x}, \tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma\vartheta$. It has a matcher $\sigma' = (\sigma \setminus \{\overline{x} \mapsto (t, \tilde{r})\}) \cup \{\overline{x} \mapsto \tilde{r}\}$ whose size is less than that of σ's. Therefore, by the induction hypothesis, $\sigma'|_{\mathcal{V}(\Delta)} \in \mathit{comp}_{\mathfrak{F}}(\Delta)$. By the definition of $\mathit{comp}_{\mathfrak{F}}$, we have $\vartheta(\sigma'|_{\mathcal{V}(\Delta)}) \in \mathit{comp}_{\mathfrak{F}}(\Gamma)$, from which we obtain $\vartheta(\sigma'|_{\mathcal{V}(\Delta)}) = (\vartheta\sigma')|_{\mathcal{V}(\Delta)} = \sigma|_{\mathcal{V}(\Delta)} = \sigma|_{\mathcal{V}(\Gamma)}$. Hence, $\sigma|_{\mathcal{V}(\Gamma)} \in \mathit{comp}_{\mathfrak{F}}(\Gamma)$. In the case (ii) we would proceed with SVP, and in (iii) and (iv) with $\mathsf{SVW\text{-}FH}$. $\Box$

Theorem 6 (Minimality). *Let Γ be a flat matching problem. Then $\mathit{comp}_{\mathfrak{F}}(\Gamma)$ is minimal.*

Proof. Follows from the fact that if a matching equation is transformed in two different ways by rules in $\mathfrak{R}$, then the local substitutions ϑ_1 and ϑ_2 used in these transformations can not be "brought" to the same instance: There are no φ_1 and φ_2 such that $v\vartheta_1\varphi_1 \doteq_{\mathsf{F}} v\vartheta_2\varphi_2$ for all $v \in \mathcal{V}_{\mathrm{Ind}}(\Gamma) \cup \mathcal{V}_{\mathrm{Seq}}(\Gamma)$ and $F()\vartheta_1\varphi_1 \doteq_{\mathsf{F}} F()\vartheta_2\varphi_2$ for all $F \in \mathcal{V}_{\mathrm{Fun}}(\Gamma)$. We call such ϑ_1 and ϑ_2 *disjoint* with respect to $\mathcal{V}(\Gamma)$. This property can be established by inspecting the rules in $\mathfrak{R}$. We just demonstrate it here for the rules SVW and $\mathsf{SVW\text{-}FH}$. Let $\Delta; \sigma$ be a system in a derivation, which is transformed in different ways by SVW and $\mathsf{SVW\text{-}FH}$, where $f(\overline{x}, \tilde{s}) \ll^?_{\mathsf{F}} f(t, \tilde{t})$ is a selected equation with f being flat. Since the transformation rules do not introduce new variables, we have $\overline{x} \in \mathcal{V}(\Gamma)$. The head of t is not f, because terms in systems are kept flattened. Both SVW and $\mathsf{SVW\text{-}FH}$ can transform the selected equation. SVW can do it with the local substitution $\vartheta_1 = \{\overline{x} \mapsto (t, \overline{x})\}$, while $\mathsf{SVW\text{-}FH}$ does it in finitely many ways. Let $\vartheta_2 = \{\overline{x} \mapsto (f(\tilde{t}'), \overline{x})\}$ be a local substitution used by $\mathsf{SVW\text{-}FH}$ in this transformation, where $\tilde{t}'$ is a (possibly empty) initial subsequence of the sequence $(t, \tilde{t})$. Then it is clear that there are no φ_1 and φ_2 such that $\overline{x}\vartheta_1\varphi_1 \doteq_{\mathsf{F}} \overline{x}\vartheta_2\varphi_2$, because t and $f(\tilde{t}')$ are two different ground terms with different heads.

From disjointness of ϑ_1 and ϑ_2 with respect to $\mathcal{V}(\Gamma)$ we obtain disjointness of $\sigma\vartheta_1$ and $\sigma\vartheta_2$: Since both $\overline{x} \in \mathcal{V}(\Gamma)$ and $\overline{x} \in \mathcal{V}(\Delta)$ hold, we have either $\overline{x} \notin \mathit{dom}(\sigma)$ or $\overline{x}\sigma = (\tilde{r}, \overline{x})$ for some $\tilde{r}$. In either case we get that there are no φ_1 and φ_2 such that $\overline{x}\sigma\vartheta_1\varphi_1 \doteq_{\mathsf{F}} \overline{x}\sigma\vartheta_2\varphi_2$.

We can proceed in a similar way for any pair of local substitutions generated by the rules in $\mathfrak{R}$. (For local substitutions generated by the same rule the argument that guarantees disjointness is based on the fact that two flattened ground terms with different number of arguments are not equal modulo flatness.) It implies that any pair of substitutions in $comp_{\mathfrak{F}}(\Gamma)$ is disjoint with respect to $\mathcal{V}(\Gamma)$ and, hence, $comp_{\mathfrak{F}}(\Gamma)$ is minimal.

□

Hence, every solvable flat matching problem has a minimal complete set of matchers. As we have seen in Example 1, for some problems this set can be infinite. It implies that flat matching is infinitary. Any complete procedure for flat matching will be nonterminating. The procedure $\mathfrak{F}$ enumerates the minimal complete set of solutions and terminates if the set is finite.

Now we identify a class of flat matching problems on which $\mathfrak{F}$ terminates, i.e., a class with a finite minimal complete set of flat matchers. The first result restricts such a class to flat matching problems without sequence variables.

Lemma 7. *The procedure $\mathfrak{F}$ terminates on a flat matching problem Γ that does not contain sequence variables.*

Proof. We introduce a complexity measure for a flat matching problem Γ as a pair $\langle n, m\rangle$ where n is the number of distinct variables in Γ and m is the multiset of sizes in the ground sides of Γ. Measures are ordered lexicographically. The ordering is, obviously, well-founded. Since the rules in $\mathfrak{R}$ do not introduce new sequence variables, the rules SVP, SVW, and SVW-FH will not be used in any derivation in $\mathfrak{F}$ that starts from Γ. The other rules strictly decrease the complexity measure as Table 1 shows, which implies that any such derivation terminates. The sign $\not\uparrow$ means the component does not increase, $\downarrow$ means it strictly decreases.

Rule	n	m
S, FVE, IVE, IVE-FH	$\downarrow$	
T, Dec	$\not\uparrow$	$\downarrow$

Table 1. Behavior on the complexity measure of the rules that do not affect sequence variables.

□

Now we enlarge the class of problems in Lemma 7. First, we introduce two new notions: A sequence variable $\overline{x}$ is called *bounded* in a flat matching problem Γ if it occurs in a subterm of Γ of the form $g(\tilde{t}_1, \overline{x}, \tilde{t}_2)$, where g is a free function symbol. We say that Γ is *bounded* if all sequence variables occurring in Γ are bounded in Γ.[1] For instance, the problem in Example 3 above is bounded.

The *minimum size* of a term t, denoted by $minsize(t)$, is the number of symbols different from variables in t. We associate to each flat matching problem $\Gamma = \{s_1 \ll^?_{\mathsf{F}} t_1, \ldots, s_n \ll^?_{\mathsf{F}} t_n\}$ the number $dif(\Gamma) = \max(\sum_{i=1}^{n}(size(t_i) - minsize(s_i)), -1)$. Obviously, if $dif(\Gamma) = -1$ then Γ is unsolvable.

Now we can weaken the condition in lemma 7:

[1] Usage of the term *bounded* is motivated by the fact that each bounded variable can be equivalently replaced by a sequence of individual variables whose length is bounded by the size of the ground side of the corresponding matching equation.

Lemma 8. *The procedure $\mathfrak{F}$ terminates on a bounded flat matching problem Γ.*

Proof. We define a complexity measure for a flat matching problem Δ as a triple $\langle n, m, k\rangle$, where n is the number of distinct variables in Δ, m is the multiset of sizes in the ground sides of Δ, and $k = \mathit{dif}(\Delta)$. Complexity measures are ordered lexicographically. In Table 2 one can see that each rule strictly decreases the complexity measure for bounded matching problems. A remark about SVW-FH is in order here: Recall that in a bounded problem, each sequence variable occurs as an argument of a term with the free head (it may occur also under a flat head). Let $\Delta = \{s_1 \ll_{\mathsf{F}}^{?} t_1, \ldots, s_n \ll_{\mathsf{F}}^{?} t_n\}$. Then each application of SVW-FH strictly increases the minimal size of at least one of the s_i's (namely, of those that contain under a free function symbol the sequence variable that the SVW-FH binds), while the minimal sizes of the other s's, in general, do not decrease and the sizes of t's, in general, do not increase. It guarantees that if Φ is obtained from Δ by SVW-FH, then $\mathit{dif}(\Delta) > \mathit{dif}(\Phi)$.

Rule	n	m	k
S, FVE, IVE, SVP, IVE-FH	$\downarrow$		
T, Dec, SVW	$\not\nearrow$	$\downarrow$	
SVW-FH	$\not\nearrow$	$\not\nearrow$	$\downarrow$

Table 2. Rules on the complexity measure of bounded flat matching problems.

The rules in $\mathfrak{R}$ preserve boundedness. Unsolvable problems are immediately transformed to $\bot$. That means that the rule SVW-FH will not apply to a Δ if $\mathit{dif}(\Delta) = -1$. The ordering on complexity measures of solvable problems is well-founded. Hence, for bounded problems no derivation in $\mathfrak{F}$ can continue infinitely.
$\square$

An interesting consequence of Lemma 8 is that linearity, in general, is not an advantage in flat matching. A term or a matching problem is called *linear* if no variable occurs more than once in it. In many equational theories matching is much easier for linear problems (e.g., linear context matching is in P [33] while context matching is NP-complete [32]), but not in flat theories. Here linear sequence variables can be quite a serious disadvantage: If a sequence variable occurs only under a flat function symbol, any complete matching procedure may run forever enumerating the infinite minimal complete set of matchers.

Now we approach the termination problem from a different side. Instead of restricting the class of problems to ensure termination of the procedure, we restrict the procedure itself, sacrificing its completeness, to achieve termination for arbitrary flat matching problems.

Examines the rules in $\mathfrak{R}$ carefully, one can observe that the rule SVW-FH, in particular, its instance with the empty $\tilde{t}_1$, is the source of nontermination of $\mathfrak{F}$. Let us split this rule into two new ones:

SVW1-FH: **Sequence Variable Widening 1 under Flat Head**

$$\{f(\overline{x}, \tilde{s}) \ll_{\mathsf{F}}^{?} f(\tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\overline{x}, \tilde{s}\vartheta) \ll_{\mathsf{F}}^{?} f(\tilde{t})\} \cup \Gamma\vartheta;\ \sigma\vartheta$$

where f is flat and $\vartheta = \{\overline{x} \mapsto (f(), \overline{x})\}$.

SVW2-FH: **Sequence Variable Widening 2 under Flat Head**

$$\{f(\overline{x}, \tilde{s}) \ll^?_{\mathsf{F}} f(t, \tilde{t}_1, \tilde{t}_2)\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\overline{x}, \tilde{s}\vartheta) \ll^?_{\mathsf{F}} f(\tilde{t}_2)\} \cup \Gamma\vartheta;\ \sigma\vartheta$$

where f is flat and $\vartheta = \{\overline{x} \mapsto (f(t, \tilde{t}_1), \overline{x})\}$.

SVW1-FH is the instance of SVW-FH when $\tilde{t}_1 = ()$, and SVW2-FH corresponds to the instance with $\tilde{t}_1 \neq ()$. We denote by $\mathfrak{R}_{\mathrm{NE}}$ the set of rules $\mathfrak{R} \setminus \{\mathsf{SVW\text{-}FH}\} \cup \{\mathsf{SVW2\text{-}FH}\}$. (NE stands for nonempty and is motivated by the way how SVW2-FH is obtained from SVW-FH.) The procedure $\mathfrak{F}_{\mathrm{NE}}$ is obtained from $\mathfrak{F}$ by replacing in its definition the set $\mathfrak{R}$ by $\mathfrak{R}_{\mathrm{NE}}$. Then $\mathfrak{F}_{\mathrm{NE}}$ is obviously incomplete, e.g., it can not solve the problem in Example 3. But it is terminating, and we can prove it easily:

Theorem 9. *The procedure $\mathfrak{F}_{\mathrm{NE}}$ is terminating.*

Proof. Let us define the complexity measure in the same way as in the proof of Lemma 7. Table 3 shows that all the rules in $\mathfrak{R}_{\mathrm{NE}}$ strictly decrease it, which implies that $\mathfrak{F}_{\mathrm{NE}}$ is terminating. □

Rule	n	m
S, FVE, IVE, SVP, IVE-FH	$\downarrow$	
T, Dec, SVW, SVW2-FH	$\not\uparrow$	$\downarrow$

Table 3. Rules in $\mathfrak{R}_{\mathrm{NE}}$ on the complexity measure.

$\mathfrak{F}_{\mathrm{NE}}$ is quite a natural restriction of $\mathfrak{F}$ that gives a terminating matching procedure for flat theories. As we will see in the next section by inspecting the results of flat matching of Mathematica, it corresponds to a further restriction of $\mathfrak{F}_{\mathrm{NE}}$.

4. Flat Matching in Mathematica

Matching with the `Flat` attribute implemented in Mathematica, to the best of our knowledge, is not described in the literature formally. Matching is used in many places and by different functions in the system. Examples given in this section are based on the behavior of Mathematica functions `MatchQ` and `ReplaceList`.[2] According to [37], function call `MatchQ[`*expr, form*`]` returns `True` if the pattern *form* matches *expr*, and returns `False` otherwise. `ReplaceList[`*expr, rules*`]` attempts to transform the entire expression *expr* by applying a rule or list of rules in all possible ways, and returns a list of the results obtained. The reason why we chose these functions is that `MatchQ` can serve as a test for matchability, and `ReplaceList` can be used to compute all possible matchers. Symbols with flat attributes are declared by `SetAttributes[`*symbol*, `Flat]`. We are interested in those pattern objects of Mathematica that are expressed using blanks and blank sequences because they can be seen as counterparts to our variables: In the pattern `f_[x_,a,y___]` the `f_` corresponds to what we call a function variable, `x_`

[2] All the experiments have been carried out on Linux and Windows versions of Mathematica 5.2 and Mathematica 6.0.

corresponds to an individual variable, and `y___` to a sequence variable. In almost all the examples below we use the conventional notation instead of Mathematica syntax.

To demonstrate differences and similarities between $\mathfrak{F}$, $\mathfrak{F}_{\mathrm{NE}}$, and flat matching in Mathematica, we collected some characteristic examples:

Example 10. Outputs of $\mathfrak{F}$, $\mathfrak{F}_{\mathrm{NE}}$, and Mathematica on selected flat matching problems. The function symbol f is flat. All the other function symbols are free. The substitution of Mathematica is guessed by the instantiations computed by `ReplaceList`.

(1) Problem: $f(x) \ll^?_{\mathsf{F}} f(a)$.
Outputs:

$$\begin{aligned}\mathfrak{F}: &\ \{\{x \mapsto a\}, \{x \mapsto f(a)\}\}.\\ \mathfrak{F}_{\mathrm{NE}}: &\ \{\{x \mapsto a\}, \{x \mapsto f(a)\}\}.\\ \mathit{Mathematica}: &\ \{\{x \mapsto f(a)\}\}.\end{aligned}$$

(2) Problem: $f(\overline{y}) \ll^?_{\mathsf{F}} f(a)$.
Outputs:

$$\begin{aligned}\mathfrak{F}: &\ \{\{\overline{y} \mapsto a\}, \{\overline{y} \mapsto f(a)\}, \{\overline{y} \mapsto (f(), a)\},\\ &\quad \{\overline{y} \mapsto (f(), f(a))\}, \{\overline{y} \mapsto (a, f())\}, \ldots\}.\\ \mathfrak{F}_{\mathrm{NE}}: &\ \{\{\overline{y} \mapsto a\}, \{\overline{y} \mapsto f(a)\}\}.\\ \mathit{Mathematica}: &\ \{\{\overline{y} \mapsto a\}\}.\end{aligned}$$

(3) Problem: $f(x, a) \ll^?_{\mathsf{F}} f(a)$.
Outputs:

$$\begin{aligned}\mathfrak{F}: &\ \{\{x \mapsto f()\}\}.\\ \mathfrak{F}_{\mathrm{NE}}: &\ \{\{x \mapsto f()\}\}.\\ \mathit{Mathematica}: &\ \bot\end{aligned}$$

(4) Problem: $f(\overline{y}, a) \ll^?_{\mathsf{F}} f(a)$.
Outputs:

$$\begin{aligned}\mathfrak{F}: &\ \{\{\overline{y} \mapsto ()\}, \{\overline{y} \mapsto f()\}, \{\overline{y} \mapsto (f(), f())\}, \ldots\}.\\ \mathfrak{F}_{\mathrm{NE}}: &\ \{\{\overline{y} \mapsto ()\}\}.\\ \mathit{Mathematica}: &\ \{\{\overline{y} \mapsto ()\}\}.\end{aligned}$$

(5) Problem: $f(x, y) \ll^?_{\mathsf{F}} f(a, b, c)$.
Outputs:

$$\begin{aligned}\mathfrak{F}: &\ \{\{x \mapsto f(), y \mapsto f(a, b, c)\}, \{x \mapsto a, y \mapsto f(b, c)\},\\ &\quad \{x \mapsto f(a), y \mapsto f(b, c)\}, \{x \mapsto f(a, b), y \mapsto c\},\\ &\quad \{x \mapsto f(a, b), y \mapsto f(c)\}, \{x \mapsto f(a, b, c), y \mapsto f()\}\}\end{aligned}$$

$$\begin{aligned}
\mathfrak{F}_{\mathrm{NE}}: \{&\{x \mapsto f(), y \mapsto f(a,b,c)\}, \{x \mapsto a, y \mapsto f(b,c)\},\\
&\{x \mapsto f(a), y \mapsto f(b,c)\}, \{x \mapsto f(a,b), y \mapsto c\},\\
&\{x \mapsto f(a,b), y \mapsto f(c)\}, \{x \mapsto f(a,b,c), y \mapsto f()\}\}\\
\mathit{Mathematica}: \{&\{x \mapsto f(a), y \mapsto f(b,c)\}, \{x \mapsto f(a,b), y \mapsto f(c)\}\}.
\end{aligned}$$

(6) Problem: $f(x,\overline{y}) \ll^{?}_{\mathsf{F}} f(a,b,c)$.
Outputs:

$$\begin{aligned}
\mathfrak{F}: \{&\{x \mapsto f(), \overline{y} \mapsto (a,b,c)\}, \{x \mapsto f(), \overline{y} \mapsto (f(a),b,c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,f(b),c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,b,f(c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(a,b),c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(a),b,f(c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,f(b,c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto f(a,b,c)\}, \ldots,\\
&\{x \mapsto a, \overline{y} \mapsto (b,c)\}, \{x \mapsto a, \overline{y} \mapsto (f(b),c)\},\\
&\{x \mapsto a, \overline{y} \mapsto (b,f(c))\}, \{x \mapsto a, \overline{y} \mapsto f(b,c)\},\\
&\{x \mapsto f(a), \overline{y} \mapsto (b,c)\}, \{x \mapsto f(a), \overline{y} \mapsto (f(b),c)\},\\
&\{x \mapsto f(a), \overline{y} \mapsto (b,f(c))\}, \{x \mapsto f(a), \overline{y} \mapsto f(b,c)\},\\
&\{x \mapsto f(a,b), \overline{y} \mapsto c\}, \ldots,\\
&\{x \mapsto f(), \overline{y} \mapsto (f(),a,b,c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,f(),b,c)\}, \ldots,\\
&\{x \mapsto f(), \overline{y} \mapsto (a,b,c,f())\}, \ldots,\\
&\{x \mapsto f(a,b,c), \overline{y} \mapsto f()\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(),f(),a,b,c)\}, \ldots,\\
&\{x \mapsto a, \overline{y} \mapsto (f(),b,c)\}, \ldots\}\\
\mathfrak{F}_{\mathrm{NE}}: \{&\{x \mapsto f(), \overline{y} \mapsto (a,b,c)\}, \{x \mapsto f(), \overline{y} \mapsto (f(a),b,c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,f(b),c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,b,f(c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(a,b),c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(a),f(b),c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(a),b,f(c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,f(b,c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto (a,f(b),f(c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto f(a,b,c)\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(a),f(b,c))\},\\
&\{x \mapsto f(), \overline{y} \mapsto (f(a,b),f(c))\},
\end{aligned}$$

$$\begin{aligned}
&\{x \mapsto f(), \overline{y} \mapsto (f(a), f(b), f(c))\},\\
&\{x \mapsto a, \overline{y} \mapsto (b, c)\}, \{x \mapsto a, \overline{y} \mapsto (f(b), c)\},\\
&\{x \mapsto a, \overline{y} \mapsto (b, f(c))\}, \{x \mapsto a, \overline{y} \mapsto f(b, c)\},\\
&\{x \mapsto a, \overline{y} \mapsto (f(b), f(c))\}, \{x \mapsto f(a), \overline{y} \mapsto (b, c)\},\\
&\{x \mapsto f(a), \overline{y} \mapsto (f(b), c)\}, \{x \mapsto f(a), \overline{y} \mapsto (b, f(c))\},\\
&\{x \mapsto f(a), \overline{y} \mapsto f(b, c)\}, \{x \mapsto f(a), \overline{y} \mapsto (f(b), f(c))\},\\
&\{x \mapsto f(a, b), \overline{y} \mapsto c\}, \{x \mapsto f(a, b), \overline{y} \mapsto f(c)\},\\
&\{x \mapsto f(a, b, c), \overline{y} \mapsto ()\}\}.\\
Mathematica :\ &\{\{x \mapsto f(a), \overline{y} \mapsto (b, c)\}, \{x \mapsto f(a, b), \overline{y} \mapsto c\},\\
&\{x \mapsto f(a, b, c), \overline{y} \mapsto ()\}\}.
\end{aligned}$$

(7) Problem: $f(x, g(x)) \ll^?_{\mathsf{F}} f(a, g(a))$.
Outputs:

$$\begin{aligned}
\mathfrak{F} :\ &\{\{x \mapsto a\}\}.\\
\mathfrak{F}_{\mathrm{NE}} :\ &\{\{x \mapsto a\}\}.\\
Mathematica :\ &\bot.
\end{aligned}$$

(8) Problem: $f(\overline{y}, g(\overline{y})) \ll^?_{\mathsf{F}} f(a, g(a))$.
Outputs:

$$\begin{aligned}
\mathfrak{F} :\ &\{\{\overline{y} \mapsto a\}\}.\\
\mathfrak{F}_{\mathrm{NE}} :\ &\{\{\overline{y} \mapsto a\}\}.\\
Mathematica :\ &\{\{\overline{y} \mapsto a\}\}.
\end{aligned}$$

(9) Problem: $f(x, g(x)) \ll^?_{\mathsf{F}} f(a, g(f(a)))$.
Outputs:

$$\begin{aligned}
\mathfrak{F} :\ &\{\{x \mapsto f(a)\}\}.\\
\mathfrak{F}_{\mathrm{NE}} :\ &\{\{x \mapsto f(a)\}\}.\\
Mathematica :\ &\{\{x \mapsto f(a)\}\}.
\end{aligned}$$

(10) Problem: $f(\overline{y}, g(\overline{y})) \ll^?_{\mathsf{F}} f(a, g(f(a)))$.
Outputs:

$$\begin{aligned}
\mathfrak{F} :\ &\{\{\overline{y} \mapsto f(a)\}\}.\\
\mathfrak{F}_{\mathrm{NE}} :\ &\{\{\overline{y} \mapsto f(a)\}\}.\\
Mathematica :\ &\bot.
\end{aligned}$$

(11) Problem: $F(x, g(x)) \ll^?_{\mathsf{F}} f(a, g(a))$.
Outputs:

$$\begin{aligned}\mathfrak{F} &: \{\{F \mapsto f, x \mapsto a\}\}.\\ \mathfrak{F}_{\mathrm{NE}} &: \{\{F \mapsto f, x \mapsto a\}\}.\\ \mathit{Mathematica} &: \{\{F \mapsto f, x \mapsto a\}\}.\end{aligned}$$

(12) Problem: $F(\overline{y}, g(\overline{y})) \ll^?_{\mathsf{F}} f(a, g(a))$.
Outputs:

$$\begin{aligned}\mathfrak{F} &: \{\{F \mapsto f, \overline{y} \mapsto a\}\}.\\ \mathfrak{F}_{\mathrm{NE}} &: \{\{F \mapsto f, \overline{y} \mapsto a\}\}.\\ \mathit{Mathematica} &: \{\{F \mapsto f, \overline{y} \mapsto a\}\}.\end{aligned}$$

(13) Problem: $F(x, g(x)) \ll^?_{\mathsf{F}} f(a, g(f(a)))$.
Outputs:

$$\begin{aligned}\mathfrak{F} &: \{\{F \mapsto f, x \mapsto f(a)\}\}.\\ \mathfrak{F}_{\mathrm{NE}} &: \{\{F \mapsto f, x \mapsto f(a)\}\}.\\ \mathit{Mathematica} &: \bot.\end{aligned}$$

(14) Problem: $F(\overline{y}, g(\overline{y})) \ll^?_{\mathsf{F}} f(a, g(f(a)))$.
Outputs:

$$\begin{aligned}\mathfrak{F} &: \{\{F \mapsto f, \overline{y} \mapsto f(a)\}\}.\\ \mathfrak{F}_{\mathrm{NE}} &: \{\{F \mapsto f, \overline{y} \mapsto f(a)\}\}.\\ \mathit{Mathematica} &: \bot.\end{aligned}$$

Analyzing these examples leads to the following observations:

(1) When an individual variable x matches a single argument a in a flat function f, Mathematica returns $x \mapsto f(a)$,[3] while $\mathfrak{F}$ and $\mathfrak{F}_{\mathrm{NE}}$ compute two bindings each: $x \mapsto a$ and $x \mapsto f(a)$.

(2) When a variable v occurs as an argument in a flat function f, Mathematica does not bind v with $f()$.

(3) When a sequence variable $\overline{x}$ matches a sequence $(s_1, \ldots, s_n)$ under a flat function f, Mathematica returns $\overline{x} \mapsto (s_1, \ldots, s_n)$. $\mathfrak{F}$ computes infinitely many bindings obtained from $\overline{x} \mapsto (s_1, \ldots, s_n)$ by putting some of the s's under f and inserting sequences of $f()$'s in between of sequence elements. $\mathfrak{F}_{\mathrm{NE}}$ computes finitely many bindings obtained from $\overline{x} \mapsto (s_1, \ldots, s_n)$ by putting some of the s's under f.

(4) When a term $F(\tilde{s})$ matches a term $f(\tilde{t})$, where f is flat, Mathematica binds F with f and continues like solving the matching problem $h(\tilde{s})\{F \mapsto f\} \ll^?_{\mathsf{F}} h(\tilde{t})$ where h is a fresh free function symbol.[4] $\mathfrak{F}$ and $\mathfrak{F}_{\mathrm{NE}}$ continue with solving the $f(\tilde{s})\{F \mapsto f\} \ll^?_{\mathsf{F}} f(\tilde{t})$. This observation comes from the problems 7-10 and 11-14.

[3] This is also explained in [37].

[4] We do not know what the motivation behind such a behavior is. We could not find any discussion on this feature.

Comparing 7 and 11, one can see that in 11, after mapping F to f, if Mathematica continued with solving $f(x, g(x)) \ll^?_{\mathsf{F}} f(a, g(a))$ then it would have failed because 7 fails on that. Also, comparing 9 and 13 justifies the observation, since otherwise Mathematica would have returned the same substitution for 9 and for 13. On the other hand, there is no difference between the answers on the problems 8 and 12, and between the answers on 10 and 14, because Mathematica behaves in the same way on those problems, no matter whether the head of the terms to be matched is flat or free.

Now we try to model the observed behavior of Mathematica. It requires to introduce a couple of new rules that will replace their old counterparts:

S-Mma: **Solve Rule in Mathematica**

$\{x \ll^?_{\mathsf{F}} t\} \cup \Gamma;\ \sigma \Longrightarrow \Gamma\vartheta;\ \sigma\vartheta,$

where Γ does not contain an equation $f(\tilde{s}) \ll^?_{\mathsf{F}} r$ with $x \in \mathcal{V}(\tilde{s})$ and $\vartheta = \{x \mapsto t\}$.

FVE-Mma: **Function Variable Elimination in Mathematica**

$\{F(\tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{g(\tilde{s}\vartheta) \ll^?_{\mathsf{F}} g(\tilde{t})\} \cup \Gamma\vartheta;\ \sigma\vartheta,$

where $\vartheta = \{F \mapsto f\}$ and g is a fresh free function symbol.

IVE-Mma: **Individual Variable Elimination in Mathematica**

$\{f(x, \tilde{s}) \ll^?_{\mathsf{F}} f(t, \tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \{x \ll^?_{\mathsf{F}} t,\ f(\tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t})\} \cup \Gamma;\ \sigma$ where f is not flat.

IVE-FH-Mma: **Ind. Var. Elimination under a Flat Head in Mathematica**

$\{f(x, \tilde{s}) \ll^?_{\mathsf{F}} f(t, \tilde{t}_1, \tilde{t}_2)\} \cup \Gamma;\ \sigma \Longrightarrow \{x \ll^?_{\mathsf{F}} f(t, \tilde{t}_1),\ f(\tilde{s}) \ll^?_{\mathsf{F}} f(\tilde{t}_2)\} \cup \Gamma;\ \sigma$

where f is flat.

Soundness and termination of these rules follow from the same properties of their old counterparts. We also introduce explicit failure rules that will help us to get rid of the decidability test in the algorithm:

SC: **Symbol Clash**

$\{f(\tilde{s}) \ll^?_{\mathsf{F}} g(\tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \bot$ if $f \neq g$.

ERS: **Empty Right Side**

$\{f(s, \tilde{s}) \ll^?_{\mathsf{F}} f()\} \cup \Gamma;\ \sigma \Longrightarrow \bot$ if $s \notin \mathcal{V}$.

ELS: **Empty Left Side**

$\{f() \ll^?_{\mathsf{F}} f(t, \tilde{t})\} \cup \Gamma;\ \sigma \Longrightarrow \bot.$

Now, let us denote by $\mathfrak{R}_{\mathrm{Mma}}$ the set of rules T, S-Mma, FVE-Mma, Dec, IVE-Mma, SVP, SVW, IVE-FH-Mma, SC, ERS, and ELS. (Note that the rule SVW-FH is omitted.) Then the algorithm $\mathfrak{F}_{\mathrm{Mma}}$ can be defined similarly to $\mathfrak{F}$, using the rules from $\mathfrak{R}_{\mathrm{Mma}}$ instead of $\mathfrak{R}$ and omitting the solvability check in the nodes of the matching tree. This leads us to the following conjecture, which we can only test on examples but obviously can not prove formally:

Conjecture 1. For flat matching problems with individual, function, and sequence variables, the algorithm $\mathfrak{F}_{\text{Mma}}$ models the input-output behavior of the flat matching algorithm of Mathematica.

To get more insight into $\mathfrak{F}_{\text{Mma}}$, we bring examples comparing $\mathfrak{F}_{\text{Mma}}$ and $\mathfrak{F}_{\text{NE}}$.

Example 11. We compare the behavior of $\mathfrak{F}_{\text{Mma}}$ and $\mathfrak{F}_{\text{NE}}$ on the matching problems below. The function symbol f is flat.

(1) Problem: $\{f(g(x), x) \ll^?_{\mathsf{F}} f(g(a), a)\}$.

Run of $\mathfrak{F}_{\text{NE}}$:

$$\begin{aligned}
&\{f(g(x), x) \ll^?_{\mathsf{F}} f(g(a), a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{Dec}} \quad \{g(x) \ll^?_{\mathsf{F}} g(a),\ f(x) \ll^?_{\mathsf{F}} f(a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{IVE}} \quad \{f(a) \ll^?_{\mathsf{F}} f(a)\};\ \{x \mapsto a\}\\
&\Longrightarrow_{\mathsf{T}} \quad \emptyset;\ \{x \mapsto a\}.
\end{aligned}$$

Run of $\mathfrak{F}_{\text{Mma}}$:

$$\begin{aligned}
&\{f(g(x), x) \ll^?_{\mathsf{F}} f(g(a), a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{Dec}} \quad \{g(x) \ll^?_{\mathsf{F}} g(a),\ f(x) \ll^?_{\mathsf{F}} f(a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{IVE\text{-}Mma}} \quad \{x \ll^?_{\mathsf{F}} a,\ f(x) \ll^?_{\mathsf{F}} f(a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{IVE\text{-}FH\text{-}Mma}} \{x \ll^?_{\mathsf{F}} a,\ x \ll^?_{\mathsf{F}} f(a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{S\text{-}Mma}} \quad \{a \ll^?_{\mathsf{F}} f(a)\};\ \{x \mapsto a\}\\
&\Longrightarrow_{\mathsf{SC}} \quad \bot.
\end{aligned}$$

(2) Problem: $\{F(g(x), x) \ll^?_{\mathsf{F}} f(g(a), a)\}$.

Run of $\mathfrak{F}_{\text{NE}}$:

$$\begin{aligned}
&\{F(g(x), x) \ll^?_{\mathsf{F}} f(g(a), a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{FVE}} \quad \{f(g(x), x) \ll^?_{\mathsf{F}} f(g(a), a)\};\ \{F \mapsto f\}\\
&\Longrightarrow_{\mathsf{Dec}} \quad \{g(x) \ll^?_{\mathsf{F}} g(a),\ f(x) \ll^?_{\mathsf{F}} f(a)\};\ \{F \mapsto f\}\\
&\Longrightarrow_{\mathsf{IVE}} \quad \{f(a) \ll^?_{\mathsf{F}} f(a)\};\ \{F \mapsto f, x \mapsto a\}\\
&\Longrightarrow_{\mathsf{T}} \quad \emptyset;\ \{F \mapsto f, x \mapsto a\}.
\end{aligned}$$

Run of $\mathfrak{F}_{\text{Mma}}$:

$$\begin{aligned}
&\{F(g(x), x) \ll^?_{\mathsf{F}} f(g(a), a)\};\ \varepsilon\\
&\Longrightarrow_{\mathsf{FVE\text{-}Mma}} \quad \{h(g(x), x) \ll^?_{\mathsf{F}} h(g(a), a)\};\ \{F \mapsto f\}\\
&\Longrightarrow_{\mathsf{Dec}} \quad \{g(x) \ll^?_{\mathsf{F}} g(a),\ h(x) \ll^?_{\mathsf{F}} h(a)\};\ \{F \mapsto f\}\\
&\Longrightarrow_{\mathsf{IVE\text{-}Mma}} \quad \{x \ll^?_{\mathsf{F}} a,\ h(x) \ll^?_{\mathsf{F}} h(a)\};\ \{F \mapsto f\}\\
&\Longrightarrow_{\mathsf{IVE\text{-}Mma}} \quad \{x \ll^?_{\mathsf{F}} a\};\ \{F \mapsto f\}\\
&\Longrightarrow_{\mathsf{S\text{-}Mma}} \quad \emptyset;\ \{F \mapsto f, x \mapsto a\}.
\end{aligned}$$

Note that Mathematica can verify that each solution computed by $\mathfrak{F}$ is correct, e.g., it sees $f(x, g(x))\{x \mapsto a\}$ and $f(a, g(a))$, where f is flat, as identical expressions, although the Mathematica matching algorithm can not compute the substitution $\{x \mapsto a\}$ that matches $f(x, g(x))$ to $f(a, g(a))$.

One of confusing examples for novice Mathematica programmers is the behavior of the system on evaluating `f[a]`, where `f` has the attribute `Flat` and the rule for `f` is defined as `f[x_]:=x`. It exceeds the iteration limit that indicates getting into an infinite loop. This behavior has an easy explanation based on $\mathfrak{F}_{\text{Mma}}$: Matching `f[x_]` to `f[a]` results (according to the rule IVE-FH-Mma) to instantiating `x` with `f[a]`. Hence, the rule rewrites `f[a]` into `f[a]`. Since Mathematica keeps applying rules to an expression it evaluates until it stops changing, the obtained `f[a]` will be again evaluated, and it gets into an infinite loop.

For the reader who is familiar with patterns in Mathematica, we note that we discussed only patterns of the form *s:obj* where *obj* is either `Blank[]` or `BlankNullSequence[]` pattern object. Other pattern objects as, for instance, `Repeated`, `Condition`, `PatternTest`, `Except`, etc. have not been considered.

We believe that $\mathfrak{F}_{\mathrm{Mma}}$ characterizes flat matching of Mathematica more formally than the explanations one can find in the literature. Moreover, it is more complete: The case with function variables matching flat functions, which we formalize in the rule FVE-Mma and which shows quite a nonstandard behavior from the theoretical point of view, is not mentioned in the documentation.

We implemented the algorithms $\mathfrak{F}$, $\mathfrak{F}_{\mathrm{NE}}$, and $\mathfrak{F}_{\mathrm{Mma}}$ in Mathematica. The code together with the problems from Example 10 can be downloaded from

`http://www.risc.uni-linz.ac.at/people/tkutsia/software.html.`[5]

5. Related Work

Languages with sequence variables, and solving methods for them, have applications in various areas, like automated reasoning [15, 28, 23, 7], logic, functional, and rule-based programming [12, 5, 26, 11, 37, 27], XML querying and processing [10, 24], semantic web [18], program synthesis and transformation [9, 30], artificial intelligence and knowledge engineering [17, 8, 19], just to name a few. We investigated equational unification with sequence variables in [22] where among other theories, the flat and orderless theories have also been studied. Within this framework, expressing Mathematica's free (i.e., without attributes) and orderless pattern matching is straightforward, and other, related problems as word equations [35, 2, 20, 34], associative unification [29], unification for path logics closed under right identity and associativity [31] can be easily modeled.

A review of Mathematica, including its programming capabilities, appeared in [13]. Evaluation of expressions and programs in four computer algebra systems, Mathematica among them, is surveyed in [14]. Largely informal explanations of pattern matching in Mathematica are given in the manual [37] and in other materials on Mathematica programming (see, e.g. [25, 16, 36] as well as the Mathematica user's forum [1]). There have been attempts to give a formal characterization of Mathematica's behavior in different contexts. For example, [4] provides a semantics justifying the use of infinity in informal limit calculations. Rewriting part of the Mathematica programming language, Mathematica/R, is characterized in [6].

6. Conclusion

In the first part of the paper we studied flat matching from the unification theory point of view. We introduced a method to solve modulo a flat theory systems of equations built over individual, sequence, and function variables and flexible arity function symbols. The method provides a minimal complete matching procedure for flat theories. Flat matching is infinitary and, hence, there are problems on which the procedure does not stop. The minimal complete set of matchers it enumerates can be infinite. However, there

[5] From the same location one can also download an implementation of flat matching algorithm that computes a finite representation of the minimal complete set of matchers. The representation is based on regular expressions over substitutions.

are terminating cases as well, when the minimal complete set of matchers is finite. We described one of such cases: If a sequence variable occurs in the input problem, then it also occurs as a direct argument of a subterm with the free head.

The second part of the paper can be interesting from a practical point of view. Here instead of restricting the class of input problems, we restricted the procedure to get a terminated (incomplete) algorithm. We showed that the rule that allows to instantiate a sequence variable $\overline{x}$ under a flat symbol f with the sequence $(f(), \overline{x})$ is the source of nontermination in the procedure. Removing it, we obtained a terminating flat matching algorithm. We made further restrictions of the algorithm to finally reach the set of rules that, in our opinion, models formally the flat matching algorithm of Mathematica and gives its precise semantics.

7. Acknowledgements

This work was supported by the European Commission Framework 6 Programme for Integrated Infrastructures Initiatives under the project SCIEnce—Symbolic Computation Infrastructure for Europe (Contract No. 026133).

The author thanks to anonymous referees for useful remarks.

References

[1] The MathGroup Archive. Mathematica User's Forum. Available at `http://forums.wolfram.com/mathgroup/archive/`, 1989–2006.

[2] H. Abdulrab and J.-P. Pécuchet. Solving word equations. *J. Symbolic Computation*, 8(5):499–522, 1990.

[3] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.

[4] M. Beeson and F. Wiedijk. The meaning of infinity in calculus and computer algebra systems. *J. Symb. Comput.*, 39(5):523–538, 2005.

[5] H. Boley. *A Tight, Practical Integration of Relations and Functions*, volume 1712 of *LNAI*. Springer, 1999.

[6] B. Buchberger. Mathematica as a rewrite language. In T. Ida, A. Ohori, and M. Takeichi, editors, *Proc. of the 2nd Fuji Int. Workshop on Functional and Logic Programming*, pages 1–13. World Scientific, 1996.

[7] B. Buchberger, A. Crăciun, T. Jebelean, L. Kovács, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. Theorema: Towards computer-aided mathematical theory exploration. *J. Applied Logic*, 4:470–504, 2006.

[8] H. Chalupsky. Ontomorph: A translation system for symbolic knowledge. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. 7th Int. Conference on Principles of Knowledge Representation and Reasoning, KR 2000*, pages 471–482, 2000.

[9] E. Chasseur and Y. Deville. Logic program schemas, constraints and semi-unification. In *Proc. LOPSTR'97*, volume 1463 of *LNCS*, pages 69–89. Springer, 1998.

[10] J. Coelho and M. Florido. CLP(Flex): Constraint logic programming applied to XML processing. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE. Proc. of Confederated Int. Conferences*, volume 3291 of *LNCS*, pages 1098–1112. Springer, 2004.

[11] J. Coelho and M. Florido. VeriFLog: A constraint logic programming approach to verification of website content. In H.-T. Shen, J. Li, M. Li, J. Ni, and W. Wang, editors, *Advanced Web and Network Technologies, and Applications*, volume 3842 of *LNCS*, pages 148–156. Springer, 2006.

[12] A. Colmerauer. An introduction to Prolog III. *Communications of ACM*, 33(7):69–91, 1990.

[13] R. J. Fateman. A review of Mathematica. *J. Symbolic Computation*, 13(5):545–579, 1992.

[14] R. J. Fateman. Symbolic mathematics system evaluators. In Y. N. Lakshman, editor, *Proc. 7th Int. Symposium on Symbolic and Algebraic Computation, ISSAC'96*, pages 86–94. ACM Press, 1996.

[15] M. L. Ginsberg. The MVL theorem proving system. *SIGART Bull.*, 2(3):57–60, 1991.

[16] A. Hayes. How and why? Mathematica techniques. *Mathematica in Education and Research*, 8(3–4):84–97, 1999.

[17] P. Hayes and C. Menzel. Semantics of Knowledge Interchange Format. http://reliant.teknowledge.com/IJCAI01/HayesMenzel-SKIF-IJCAI2001.pdf, 2001.

[18] I. Horrocks and A. Voronkov. Reasoning support for expressive ontology languages using a theorem prover. In J. Dix and S. J. Hegner, editors, *Proc. 4th International Symposium on Foundations of Information and Knowledge Systems, FoIKS'06*, volume 3861 of *LNCS*, pages 201–218. Springer, 2006.

[19] ISO/IEC. ISO/IEC 24707. Information technology—Common Logic (CL): A framework for a family of logic-based languages. http://standards.iso.org/ittf/PubliclyAvailableStandards/c039175_ISO_IEC_24707_2007(E).zip, 2007.

[20] J. Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, 1990.

[21] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of A. Robinson*, pages 257–321. The MIT Press, Cambridge, Massachusetts, US, 1991.

[22] T. Kutsia. *Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols.* PhD thesis, Johannes Kepler University, Linz, Austria, 2002.

[23] T. Kutsia. Theorem proving with sequence variables and flexible arity symbols. In M. Baaz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning. Proceedings of the 9th International Conference, LPAR'02*, volume 2514 of *LNAI*, pages 278–291. Springer, 14–18 October 2002.

[24] T. Kutsia. Context sequence matching for XML. *Electronic Notes on Theoretical Computer Science*, 157(2):47–65, 2006.

[25] R. Maeder. *Programming in Mathematica.* Addison-Wesley, third edition, 1996.

[26] M. Marin and D. Ţepeneu. Programming with sequence variables: The *Sequentica* package. In P. Mitic, P. Ramsden, and J. Carne, editors, *Challenging the Boundaries of Symbolic Computation. Proc. of 5th Int.* Mathematica *Symposium*, pages 17–24, London, 2003. Imperial College Press.

[27] M. Marin and T. Kutsia. Foundations of the rule-based system RhoLog. *J. Applied Non-Classical Logics*, 16(1–2):151–168, 2006.

[28] L. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[29] G. Plotkin. Building in equational theories. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 7, pages 73–90. Edinburgh University Press, 1972.

[30] J. Richardson and N. E. Fuchs. Development of correct transformation schemata for Prolog programs. In N. E. Fuchs, editor, *Proc. of the 7th Int. Workshop on Logic Program Synthesis and Transformation*, volume 1463 of *LNCS*, pages 263–281. Springer, 1997.

[31] R. Schmidt. *E*-Unification for subsystems of S4. In T. Nipkow, editor, *Proc. of the 9th Int. Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, pages 106–120. Springer, 1998.

[32] M. Schmidt-Schauß and K. U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symbolic Computation*, 33(1):77–122, 2002.

[33] M. Schmidt-Schauß and J. Stuber. The complexity of linear and stratified context matching problems. *Theory of Computing Syst.*, 37(6):717–740, 2004.

[34] K. U. Schulz. Word unification and transformation of generalized equations. *J. Automated Reasoning*, 11(2):149–184, 1993.

[35] J. Siekmann. String unification. Research paper, Essex University, 1975.

[36] M. Trott. *The Mathematica Guidebook: Programming.* Springer, New York, 2004.

[37] S. Wolfram. *The Mathematica Book.* Wolfram Media, 5th edition, 2003.