Primitive Rewriting

Nachum Dershowitz[*]
School of Computer Science
Tel Aviv University
Ramat Aviv 69978, Israel
tel.: +972-3-640-5356
nachum.dershowitz@cs.tau.ac.il

October 6, 2005

Abstract

Undecidability results in rewriting have usually been proved by reduction from undecidable problems of Turing machines or, more recently, from Post's Correspondence Problem. Another natural candidate for proofs regarding term rewriting is Recursion Theory, a direction we promote in this contribution.

We present some undecidability results for "primitive" term rewriting systems, which encode primitive-recursive definitions, in the manner suggested by Klop. We also reprove some undecidability results for orthogonal and non-orthogonal rewriting by applying standard results in recursion theory.

1 Introduction

Indeed, if general recursive function
is the formal equivalent of effective calculability,
its formulation may play a role
in the history of combinatory mathematics
second only to that of the formulation of natural number.
— Emil Post (1944)

A number of models of computation vie for the rôle of "most basic" mechanism for defining effective computations. These include: semi-Thue systems, Markov's normal algorithms, Church's lambda calculus, Schönfinkel's combinatory logic, Turing's "logical computing" machines, and Gödel's recursive func-

[*]This research was supported by the Israel Science Foundation (grant no. 250/05).

tions. Although they operate over different domains (strings, terms, numbers), they are all of equivalent computational power.[1]

First-order term rewriting makes for a very natural symbolic programming paradigm based on subterm replacement, without bound variables or built-in operations. The two most basic properties a rewrite system may possess are termination (a.k.a. strong normalization) and confluence (the famous Church-Rosser property). Variations on these include (weak) normalization, unique normal forms, and ground confluence. For a comprehensive text on rewriting, see the recent volume by the "Terese" group, based in Amsterdam [38].

It comes as no surprise that rewrite systems have the same computational power as the other basic models.[2] Moreover, rewrite systems may be restricted in various ways, including left-linearity, orthogonality, and constructor-basedness, without weakening the model from the point of view of computability.

To quote Klop [31, p. 356]: "As is to be expected, most of the properties of TRSs [term rewriting systems] are undecidable. Consider only TRSs with finite signature and finitely many reduction rules. Then it is undecidable whether confluence holds, and also whether termination holds." Early undecidability results in string and term rewriting were proved by reduction from undecidable problems of Turing machines (e.g. [9, 25]). More recently, Post's Correspondence Problem [47] has been used: for string rewriting by Book [6]; for term rewriting in [28, 36] (see also [14, 15] and [38, Sect. 5.3.3]). The most natural candidate for proofs regarding term rewriting, however, is recursion theory, a direction we promote here.

Recursive function theory is a uniquely suitable candidate for demonstrating, by means of suitable reductions, that various properties of members of classes of rewrite systems are undecidable. Standard works on recursion theory include [41, 45, 48, 53]. The encoding of recursive functions as term-rewriting systems is part of the field's age-old "folklore", and is mentioned by Klop as an exercise in his 1992 survey [32].

This paper present some undecidability results for "primitive" term rewriting systems, which encode primitive-recursive definitions. Primitive rewriting is defined in Sect. 3. Section 4 shows how to also faithfully encode partial-recursive functions. Kleene's computation predicate—which is central to the undecidability results—is coded as a primitive rewrite system in the Appendix, and its properties are discussed in Sect. 5. In Sects. 6 and 7, we reprove (and improve) some undecidability results for orthogonal and non-orthogonal rewriting (see [38, Chap. 5]) by applying standard results in recursion theory. The concluding section lists what we believe to be new by way of sufficient conditions for undecidability obtained in this way.

[1]See [4, 5] for a discussion of problems pertaining to comparisons of computational power of models operating over diverse domains.

[2]Of course, the classical Church-Turing Thesis asserts that these sets of functions are exactly what are mechanistically computable. See [3].

2 Background

A total function f over the natural numbers is *primitive recursive* if it is the constant $\lambda.0$, a projection function $\lambda x_1, \ldots, x_k.x_i$, the successor function $\lambda x.x + 1$, the composition of other primitive-recursive functions, or else is itself definable by primitive recursion of the form:

$$f(n, \ldots, x_i, \ldots) \quad := \quad \begin{cases} g(\ldots, x_i, \ldots) & n = 0 \\ h(f(n-1, \ldots, x_i, \ldots), n-1, \ldots, x_i, \ldots) & \text{otherwise} \end{cases} ,$$

where g and h are already known to be primitive recursive.

A partial function f over the natural numbers is *partial recursive* if it is primitive recursive, or if it can be defined by composition or primitive recursion from other partial-recursive functions, or if it can be defined by minimization (μ-recursion):

$$f(\ldots, x_i, \ldots) \quad := \quad \mu_{n \in \mathbb{N}}\{q(n, \ldots, x_i, \ldots)\} ,$$

where q is a partial-recursive predicate.[3] Recursive functions are computed leftmost-innermost [48, Sect. 1.2], which is the computation rule that goes into an infinite loop whenever any computation rule can (see, e.g., [37]). In other words, the result is always the least-defined partial function possible.[4]

A partial-recursive function is *(general) recursive* if it is total (always defined). It is well-known that the class of general recursive functions coincides with the Turing-computable (total) functions over (encodings of) the naturals.

Kleene's Normal-Form Theorem [48, Thm. 1-X] states that there exist primitive recursive functions U and T^K such that

$$\lambda \bar{x}.\ U(\mu z.T^K(j, \bar{x}, z)) \tag{1}$$

enumerates all the partial-recursive functions ($\bar{x}$ is a sequence of variables). The computation predicate $T^K(j, \bar{a}, z)$ ("Kleene's T"; see [29]), checks whether $z \in \mathbb{N}$ is (a numerical encoding of) a list beginning with the term $f_j(\bar{a})$, with arguments $a_i \in \mathbb{N}$, continuing step-by-step as in a valid computation, and ending with a natural number for the value of $f_j(\bar{a})$; U extracts that number. In modern parlance, we would say that the partial-recursive function

$$\lambda j.\lambda \bar{x}.\ U(\mu z.T^K(j, \bar{x}, z)) \tag{2}$$

is an "interpreter" for partial recursion, analogous to the Universal Turing Machine.

Two basic undecidability results in recursion theory (due to Kleene [29]) follow from the existence of this partial-recursive function:

[3]By *predicate* we mean any function, but with 0 interpreted as false and non-zero (usually 1) signifying truth.

[4]For example, with definitions $\kappa(x) := 1$ and $\omega := \mu\{0\}$, $\kappa(\omega)$ is undefined.

DEF(f,n). Given a definition of a partial-recursive function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ and a natural number $n \in \mathbb{N}$, it is undecidable whether $f(n)$ is defined [48, Thm. 1-VII]. By "definition", we mean here the index of an enumeration or the Gödel number of a program.

TOT(f). Given a definition of a partial-recursive function $f : \mathbb{N} \rightharpoonup \mathbb{N}$, it is undecidable whether f is recursive [48, Thm. 1-VIII]. The TOT problem, like its analogue for Turing machines, is not even semi-decidable.

In the appendix, we define an injection $\sharp : f \mapsto f^\sharp$ from the definition of a partial-recursive function f (that is, from the sequence of compositions, recursions, and minimizations that define f) into the naturals. The rewrite program **TK** given there defines a primitive-recursive function $\mathcal{T}$ such that (restricting to one-argument functions):

$$\mathrm{DEF}(f,n) \qquad \Leftrightarrow \qquad \exists y \in \mathbb{N}.\ \mathcal{T}(f^\sharp, n, y) = 1\ . \tag{3}$$

For $j \in \mathbb{N}$ that do not correspond to any program, $\mathcal{T}(j,n,y) = 0$, for all n and y. This suffices for our purposes.

3 Primitive Rewriting

In [32, Ex. 2.2.9], Klop mentioned that the primitive-recursive functions can be directly programmed as a terminating, orthogonal, constructor-based term-rewriting system, where the two constructors are the constant $\mathsf{0}$ and the unary successor function s. There are collapsing rules

$$\pi_i^k(x_1, \ldots, x_k) \qquad \to \qquad x_i\ ,$$

for each k and i, $1 \le i \le k$, corresponding to projections. All other functions f are defined by rules that are either recursion-free compositions of the following form:

$$f(\ldots, x_i, \ldots) \qquad \to \qquad g(h_1(\ldots, x_i, \ldots), \ldots, h_k(\ldots, x_i, \ldots))\ ,$$

or else primitive recursions of the form

$$\begin{array}{rcl} f(\mathsf{0}, \ldots, x_i, \ldots) & \to & g(\ldots, x_i, \ldots) \\ f(\mathsf{s}(n), \ldots, x_i, \ldots) & \to & h(f(n, \ldots, x_i, \ldots), n, \ldots, x_i, \ldots)\ . \end{array}$$

More conveniently, one can allow arbitrary compositions of previously defined functions on right-hand sides of defining rules. Thus, primitive-recursive functions can be defined either by composition:

$$f(\ldots, x_i, \ldots) \qquad \to \qquad G[\ldots, x_i, \ldots]\ ,$$

or by non-nested recursion over the naturals:

$$\begin{array}{rcl} f(\mathsf{0},\ldots,x_i,\ldots) & \to & G[\ldots,x_i,\ldots] \\ f(\mathsf{s}(n),\ldots,x_i,\ldots) & \to & H[f(n,\ldots,x_i,\ldots),n,\ldots,x_i,\ldots]\ , \end{array}$$

where G and H are "contexts" (terms with holes) built from already-defined functions, and the recursive call and the arguments n and x_i may appear any number of times in H. (One can also allow the recursive decrease to be in any one, fixed position.) Such definitions can be easily deconstructed into a sequence of composed functions, preserving derivability, $\to^+$.

Numbers $n \in \mathbb{N}$ are represented by terms $\widehat{n} = \mathsf{s}^n(\mathsf{0})$ in standard unary successor notation. Let $\widehat{\mathbb{N}} = \{\widehat{n} : n \in \mathbb{N}\}$ be the set of these "tally" numbers. Factorial, for example, is defined as follows (using standard infix notation):

$$\begin{array}{rcl} \mathsf{0}+x & \to & x \\ \mathsf{s}(z)+x & \to & \mathsf{s}(z+x) \\ \mathsf{0}\times x & \to & \mathsf{0} \\ \mathsf{s}(z)\times x & \to & (z\times x)+x \\ \mathsf{0}! & \to & \mathsf{s}(\mathsf{0}) \\ (\mathsf{s}(z))! & \to & \mathsf{s}(z)\times(z!)\ . \end{array}$$

Such *primitive* rewriting systems can be made non-erasing (sometimes called "regular"), in the sense that all variables on the left of a rule also appear on the right, and non-collapsing—no right side just a variable, by using the following primitive functions:

$$\begin{array}{rcl} \iota(\mathsf{0}) & \to & \mathsf{0} \\ \iota(\mathsf{s}(n)) & \to & \mathsf{s}(\iota(n)) \\ \varepsilon(\mathsf{0},n) & \to & \iota(n) \\ \varepsilon(\mathsf{s}(m),n) & \to & \varepsilon(m,n)\ . \end{array}$$

Then the right side x_i of each projection rule π_i^k can be enveloped with calls to ε for each of the irrelevant variables:

$$\pi_i^k(x_1,\ldots,x_k) \quad \to \quad \varepsilon(x_1,\ldots\varepsilon(x_{i-1},\varepsilon(x_{i+1},\ldots,\varepsilon(x_n,x_i)\cdots))\cdots)\ .$$

To reduce the depth of right sides, one can use a sequence of "erasure" rules, instead:

$$\begin{array}{rcll} \pi_1^1(x) & \to & x & \\ \pi_1^k(x_1,\ldots,x_k) & \to & \varepsilon(x_k,\pi_1^{k-1}(x_1,\ldots,x_{k-1})) & k>1 \\ \pi_i^k(x_1,\ldots,x_k) & \to & \varepsilon(x_1,\pi_{i-1}^{k-1}(x_2,\ldots,x_k)) & i,k>1\ . \end{array}$$

So massaged, every primitive rewrite system possesses the following properties:

1. it is terminating;

2. it is what we will call *definitional*, that is,
   (a) orthogonal—left-linear with no critical pairs, hence
   (b) confluent,
   (c) constructor-based—all but the outermost symbols on the left are constructors (either $\mathsf{0}$ or s) or variables,
   (d) constructor complete—every non-constructor variable-free term is reducible,[5]
   (e) non-erasing, and
   (f) non-collapsing;
3. and it is 3-deep (having maximum nesting, on the left and on the right, of 3), with only variables occuring at depth 3 (that is, below at most 2 symbols).

Plaisted [46] noted that every primitive-recursive function written as a rewrite system (as above) is provably terminating with his simple path ordering (of order type ω^{ω^N}). Likewise, they can be shown terminating with a lexicographic or multiset path ordering (see [11]). In the other direction, Hofbauer [23] (taking the exponential termination functions of Iturriaga [27] a few steps further) showed that any rewrite system that can be proved terminating using a recursive path ordering must have primitive-recursive derivation length. For some recent related results, see [1, 8].

4 Partial Rewriting

Algebraic rewriting does not have bound variables, so to simulate general recursion we employ a trick, namely, separate minimization functions for each predicate:

$$\begin{array}{rcl} \mu_q(z, \mathsf{s}(y), \ldots, x_i, \ldots) & \to & z \\ \mu_q(z, \mathsf{0}, \ldots, x_i, \ldots) & \to & \mu_q(\mathsf{s}(z), q(\mathsf{s}(z), \ldots, x_i, \ldots), \ldots, x_i, \ldots) \, , \end{array}$$

where q is a partial-recursive predicate. Better yet, we can let an arbitrary expression serve as test, with any non-zero value signifying truth:

$$\begin{array}{rcl} \mu_Q(z, \mathsf{s}(y), \ldots, x_i, \ldots) & \to & z \\ \mu_Q(z, \mathsf{0}, \ldots, x_i, \ldots) & \to & \mu_Q(\mathsf{s}(z), Q[\mathsf{s}(z), \ldots, x_i, \ldots], \ldots, x_i, \ldots) \, . \end{array}$$

[5]Sufficient completeness (the "no junk" condition) means that every ground (variable-free) term is equal (convertible) to a constructor-only term. In a rewriting context, one normally asks that ground non-constructor terms actually normalize to constructor-only terms (incorporating a degree of ground confluence). Since we already have a separate termination property, we only ask that every ground non-constructor term be rewritable. Combined with termination, this yields the usual sufficient-completeness property. (I am borrowing the term "constructor completeness" from notes by Heinrich Hußmann.)

Then, to compute a function f defined by minimization vis-à-vis Q, we start off with

$$f(\dots, x_i, \dots) \quad \to \quad \mu_q(\mathsf{0}, Q[\mathsf{0}, \dots, x_i, \dots], \dots, x_i, \dots) \ .$$

To avoid introducing spurious cases of nontermination, q (or Q) must be *monotonic*, in the sense that $q(k, \bar{x}) > 0$ implies $q(m, \bar{x}) > 0$ for all $m > k$. Were we to allow non-monotonic predicates q, then the computation of $\mu_q(\widehat{N}, \mathsf{0}, \dots, K_i, \dots)$ might diverge for large N, even as f itself never does, since $q(z, \dots, x_i, \dots)$ may yield false for all but one z.

Luckily, with no loss of generality, any ordinary predicate q' can be recast monotonically as

$$q(n, \dots, x_i, \dots) \quad := \quad \sum_{i=0}^{n} q'(i, \dots, x_i, \dots) \ ,$$

where the sum serves as disjunction, and is primitive recursive when q is. The minima μ_q and $\mu_{q'}$ satisfying the two tests (when starting from 0) are the same.

By extension, such rewrite systems, built from primitive recursion and monotonic minimization, will be called *partial recursive.* When they terminate they are *general recursive.*

For example, natural-number division (which is actually primitive recursive) may be defined as follows:

$$\begin{array}{rcl}
\mathsf{p}(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{p}(\mathsf{s}(z)) & \to & z \\
x \dot{-} \mathsf{0} & \to & x \\
x \dot{-} \mathsf{s}(z) & \to & p(x \dot{-} z) \\
\mu(z, \mathsf{s}(v), x, y) & \to & z \\
\mu(z, \mathsf{0}, x, y) & \to & \mu(\mathsf{s}(z),\ (y \times \mathsf{s}(\mathsf{s}(z))) \dot{-}\ x,\ x,\ y) \\
x \div y & \to & \mu(\mathsf{0}, y \dot{-} x, x, y) \ .
\end{array}$$

Rules for the base case of minimization can be made non-erasing, like we did for projections. That done, a term reduces to a numeral *only* if it is a *ground* term built from the constructors, $\mathsf{0}$and s, and from functions defined according to the above schemata.

Partial-recursive rewrite systems are definitional, and, as such, they are confluent. General-recursive systems are definitional and terminating. These inclusions are summarized in Fig. 1.

It is important to take note of the fact that partial-recursive rewrite systems terminate regardless of strategy (in other words, they are strongly normalizing) if, and only if, they terminate via innermost rewriting, since partial systems are orthogonal [42] ([38, Thm. 4.8.7]).[6] Furthermore, as they are also non-erasing, a partial-recursive rewriting system terminates if, and only if, it is (weakly)

[6]Left-linearity is inessential [16]; the non-overlapping condition can also be weakened [21, 13].

primitive-recursive system

$\cap$

general-recursive system

$\cap$

partial-recursive system

$\cap$

definitional system

Figure 1: Hierarchy of recursive rewriting.

---

normalizing [7] ([38, Thm. 4.8.5]). These observations remain true for questions regarding specific initial terms, too [38, p. 128].

Our rewriting implementation of primitive and partial recursion is sound:

Proposition 1 *For all partial-recursive functions $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$, implemented as described above by a symbol f in rewrite system F,*

$$f(a_1, \ldots, a_k) = n \qquad \Leftrightarrow \qquad \mathsf{f}(\widehat{a_1}, \ldots, \widehat{a_k}) \xrightarrow[F]{!} \widehat{n}\ ,$$

for all $n, a_1, \ldots, a_k \in \mathbb{N}$, where $\widehat{n}$ is the (normal form) term representing the number n and $\rightarrow^!$ is reduction to normal form—using any arbitrary rewriting strategy.

Proof: If $f(a_1, \ldots, a_k) = n$, then leftmost-innermost rewriting with F will mimic the recursive computation and yield $\widehat{n}$. Since orthogonal systems are confluent, $\widehat{n}$ is the only normal form. The strategy does not matter, since, as just pointed out, non-erasing orthogonal systems terminate regardless of strategy for a given term if they normalize using any strategy.[7] $\square$

The undecidability of definedness (DEF) for partial-recursive rewriting follows directly, by a standard diagonalization argument: We have that DEF(F, n), for rewrite system F and $n \in \mathbb{N}$, if $\exists m \in \widehat{\mathbb{N}}.\ \mathsf{f}(\widehat{n}) \rightarrow^!_F m$. Were there a recursive system defining a recursive function D for deciding DEF, then the following system X, with the system for D, would be partial recursive (cf. [54]):

$$\begin{array}{rcl} \neg x & \rightarrow & \mathsf{s}(0) \dot{-} x \\ \mu(z, \mathsf{s}(y), f) & \rightarrow & z \\ \mu(z, 0, f) & \rightarrow & \mu(\mathsf{s}(z), \neg\mathsf{D}(f, f), f) \\ \mathsf{X}(g) & \rightarrow & \mu(0, \neg\mathsf{D}(g, g), g)\ . \end{array}$$

[7]To explicate: The non-erasing version of κ from footnote 4 would be $\kappa(x) \rightarrow \varepsilon(x, \hat{1})$. Since the rules for ε would not apply to the term ω or any of its descendants, all computations of $\kappa(\omega)$ diverge. With the erasing version, on the other hand, $\kappa(\omega) \rightarrow^! \hat{1}$.

But then

$$\begin{array}{llll} \mathsf{X}(X^\sharp) \xrightarrow[X]{!} \mathsf{0} & \Leftrightarrow & \neg\mathsf{D}(X^\sharp, X^\sharp) \xrightarrow[X]{!} \mathsf{s}(\mathsf{0}) & \Leftrightarrow \\ \mathsf{D}(X^\sharp, X^\sharp) \xrightarrow[X]{!} \mathsf{0} & \Leftrightarrow & \forall m \in \widehat{\mathbb{N}}.\ \mathsf{X}(X^\sharp) \not\xrightarrow[X]{!} m\ , & \end{array}$$

a contradiction. The first biconditional derives from the definition of X; the last, from the presumption that D decides DEF.

5 Computations

A primitive rewrite system **TK** for the computation predicate $\mathcal{T}$ (implementing T^K), for functions of any arity, is given in the Appendix. Soundness of this implementation of T^K means the following:

Proposition 2 *For all partial-recursive functions $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$,*

$$f(\ldots, a_i, \ldots) = n \quad \Leftrightarrow \quad \exists y \in \widehat{\mathbb{N}}.\ \mathcal{T}(f^\sharp, \ldots, \widehat{a_i}, \ldots, y) \xrightarrow[\mathbf{TK}]{!} \hat{1} \ \wedge\ \mathcal{U}(y) \xrightarrow[\mathbf{TK}]{!} \widehat{n}$$

and

$$f(\ldots, a_i, \ldots) = \bot \quad \Leftrightarrow \quad \forall y \in \widehat{\mathbb{N}}.\ \mathcal{T}(f^\sharp, \ldots, \widehat{a_i}, \ldots, y) \xrightarrow[\mathbf{TK}]{!} \mathsf{0}\ ,$$

where $\bot$ denotes undefined, $\mathcal{T}$ is the symbol for the computation predicate T^K in rewrite system **TK**, *$\mathcal{U}$ is the symbol for the last-element function, and $f^\sharp$ is the numeral representing the rewrite program for f.*

The monotonic version of predicate $\mathcal{T}$ is the following:

$$\begin{array}{rcl} \mathcal{T}^*(j, \ldots, x_i, \ldots, \mathsf{0}) & \to & \mathcal{T}(j, \ldots, x_i, \ldots, \mathsf{0}) \\ \mathcal{T}^*(j, \ldots, x_i, \ldots, \mathsf{s}(y)) & \to & \mathcal{T}^*(j, \ldots, x_i, \ldots, y) + \mathcal{T}(j, \ldots, x_i, \ldots, \mathsf{s}(y))\ . \end{array}$$

Then, by the Normal Form Theorem, to compute any partial-recursive function f, one can use primitive-recursive $\mathcal{T}^*$ along with the following general-recursive rules:

$$\begin{array}{rcl} \mu(z, \mathsf{s}(y), \ldots, x_i, \ldots) & \to & z \\ \mu(z, \mathsf{0}, \ldots, x_i, \ldots) & \to & \mu(\mathsf{s}(z), \mathcal{T}^*(f^\sharp, \ldots, x_i, \ldots, \mathsf{s}(z)), \ldots, x_i, \ldots) \\ \mathsf{f}(\ldots, x_i, \ldots) & \to & \mathcal{U}(\mu(\mathsf{0}, \mathcal{T}^*(f^\sharp, \ldots, x_i, \ldots, \mathsf{0}), \ldots, x_i, \ldots))\ . \end{array}$$

Call this system (including **TK**, and made non-erasing) $\mathbf{R}_f$.

Proposition 3 *For all partial-recursive functions $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$, implemented by a symbol f in rewrite system $\mathbf{R}_f$,*

$$f(a_1, \ldots, a_k) = n \quad \Leftrightarrow \quad \mathsf{f}(\widehat{a_1}, \ldots, \widehat{a_k}) \xrightarrow[\mathbf{R}_f]{!} \widehat{n}\ .$$

As previously mentioned, it is also undecidable whether a partial-recursive system is actually general-recursive. This TOT problem for rewriting can be shown to be non-semi-decidable by using **TK**, with no need to rely on results for Turing machines or recursive functions.

6 Word Problems

In this and the following section, we restrict attention to properties of unary functions. As a corollary of Proposition 2, we have

Proposition 4

$$\mathrm{DEF}(f,n) \qquad \Leftrightarrow \qquad \exists y \in \widehat{\mathbb{N}}.\ \mathcal{T}(f^\sharp, \widehat{n}, y) \xrightarrow[\mathbf{TK}]{!} \hat{1}\ .$$

Proof: This is due to the fact that **TK** (see Appendix) is designed so as to reduce all (ground) terms headed by $\mathcal{T}$ to either $\mathsf{0}$ (for false) or $\hat{1}$ (for true). □

The (rewrite) matching problem, MATCH, is

$$\mathrm{MATCH}(R,t,N) \qquad := \qquad \exists \sigma.\ t\sigma \xrightarrow[R]{!} N\ ,$$

where R is a rewrite system, t is a term containing variables, N is a ground (that is, variable-free) normal form, and σ is a (ground) substitution.

Theorem 5 *Matching of primitive rewriting is undecidable.*

The proof of undecidability of matching for terminating confluent systems in [22, Cor. 3.11] uses a non-erasing, but overlapping system. The simpler proof in [2], based on the unsolvability of Diophantine equations, uses a system for addition and multiplication of integers that is overlapping, non-left-linear, collapsing, erasing, and non-constructor-based. It has recently been shown that matching (as well as unification) is decidable for confluent systems if no variables on the right appear below the root [40].

Undecidability can be shown from the older recursion theory results—without recourse to the difficult resolution of Hilbert's Tenth Problem, as follows:

Proof: The reduction is from undecidable $\mathrm{DEF}(f,n)$ to the instance $\mathrm{MATCH}(\mathbf{TK}, \mathcal{T}(f^\sharp, \widehat{n}, y), \hat{1})$. We have

$$\begin{aligned}
\mathrm{DEF}(f,n) \qquad &\Leftrightarrow \qquad \exists y \in \mathcal{G}.\ \mathcal{T}(f^\sharp, \widehat{n}, y) \xrightarrow[\mathbf{TK}]{!} \hat{1} \\
&\Leftrightarrow \qquad \exists \sigma.\ \mathcal{T}(f^\sharp, \widehat{n}, y)\sigma \xrightarrow[\mathbf{TK}]{!} \hat{1} \\
&\Leftrightarrow \qquad \mathrm{MATCH}(\mathbf{TK}, \mathcal{T}(f^\sharp, \widehat{n}, y), \hat{1})\ ,
\end{aligned}$$

where $\mathcal{G}$ is the set of ground (variable-free) terms over the vocabulary of **TK**. The first equivalence is Proposition 4, except that we need the fact that **TK** is constructor complete to ensure that any ground y that satisfies $\mathcal{T}$ reduces to a numeral, since it must be a numeral for **TK** to reduce the term to normal form. The second step is simply because y is the sole variable in the initial term. □

It is similarly undecidable if two terms have the same normal form.

The ground confluence problem GCR (for terminating systems) is

$$\mathrm{GCR}(R) \quad := \quad \forall s,t,u \in \mathcal{G}.\ u \xrightarrow[R]{!} s \wedge u \xrightarrow[R]{!} t \Rightarrow s = t\ ,$$

where $\mathcal{G}$ is the set of variable-free terms over the vocabulary of R.

Theorem 6 *Ground confluence of terminating left-linear constructor-based non-erasing non-collapsing constructor-complete rewrite systems is undecidable.*

Ground confluence of terminating systems was shown undecidable in [28], both for terminating string systems (which are left- and right-linear, non-erasing, and non-collapsing) and for terminating left- *or* right-linear systems with right-side depth limited to 2. One can't have orthogonality (absence of critical pairs, in addition to left-linearity) here, since orthogonal systems are confluent.

Proof: The reduction is

$$\mathrm{DEF}(f,n) \quad \Leftrightarrow \quad \neg\mathrm{GCR}(\mathbf{TK} \cup K^n_f)\ ,$$

where K^n_f contains the (non-erasing, non-collapsing) rule

$$\mathcal{T}(f^\sharp, \widehat{n}, y) \quad \to \quad \varepsilon(y, \mathsf{0})\ .$$

Note that this rule overlaps rules of **TK**. If, and only if, $f(n)$ is defined, is there a (ground) numeral $\widehat{y} \in \widehat{\mathbb{N}}$ such that

$$\mathcal{T}(f^\sharp, \widehat{n}, \widehat{y}) \quad \xrightarrow[\mathbf{TK}]{!} \quad \hat{1}\ ,$$

making for two normal forms $(\mathsf{0}, \hat{1})$ for $\mathcal{T}(f^\sharp, \widehat{n}, \widehat{y})$. □

The confluence problem CR is

$$\mathrm{CR}(R) \quad := \quad \forall s,t,u. \quad u \xrightarrow[R]{*} s \wedge u \xrightarrow[R]{*} t \quad \Rightarrow \quad \exists v.\ s \xrightarrow[R]{*} v \wedge t \xrightarrow[R]{*} v\ .$$

Theorem 7 *Confluence of non-overlapping constructor-based non-erasing non-collapsing rewriting is undecidable.*

Undecidability of confluence of nonterminating systems is claimed in [26]: "The property of confluence is undecidable for arbitrary rewriting systems, since a confluence test could be used to decide the equivalence, for instance, of recursive program schemes." Standard proofs (e.g. [38, Thm. 5.2.1]) are based on overlapping, but left-linear, constructions. Since orthogonal systems are always confluent, one can't have both left-linearity and non-overlappingness.[8]

Confluence is known to be undecidable (in fact, not even semi-decidable or co-r.e.) even if the rules are left- and right-linear, constructor-complete, and

[8]Recursive program schemes [38, Def. 3.4.7] are orthogonal, but two schemes together are not.

constructor-based, and all critical pairs obtained from overlaps resolve (i.e. the system is locally, or weakly, confluent) [15, Sect. 4]. The terminating case is decidable, even in the presence of overlapping left sides, by the famous Critical Pair Lemma of Knuth [33]; see [38, Thm. 2.7.16]. It has also recently been shown that confluence is decidable for right-linear systems if no variables appear below depth 1 [18] (extending earlier decidability results [44, 20]).

Proof: We cannot use the same K_f^n as in the previous proof, since its left side overlaps rules of **TK**. Instead let B_f^n be

$$\mathsf{B}(\hat{1}, y) \quad \to \quad \mathsf{s}(\mathsf{B}(\mathcal{T}(f^\sharp, \widehat{n}, y), y)) \ ,$$

which has the property that

$$\mathsf{B}(\hat{1}, \widehat{y}) \quad \xrightarrow{!} \quad \mathsf{s}(\mathsf{B}(\hat{1}, \widehat{y})) \ ,$$

for some numeral $\widehat{y}$, if, and only if, $\mathcal{T}(f^\sharp, \widehat{n}, \widehat{y}) \to^* \hat{1}$. Now

$$\begin{array}{lcl} \mathrm{DEF}(f, n) & \Leftrightarrow & \exists y \in \widehat{\mathbb{N}}.\ \mathcal{T}(f^\sharp, \widehat{n}, y) \xrightarrow[\mathbf{TK}]{!} \hat{1} \\ & \Leftrightarrow & \neg\mathrm{CR}(\mathbf{TK} \cup B_f^n \cup H) \ , \end{array}$$

where non-linear system H is

$$\begin{array}{rcl} \mathsf{H}(x, x) & \to & \varepsilon(\mathsf{x}, \mathsf{a}) \\ \mathsf{H}(\mathsf{s}(x), x) & \to & \varepsilon(\mathsf{x}, \mathsf{c}) \ . \end{array}$$

The rules for B and H are akin to Huet's [24] example of non-terminating non-overlapping non-confluence. So, if $f(n)$ is defined, then $\mathsf{H}(\mathsf{B}(\hat{1}, \widehat{y}), \mathsf{B}(\hat{1}, \widehat{y}))$ rewrites to a term containing a by the first rule of H and to a term containing c in two stages, applying B_f^n, followed by the second H rule. Since there is no other way for a term t to rewrite to $\mathsf{s}(t)$, non-confluence is a perfect indication that $f(n)$ is defined. □

The modular (shared-constructor) confluence problem CR2 is

$$\mathrm{CR2}(R, S) \quad := \quad \mathrm{CR}(R \cup S) \ ,$$

where R and S are confluent systems with only constructors in common.

Since H shares no defined symbols with **TK** or B:

Corollary 8 *Modular confluence of constructor-based rewriting is undecidable.*

That constructor-sharing combinations need not preserve confluence was pointed out in [35] (just consider H together with $\mathsf{b} \to \mathsf{s}(\mathsf{b})$).

7 Halting Problems

The normalizability problem NORM is

$$\mathrm{NORM}(R,t) \qquad := \qquad \exists z.\ t \xrightarrow[R]{!} z\ .$$

Theorem 9 *Normalizability of definitional rewriting is undecidable.*

That normalizability is undecidable for non-constructor-based rewriting is obvious from the rewrite system CL for combinatory logic in Klop's monograph [30, Sect. 2.2].

Proof: DEF(f,n) reduces to NORM$(F, f(\widehat{n}))$, where F is the partial recursive rewrite system for f. This is basically just faithfulness of F, as stated in Proposition 1. Thus, normalizability for partial-recursive rewriting is undecidable, and, *a fortiori*, for arbitrary definitional rewriting. □

For the (weak) normalization problem WN,

$$\mathrm{WN}(R) \qquad := \qquad \forall t.\ \mathrm{NORM}(R,t)\ ,$$

the situation is the same:

Theorem 10 *Normalization of definitional rewriting is not semi-decidable.*

Proof: The reduction is

$$\mathrm{TOT}(f) \qquad \Leftrightarrow \qquad \mathrm{WN}(\mathbf{R}_f)\ .$$

By Proposition 3, f is total if, and only if, $\mathbf{R}_f$ normalizes all terms $\mathsf{f}(\widehat{a})$. So, if f is not total, there is a term that $\mathbf{R}_f$ fails to normalize.

For the other direction, when a term (ground or not) is non-terminating for non-overlapping $\mathbf{R}_f$, there must be—by standard techniques in rewriting [10, 16]—an infinite innermost derivation,

$$\mu(z,\mathsf{0},a) \quad \xrightarrow[\mathbf{R}_f]{} \quad \mu(\mathsf{s}(z), \mathcal{T}^*(f^\sharp, a, \mathsf{s}(z)), a) \quad \xrightarrow[\mathbf{R}_f]{+} \quad \mu(\mathsf{s}(z),\mathsf{0},a) \quad \xrightarrow[\mathbf{R}_f]{+} \quad \cdots\ ,$$

punctuated by instances of the main μ-rule (the only potentially non-terminating rule), all subterms of which are already in normal-form. Considering how $\mathbf{R}_f$ looks, that means that

$$\mathcal{T}^*(f^\sharp, a, \mathsf{s}^i(z)) \quad \xrightarrow[\mathbf{R}_f]{+} \quad 0\ ,$$

for all $i > 0$. Since $\mathbf{R}_f$ is non-erasing and constructor-based, this can only be if the culprits z and a reduce to numerals. Since $\mathcal{T}^*$ is monotonic, we have

$$\mathcal{T}^*(f^\sharp, \widehat{n}, y) \quad \xrightarrow[\mathbf{R}_f]{+} \quad 0\ ,$$

for all $y \in \widehat{\mathbb{N}}$, where $\widehat{n}$ is the normal form of a. In other words, $f(n)$ admits no finite computation.

We needed to use Kleene Normal Form here (that is, $\mathbf{R}_f$ instead of F) to preclude the possibility that f is total, whereas a function g that it uses is not, in which case some term containing g would never terminate. □

The (uniform/strong) termination problem SN is

$$\mathrm{SN}(R) \qquad := \qquad \neg\exists t.\ t \xrightarrow[R]{} \bot\ ,$$

where the postfixed notation $\to \bot$ indicates the existence of a divergent derivation: $t \to_R \bot$ for system R if there are terms $\{t_i\}_i$ such that $t \to_R t_0 \to_R t_1 \to_R \cdots$.

Corollary 11 *Termination of definitional rewriting is not semi-decidable.*

Proof: As mentioned above, non-erasing orthogonal systems like $\mathbf{R}_f$ terminate (SN) if and only if they are normalizing (WN). □

Termination (and normalization) of (overlapping) string rewriting was proved undecidable in a technical report by Huet and Lankford [25], using a Turing-machine construction for the semi-Thue word problem. (See [38, Sect. 5.3.1] for a similar proof; such a reduction was given by Davis in [9]; see also [53].) Lescanne's proof in [36] is left-linear and constructor-based and can be made non-overlapping. It has recently been shown that termination is decidable for right-linear systems if no variables appear on the right below depth 1 [19] (extending earlier decidability results [25, 10]).

The (disjoint) modular termination problem SN2 is

$$\mathrm{SN2}(R, S) \qquad := \qquad \mathrm{SN}(R \cup S)\ ,$$

where R and S are terminating systems with no common function symbols or constants *at all*.

Theorem 12 ([39]) *Modular termination of left-linear rewriting is undecidable.*

Modular termination of left-linear rewriting was found undecidable in [39, 50].[9] Were the systems also locally confluent, their disjoint union would perforce be terminating [52].

Proof: The reduction is

$$\begin{array}{lcl} \mathrm{DEF}(f, n) & \Leftrightarrow & \exists y \in \widehat{\mathbb{N}}.\ \mathcal{T}(f^\sharp, \widehat{n}, y) \xrightarrow[\mathbf{TK}]{!} \hat{1} \\ & \Leftrightarrow & \neg\mathrm{SN}(\mathbf{TK} \cup G_f^n \cup Z) \\ & \Leftrightarrow & \neg\mathrm{SN2}(\mathbf{TK} \cup G_f^n, Z)\ , \end{array}$$

[9]Middeldorp [39, p. 65] credits the author of this paper with simultaneity.

where G_f^n is

$$\begin{array}{rcl} \mathsf{G}(0,x,y) & \to & x \\ \mathsf{G}(\hat{1},x,y) & \to & y \\ \mathsf{G}(\mathcal{T}(f^\sharp,\widehat{n},z),x,y) & \to & x\ , \end{array}$$

and Z (cf. [51]) is

$$\mathsf{Z}(\mathsf{a},\mathsf{b},z) \quad \to \quad \mathsf{Z}(z,z,z)\ .$$

The point here is that Z alone is terminating, but turns nonterminating when combined with (terminating) rules that rewrite some term to both a and b. System G_f^n, though not containing those constants, does just that if, but only if, $\mathcal{T}(f^\sharp,\widehat{n},z)$ reduces to $\hat{1}$, for some computation z. (The first G rule is only there to keep one of the systems sufficiently complete.) □

The undecidability of termination for various forms of hierarchically combined systems [12, 34, 43] follows directly from the strictly hierarchical form of partial-recursive rewriting, whose right-hand sides only refer to previously defined symbols and whose left sides are, by nature, non-overlapping.

8 Conclusion

The relation between the lambda calculus, combinatory logic, and recursion theory is classical. In 1980, Klop [30] forged the link between combinators and rewriting. Here, we have tried to flesh out the "missing" connection, bridging recursive function theory and term rewriting.

As a consequence of this connection, and the tool added to our arsenal, we have made the following small improvements over well-known undecidability results in rewriting:

- Matching is undecidable for convergent (that is, confluent and terminating), left-linear, non-erasing, constructor-based systems, even when they are non-overlapping and sufficiently complete (Theorem 5).
- Matching is undecidable for convergent, sufficiently-complete systems, even when they are left-linear, non-overlapping, constructor-based, non-erasing, and non-collapsing (Theorem 5).
- Confluence is undecidable for non-erasing, non-collapsing, constructor-based systems, even if they are non-overlapping (Theorem 7).
- Ground confluence is undecidable for terminating, left-linear, non-erasing, non-collapsing, sufficiently-complete systems, even if they are constructor-based (Theorem 6).
- Modular (shared-constructor) confluence is undecidable—even for non-erasing, non-collapsing, constructor-based systems (Corollary 8).
- Normalizability is not decidable for orthogonal, constructor-based systems, even if they are constructor complete and non-erasing (Theorem 9).

- Termination and (weak) normalization are not semi-decidable for orthogonal, constructor-based systems, even if they are constructor complete and non-erasing (Theorem 10; Corollary 11).

Acknowledgement

I thank the referees for their critical reading and constructive suggestions.

References

[1] Amir M. Ben-Amram. General size-change termination and lexicographic descent. In Torben Mogensen, David Schmidt, and I. Hal Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, volume 2566 of *Lecture Notes in Computer Science*, pages 3–17. Springer-Verlag, 2002.

[2] Alexander Bockmayr. A note on a canonical theory with undecidable unification and matching problem. *J. Automated Reasoning*, 3(4):379–381, 1987.

[3] Udi Boker and Nachum Dershowitz. A formalization of the Church-Turing Thesis. Available at `http://www.cs.tau.ac.il/~nachum/papers/ChurchTuringThesis.pdf`.

[4] Udi Boker and Nachum Dershowitz. Comparing computational power. *Logic Journal of the IGPL*, 2006. To appear; available at: `http://www.cs.tau.ac.il/~nachum/papers/ComparingComputationalPower.pdf`.

[5] Udi Boker and Nachum Dershowitz. A hypercomputational alien. *J. of Applied Mathematica & Computation*, 2006. To appear; available at `http://www.cs.tau.ac.il/~nachum/papers/HypercomputationalAlien.pdf`.

[6] Ronald V. Book. Thue systems as rewriting systems. *J. Symbolic Computation*, 3(1&2):39–68, February/April 1987.

[7] Alonzo Church. *The Calculi of Lambda Conversion*, volume 6 of *Ann. Mathematics Studies.* Princeton University Press, Princeton, NJ, 1941.

[8] E. Adam Cichon and Elias Tahhan-Bittar. Strictly orthogonal left linear rewrite systems and primitive recursion. *Ann. Pure Appl. Logic*, 108(1–3):79–101, 2001.

[9] Martin Davis. *Computability and Unsolvability.* McGraw-Hill, New York, 1958.

[10] Nachum Dershowitz. Termination of linear rewriting systems (Preliminary version). In *Proceedings of the Eighth International Colloquium on Automata, Languages and Programming (Acre, Israel)*, volume 115 of *Lecture*

Notes in Computer Science, pages 448–458, Berlin, July 1981. European Association of Theoretical Computer Science, Springer-Verlag.

[11] Nachum Dershowitz. Termination of rewriting. *J. Symbolic Computation*, 3(1&2):69–115, February/April 1987. Corrigendum: *4*, 3 (December 1987), 409–410.

[12] Nachum Dershowitz. Hierarchical termination. In N. Dershowitz and N. Lindenstrauss, editors, *Proceedings of the Fourth International Workshop on Conditional and Typed Rewriting Systems (Jerusalem, Israel, July 1994)*, volume 968 of *Lecture Notes in Computer Science*, pages 89–105, Berlin, 1995. Springer-Verlag.

[13] Nachum Dershowitz and Charles Hoot. Natural termination. *Theoretical Computer Science*, 142(2):179–207, May 1995.

[14] Alfons Geser, Aart Middeldorp, Enno Ohlebusch, and Hans Zantema. Relative undecidability in term rewriting: I. The termination hierarchy. *Information and Computation*, 178(1):101–131, 2002.

[15] Alfons Geser, Aart Middeldorp, Enno Ohlebusch, and Hans Zantema. Relative undecidability in term rewriting: II. The confluence hierarchy. *Information and Computation*, 178(1):132–148, 2002.

[16] Oliver Geupel. Overlap closures and termination of term rewriting systems. Technical report, Universität Passau, Passau, West Germany, July 1989.

[17] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. Translated as "On Formally Undecidable Propositions of Principia Mathematica and Related Systems. I", for Basic Books (New York, 1962) and in M. Davis (ed.), *The Undecidable*, Raven Press, Hewlett, NY, 1965. Available at `http://home.ddc.net/ygg/etext/godel/godel3.htm` (viewed September 2005).

[18] Guillem Godoy and Ashish Tiwari. Confluence of shallow right-linear rewrite systems. In L. Ong, editor, *Computer Science Logic, 14th Annual Conf.*, volume 3634 of *LNCS*, pages 541–556. Springer-Verlag, August 2005.

[19] Guillem Godoy and Ashish Tiwari. Termination of rewrite systems with shallow right-linear, collapsing, and right-ground rules. In R. Nieuwenhuis, editor, *20th Intl. Conf. on Automated Deduction*, volume 3632 of *LNCS*, pages 164–176. Springer-Verlag, July 2005.

[20] Guillem Godoy, Ashish Tiwari, and Rakesh Verma. Characterizing confluence by rewrite closure and right ground term rewrite systems. *Applied Algebra on Engineering, Communication and Computer Science*, 15(1):13–36, June 2004.

[21] Bernhard Gramlich. On proving termination by innermost termination. In H. Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA-96, New Brunswick, NJ)*, volume 1103 of *Lecture Notes in Computer Science*, pages 93–107. Springer-Verlag, July 1996.

[22] Stephan Heilbrunner and Steffen Hölldobler. The undecidability of the unification and matching problem for canonical theories. *Acta Informatica*, 24(2):157–171, April 1987.

[23] Dieter Hofbauer and Clemens Lautemann. Termination proofs and the length of derivations (Preliminary version). In N. Dershowitz, editor, *Proceedings of the International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 167–177, Berlin, April 1989. Springer-Verlag.

[24] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. of the Association for Computing Machinery*, 27(4):797–821, October 1980.

[25] Gérard Huet and Dallas S. Lankford. On the uniform halting problem for term rewriting systems. Rapport laboria 283, Institut de Recherche en Informatique et en Automatique, Le Chesnay, France, March 1978.

[26] Gérard Huet and Derek C. Oppen. Equations and rewrite rules: A survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, New York, 1980.

[27] Renato Iturriaga. *Contributions to Mechanical Mathematics*. PhD thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1967.

[28] Deepak Kapur, Paliath Narendran, and Friedrich Otto. On ground confluence of term rewriting systems. *Information and Computation*, 86(1):14–31, May 1990.

[29] Stephen C. Kleene. General recursive functions of natural numbers. *Mathematische Annales*, 112:727–742, 1936.

[30] Jan Willem Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, 1980.

[31] Jan Willem Klop. Term rewriting systems: from Church-Rosser to Knuth-Bendix and beyond. In M. S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming (Warwick, England)*, Lecture Notes in Computer Science, pages 350–369. Springer-Verlag, July 1990.

[32] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–117. Oxford University Press, Oxford, 1992.

[33] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, U. K., 1970. Reprinted in *Automation of Reasoning 2*, Springer-Verlag, Berlin, pp. 342–376 (1983).

[34] Madala R. K. Krishna Rao. Modular proofs for completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151:487–512, 1995.

[35] M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *Theoretical Computer Science*, 103:273–282, 1992.

[36] Pierre Lescanne. On termination of one rule rewrite systems. *Theoretical Computer Science*, 132(1–2):395–401, 1994.

[37] Zohar Manna. *Mathematical Theory of Computation.* McGraw-Hill, New York, 1974.

[38] "Terese" (Marc Bezem, Jan Willem Klop, and Roel de Vrijer, eds.). *Term Rewriting Systems.* Cambridge University Press, 2002.

[39] Aart Middeldorp. *Modular Properties of Term Rewriting Systems.* PhD thesis, Vrije Universiteit, Amsterdam, 1990.

[40] Ichiro Mitsuhashi, Michio Oyamaguchi, Yoshikatsu Ohta, and Toshiyuki Yamada. The joinability and unification problems for confluent semi-constructor TRSs. In Vincent van Oostrom, editor, *Proceedings of the 15th International Conference on Rewriting Techniques and Applications, Aachen, Germany, June 2004*, volume 3091 of *Lecture Notes in Computer Science*, pages 285–300. Springer-Verlag, 2004.

[41] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, 1989.

[42] Michael J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1977.

[43] Enno Ohlebusch. *Modular Properties of Composable Term Rewriting Systems.* PhD thesis, Abteilung Informationstechnik, Universität Bielefeld, Bielefeld, Germany, 1994.

[44] Michio Oyamaguchi. The Church-Rosser property for ground term rewriting systems is decidable. *Theoretical Computer Science*, 49(1):43–79, 1987.

[45] Rózsa Péter. *Recursive Functions.* Academic Press, 1967.

[46] David A. Plaisted. Well-founded orderings for proving termination of systems of rewrite rules. Report R-78-932, Department of Computer Science, University of Illinois, Urbana, IL, July 1978.

[47] Emil L. Post. Recursive unsolvability of a problem of Thue. *J. of Symbolic Logic*, 13:1–11, 1947.

[48] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1966.

[49] Kai Salomaa. Confluence, ground confluence, and termination of monadic term rewriting systems. *Journal of Information Processing and Cybernetics EIK*, 28:279–309, 1992.

[50] Kai Salomaa. On the modularity of decidability of completeness and termination. *Journal of Automata, Languages and Combinatorics*, 1:37–53, 1996.

[51] Yoshihito Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[52] Yoshihito Toyama, Jan Willem Klop, and Hendrik Pieter Barendregt. Termination for direct sums of left-linear complete term rewriting systems. *J. of the Association for Computing Machinery*, 42(6):1275–1304, November 1995.

[53] Ann Yasuhara. *Recursive Function Theory and Logic.* Academic Press, 1971.

[54] Doron Zeilberger. A 2-minute proof of the 2nd-most important theorem of the 2nd millennium. At `http://www.math.rutgers.edu/~zeilberg/mamarim/mamarimhtml/halt.html` (viewed September 2005).

Appendix: The Computation Predicate

Most descriptions of computation predicates are non-algorithmic. (An exception is [41, Chap. I.7].) To fill this lacuna, we give here a complete primitive rewrite program **TK** for T^K.[10]

Lists are encoded as pairs (of pairs) in the customary fashion, due to Gödel:

$$\langle m, n\rangle \mapsto 2^m(2n+1) .$$

The empty list is 0. Lists $\langle x_1, \ldots, x_k\rangle$ are just nested pairs of the form $\langle x_1, \langle x_2, \langle \cdots x_k\rangle \cdots\rangle\rangle$, etc. Terms are encoded as lists. With this encoding,

[10]The rules are also online at `www.cs.tau.ac.il/~nachum/TK.r`, and a faithful coding in Lisp at `www.cs.tau.ac.il/~nachum/TK.l`.

Table 1: Basic arithmetic and logic. Primitive rules for predecessor (p), addition ($+$), natural subtraction ($\dot{-}$), and multiplication ($\times$ or juxtaposition) were given in Sects. 3 and 4 of the text.

$\mathsf{1}$	$\rightarrow$	$\mathsf{s(0)}$	$\mathsf{2}$	$\rightarrow$	$\mathsf{s(1)}$
$\mathsf{3}$	$\rightarrow$	$\mathsf{s(2)}$	$\mathsf{4}$	$\rightarrow$	$\mathsf{s(3)}$
$\mathsf{5}$	$\rightarrow$	$\mathsf{s(4)}$			
2^{0}	$\rightarrow$	$\mathsf{1}$	$2^{\mathsf{s}(n)}$	$\rightarrow$	$2 \cdot 2^n$
$\neg x$	$\rightarrow$	$\mathsf{1} \dot{-}\, x$	$\delta(x)$	$\rightarrow$	$\neg\neg x$
$m > n$	$\rightarrow$	$\delta(m \dot{-}\, n)$	true	$\rightarrow$	$\mathsf{1}$
$x \vee y$	$\rightarrow$	$\delta(x+y)$	$x \wedge y$	$\rightarrow$	xy
$m \neq n$	$\rightarrow$	$(m > n) \vee (n > m)$	$m = n$	$\rightarrow$	$\neg(m \neq n)$
$\mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ z$	$\rightarrow$	$(x > \mathsf{0})y + (x = \mathsf{0})z$			
$\mathsf{div}(\mathsf{0}, m, n)$	$\rightarrow$	$\mathsf{0}$			
$\mathsf{div}(\mathsf{s}(k), m, n)$	$\rightarrow$	$\mathsf{if}\ m + \mathsf{1} > (k + \mathsf{1})n\ \mathsf{then}\ k + \mathsf{1}\ \mathsf{else}\ \mathsf{div}(k, m, n)$			
$m \div n$	$\rightarrow$	$\mathsf{div}(m, m, n)$	$m \nmid n$	$\rightarrow$	$n > (n \div m)m$
$\lg \mathsf{0}$	$\rightarrow$	$\mathsf{0}$			
$\lg \mathsf{s}(n)$	$\rightarrow$	$\mathsf{if}\ \mathsf{2} \nmid (n + \mathsf{1})\ \mathsf{then}\ \lg n\ \mathsf{else}\ \mathsf{1} + \lg n$			

it is easy to show by induction that $\lg \ell$ is an upper bound on the length of a list ℓ ($\lg \mathsf{nil} = \lg \mathsf{0} = \mathsf{0}$), and $1 + \lg \ell$ on its list-nesting depth.

Table 1 recapitulates definitions of the standard logical and arithmetic operations, including binary logarithm ($\lg x := \lfloor \log_2 x \rfloor$)[11] and a conditional if-then-else that evaluates all three of its arguments.

The primitive-recursive Lisp operations are as follows:

$$\begin{array}{rcl}
\mathsf{nil} & \rightarrow & \mathsf{0} \\
x\mathbf{:}y & \rightarrow & 2^x(2y+\mathsf{1}) \\
\mathsf{car}_2(\mathsf{0}, y) & \rightarrow & \mathsf{nil} \\
\mathsf{car}_2(\mathsf{s}(x), y) & \rightarrow & \mathsf{if}\ 2^{x+1} \nmid y\ \mathsf{then}\ \mathsf{car}_2(x, y) \\
 & & \qquad\qquad \mathsf{else}\ \mathsf{car}_2(x, y) + \mathsf{1} \\
\mathsf{car}\ x & \rightarrow & \mathsf{car}_2(\lg x, x) \\
\mathsf{cdr}\ x & \rightarrow & x \div 2^{(\mathsf{car}\ x)+\mathsf{1}}
\end{array}$$

For readability we use a colon for Lisp's list constructor, **cons**. We will need the fact that, as programmed, $\mathsf{car}(\mathsf{nil}) = \mathsf{cdr}(\mathsf{nil}) = \mathsf{nil}$.

[11]The lg notation for $\log_2$ was suggested by Ed Reingold and popularized by Don Knuth.

Other standard list operations are easy:

$$
\begin{array}{rcl}
\mathsf{cadr}\ x & \to & \mathsf{car}(\mathsf{cdr}\ x) \\
\mathsf{cddr}\ x & \to & \mathsf{cdr}(\mathsf{cdr}\ x) \\
\mathsf{nthcdr}(\mathsf{0}, y) & \to & y \\
\mathsf{nthcdr}(\mathsf{s}(n), y) & \to & \mathsf{cdr}(\mathsf{nthcdr}(n, y)) \\
\mathsf{nth}(n, y) & \to & \mathsf{car}(\mathsf{nthcdr}(n, y)) \\
\mathsf{length}_2(\mathsf{0}, y) & \to & y \\
\mathsf{length}_2(\mathsf{s}(x), y) & \to & \mathsf{if}\ \mathsf{nthcdr}(\lg y \dot{-} (x+\mathsf{1}), y) = \mathsf{nil}\ \ \mathsf{then}\ \lg y \dot{-} (x+\mathsf{1}) \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \mathsf{length}_2(x, y) \\
|x| & \to & \mathsf{length}_2(\lg x, x) \\
\mathsf{append}_2(\mathsf{0}, x, y) & \to & y \\
\mathsf{append}_2(\mathsf{s}(n), x, y) & \to & \mathsf{nth}(|x| \dot{-} (n+\mathsf{1}), x)\mathbf{:}\mathsf{append}_2(n, x, y) \\
x * y & \to & \mathsf{append}_2(|x|, x, y)
\end{array}
$$

The function $|x|$ gives the list length of x; we use an asterisk $*$ for the list append function (as did Gödel [17]).

We will have recourse to a few additional functions for lists and terms:

$$
\begin{array}{rcl}
\mathsf{nthcadr}(\mathsf{0}, x) & \to & x \\
\mathsf{nthcadr}(\mathsf{s}(n), x) & \to & \mathsf{cadr}(\mathsf{nthcadr}(n, x)) \\
\mathsf{prefix}(\mathsf{0}, x) & \to & \mathsf{nil} \\
\mathsf{prefix}(\mathsf{s}(n), x) & \to & \mathsf{prefix}(n, x) * (\mathsf{nth}(n, x)\mathbf{:}\mathsf{nil}) \\
\mathsf{pos}_2(\mathsf{0}, p, x) & \to & x \\
\mathsf{pos}_2(\mathsf{s}(n), p, x) & \to & \mathsf{nth}(\mathsf{nth}(n, p), \mathsf{pos}(n, p, x)) \\
\mathsf{pos}(p, x) & \to & \mathsf{pos}_2(|p|, p, x) \\
\mathcal{U}(z) & \to & \mathsf{nth}(|z| \dot{-} \mathsf{1}, z)
\end{array}
$$

Function nthcadr digs down first arguments; pos returns a subterm at a given Dewey decimal position; $\mathcal{U}$ is the last element in a sequence, used in Proposition 2.

Primitive programs are enumerated (into the naturals) in the following manner:

- $\mathsf{0}$ is the constant (function) 0;
- $\mathsf{1}$ is the unary successor function s;
- $\mathsf{1}\mathbf{:}i$ is the ith projection rule π_i for any arity;
- $\mathsf{2}\mathbf{:}g\mathbf{:}\bar{h}$ is the composition $g(\ldots, h_i, \ldots)$ of g with h_i;
- $\mathsf{3}\mathbf{:}g\mathbf{:}h$ is primitive recursion, with base case g and recursive case h;
- $\mathsf{4}\mathbf{:}q$ is minimization over predicate q;
- $\mathsf{5}\mathbf{:}q$ is an auxiliary function for minimization.

A function application $f(x_1, \dots, x_k)$ is encoded as a list $\langle f, x_1, \dots, x_k\rangle$. Accordingly, the successor of numeral n is the pair $\langle 1, n\rangle$, while its predecessor is $\mathsf{cadr}\ n$. So, normal forms look like $\langle 1, \langle 1, \langle \cdots \langle 1, 0\rangle \cdots\rangle\rangle$. The auxiliary function, $\mathsf{5{:}}q$, is for minimization starting from some given lower bound. That is, $(\mathsf{5{:}}q)(k, c, \bar{x})$ is k when c is true; otherwise, it is the smallest $i > k$ such that $q(i, \bar{x})$.

To find the next (leftmost innermost) redex, we use a test nf for normal form (i.e. a numeral) and a function next to find the first non-normal-form in a list:

$$
\begin{array}{rcl}
\mathsf{nf}_2(\mathsf{0}, x) & \to & \mathsf{true} \\
\mathsf{nf}_2(\mathsf{s}(n), x) & \to & (\mathsf{nthcadr}(n, x) = \mathsf{0} \vee \mathsf{car}(\mathsf{nthcadr}(n, x)) = 1) \wedge \mathsf{nf}_2(n, x) \\
\mathsf{nf}(x) & \to & \mathsf{nf}_2(\lg x, x) \\
\mathsf{next}_2(\mathsf{0}, x) & \to & |x| \\
\mathsf{next}_2(\mathsf{s}(n), x) & \to & \mathsf{if}\ \mathsf{nf}(\mathsf{nth}(|x| \mathbin{\dot{-}} (n+1), x))\ \mathsf{then}\ \mathsf{next}_2(n, x) \\
 & & \qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ |x| \mathbin{\dot{-}} (n+1) \\
\mathsf{next}(x) & \to & \mathsf{next}_2(|x|, x) \\
\mathsf{redex}_2(\mathsf{0}, x) & \to & \mathsf{nil} \\
\mathsf{redex}_2(\mathsf{s}(n), x) & \to &
\end{array}
$$

$$
\begin{array}{l}
\mathsf{if}\ \mathsf{next}(\mathsf{cdr}(\mathsf{pos}(\mathsf{redex}_2(n, x), x))) = |\mathsf{cdr}(\mathsf{pos}(\mathsf{redex}_2(n, x), x))| \\
\quad \mathsf{then}\ \mathsf{redex}_2(n, x) \\
\quad \mathsf{else}\ \mathsf{redex}_2(n, x) * (1 + \mathsf{next}(\mathsf{cdr}(\mathsf{pos}(\mathsf{redex}_2(n, x), x))) \mathsf{{:}nil}
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{redex}(x) & \to & \mathsf{redex}_2(1 + \lg x, x)
\end{array}
$$

To apply the function definition at that point, we proceed by case analysis on the different ways of building partial-recursive functions:

$$
\begin{array}{rcl}
\mathsf{succ}(x) & \to & \mathsf{1{:}}x\mathsf{{:}nil} \\
\mathsf{dist}(\mathsf{0}, g, x) & \to & \mathsf{nil} \\
\mathsf{dist}(\mathsf{s}(n), g, x) & \to & (\mathsf{nth}(|g| \mathbin{\dot{-}} (n+1), g)\mathsf{{:}}x)\mathsf{{:}dist}(n, g, x) \\
\mathsf{apply}(f, x) & \to & \mathsf{if}\ \mathsf{car}\ f = \mathsf{0}\ \mathsf{then}\ \mathsf{0} \\
 & & \mathsf{else\ if}\ \mathsf{car}\ f = \mathsf{1}\ \mathsf{then}\ \mathsf{nth}(\mathsf{cdr}\ f, x) \\
 & & \mathsf{else\ if}\ \mathsf{car}\ f = \mathsf{2}\ \mathsf{then}\ \mathsf{cadr}\ f\mathsf{{:}dist}(|\mathsf{cddr}\ f|, \mathsf{cddr}\ f, x) \\
 & & \mathsf{else\ if}\ \mathsf{car}\ f = \mathsf{3} \wedge \mathsf{car}\ x = \mathsf{0}\ \mathsf{then}\ \mathsf{cadr}\ f\mathsf{{:}cdr}\ x \\
 & & \mathsf{else\ if}\ \mathsf{car}\ f = \mathsf{3}\ \mathsf{then}\ \mathsf{cddr}\ f\mathsf{{:}}(f\mathsf{{:}cadr}(\mathsf{car}\ x)\mathsf{{:}cdr}\ x)\mathsf{{:}}x \\
 & & \mathsf{else\ if}\ \mathsf{car}\ f = \mathsf{4}\ \mathsf{then}\ (\mathsf{5{:}cdr}\ f)\mathsf{{:}0{:}}(\mathsf{cdr}\ f\mathsf{{:}0{:}}x)\mathsf{{:}}x \\
 & & \mathsf{else\ if}\ \mathsf{car}\ f = \mathsf{5} \wedge \mathsf{cadr}\ x\ \mathsf{then}\ \mathsf{car}\ x \\
 & & \mathsf{else\ if}\ \mathsf{car}\ f = \mathsf{5} \\
 & & \qquad \mathsf{then}\ f\mathsf{{:}}(\mathsf{cdr}\ f\mathsf{{:}succ}(\mathsf{car}\ x)\mathsf{{:}}x)\mathsf{{:}succ}(\mathsf{car}\ x)\mathsf{{:}}x \\
 & & \mathsf{else}\ f\mathsf{{:}}x
\end{array}
$$

The helper function dist distributes a list of functions over shared arguments.

Finally, to check the validity of a computation, we check that the first element is the function call in question, that each step is an application of one of the above function applications, and that the final element is a numeral:

$$\begin{array}{rcl}
\mathsf{change}(n,x,y) & \to & \mathsf{prefix}(n,x) * (y\mathbf{:}\mathsf{nthcdr}(n+1,x)) \\
\mathsf{term}(0,p,x) & \to & \mathsf{apply}(\mathsf{car}\ x, \mathsf{cdr}\ x) \\
\mathsf{term}(\mathsf{s}(n),p,x) & \to & \mathsf{change}(\mathsf{nth}(|p| \dot{-} (n+1),p),x,\mathsf{term}(n,p,x)) \\
\mathsf{step}(x) & \to & \mathsf{term}(|\mathsf{redex}(x)|,\mathsf{redex}(x),x) \\
\mathsf{steps}(0,z) & \to & \mathsf{true} \\
\mathsf{steps}(\mathsf{s}(n),z) & \to & \mathsf{step}(\mathsf{nth}(n,z)) = \mathsf{nth}(n+1,z) \wedge \mathsf{steps}(n,z) \\
\mathcal{T}(f,x,y) & \to & (\mathsf{car}(y) = f\mathbf{:}x) \wedge \mathsf{steps}(|y| \dot{-} 1,y) \wedge \mathsf{nf}(\mathcal{U}(y)) \\
\mathcal{T}^*(f,x,0) & \to & \mathcal{T}(f,x,0) \\
\mathcal{T}^*(f,x,\mathsf{s}(y)) & \to & \mathcal{T}^*(f,x,y) + \mathcal{T}(f,x,\mathsf{s}(y))
\end{array}$$

The function $\mathsf{change}(n,x,y)$ replaces the nth element x_n in a sequence x with y; term substitutes a reduct for redex at a given position; $\mathcal{T}$ is Kleene's computation predicate; $\mathcal{T}^*$ is its monotonic counterpart.

With the above definitions, but sans the auxiliary functions, **TK** boils down to 42 convergent, orthogonal, constructor-based rules. As explained in the text, the system should be transformed into one that is non-erasing and non-collapsing.

Recursive rules are of nesting depth 3 (with only variables on level three) on the left and on the right, and compositions also have right-depth 3. Were all right sides non-nested and linear (they are not), then confluence and termination would be decidable [49]. In fact, one needs only two rules of left nesting-depth 3 to encode all of partial-recursive rewriting and bring on the undecidability results of the previous sections. To that end, one would systematically replace each non-shallow recursive definition f, other than the predecessor function p (defined in the text), with left-shallow rules

$$\begin{array}{rcl}
f(z,\ldots,x_i,\ldots) & \to & f'(\mathsf{t}(z),\mathsf{p}(z),\ldots,x_i,\ldots) \\
f'(\mathsf{F},z,\ldots,x_i,\ldots) & \to & g(\ldots,x_i,\ldots) \\
f'(\mathsf{T},z,\ldots,x_i,\ldots) & \to & h(f(z,\ldots,x_i,\ldots),z,\ldots,x_i,\ldots)\ ,
\end{array}$$

and then add the following:

$$\begin{array}{rcl}
\mathsf{t}(0) & \to & \mathsf{F} \\
\mathsf{t}(\mathsf{s}(x)) & \to & \mathsf{T}\ .
\end{array}$$