New Lower Bounds for the Expressiveness and the Higher-Order Matching Problem in the Simply Typed Lambda Calculus

Sergei Vorobyov

MPI–I–1999–3–001 July 1999

Author's Address

Sergei Vorobyov: Max-Planck Institut für Informatik, Im Stadtwald, D-66123, Saarbrücken, Germany, `sv@mpi-sb.mpg.de`, `http://www.mpi-sb.mpg.de/~sv`.

Publication Notes

Submitted.

(Publication date of this report: July 27, 1999.)

Acknowledgements

Thanks to everybody in the list of cited papers. Thanks to Harald Ganzinger who was, as usual, my first reader and critic, for a series of important corrections and remarks.

Abstract

The contribution of this paper is three-fold:

1) We analyze expressiveness of the simply typed lambda calculus (STLC) over a single base type, and show how *k-DEXPTIME* computations can be simulated in the order $k+6$ STLC. This gives a double order improvement over the lower bound of (Hillebrand & Kanellakis 1996), reducing *k-DEXPTIME* to the order $2k+3$ STLC.

2) We show that *k-DEXPTIME* is linearly reducible to the *higher-order matching problem* (in STLC) of order $k+7$. Thus, order $k+7$ matching *requires* (lower bound) k-level exponential time. This refines over the best previously known lower bound $2^{\left.2^{\cdot^{\cdot^{2}}}\right\}cn/\log(n)}$ from (Vorobyov 1997), which holds in assumption that orders of types are *unbounded*, but does not imply any nontrivial lower bounds when the order of variables is *fixed*.

3) These results are based on the new simplified and streamlined proof of Statman's famous theorem. Previous proofs in (Statman 1979, Mairson 1992, Vorobyov 1997) were based on a two-step reduction: proving a non-elementary lower bound for Henkin's higher-order theory Ω of propositional types and then encoding it in the STLC. We give a direct generic reduction from *k-DEXPTIME* to the STLC, which is conceptually much simpler, and gives stronger and more informative lower bounds for the fixed-order STLC, in contrast with the previous proofs.

Keywords

Typed lambda calculus, lower complexity bounds, non-elementary recursive problems.

Contents

1 Introduction

The simply typed lambda calculus (STLC) invented by Church is a fundamental formalism underlying different aspects of programming languages semantics. Being one of the oldest, best investigated and understood, it enjoys many prominent and easily stated properties. Still several old and important problems remain open. Here we address two such problems: lower bounds for expressiveness and higher-order matching.

1.1 Lower Bounds for Expressiveness

A famous result due to (Statman 1979) that the problem of deciding whether two simply typed terms reduce to the same normal form is not elementary recursive is well known. Less known is the fact that both most well-known existing proofs (Statman 1979, Mairson 1992) imply only a very poor (although non-elementary) lower bound $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\Big\}{\scriptstyle c\log(\log(\log(n)))}$ for the problem, where $c > 0$ is an (undetermined) constant and n is the length of input; see (Vorobyov 1997). Such a non-elementary lower bound may be ignored as immaterial, because $\log(\log(\log(n)))$ is a very slowly growing function, and the undetermined constant c may be very small, say $1/100$, in which case the value of the stack of twos above does not exceed 2 for all n less than astronomical $2^{2^{2^{100}}}$. Although the above lower bound was recently improved (Vorobyov 1997) to more significant $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\Big\}{\scriptstyle c\log(n)}$ for explicitly and $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\Big\}{\scriptstyle cn}$ for implicitly typed STLC (thus Statman's result does matter!), the situation is still not completely satisfactory. In fact, none of the proofs (Statman 1979, Mairson 1992, Vorobyov 1997) implies any lower bounds for the fragments of STLC of *bounded order*. Say that the order of the term is the maximal order of type assigned to its subterm. What is the lower bound on the equality in STLC$^{\leq k}$ of terms of order up to a fixed k? (Statman 1979, Mairson 1992, Vorobyov 1997) do not give the answer and in fact essentially use unboundedly increasing orders to prove the non-elementary lower bound.

Much stronger lower bounds for STLC$^{\leq k}$ of bounded orders are given by (Hillebrand, Kanellakis & Mairson 1993, Hillebrand & Kanellakis 1994, Hillebrand & Kanellakis 1996): PTIME for STLC$^{\leq 4}$, PSPACE for STLC$^{\leq 5}$, k-*DEXPTIME* for STLC$^{\leq 2k+3}$, and k-*EXPSPACE* for STLC$^{\leq 2k+4}$. Still, these results are not completely satisfactory because to jump one level higher in time (space) exponential hierarchy one pays two orders. We make an improvement by proving the following:

> *k-DEXPTIME is reducible to $STLC^{\leq k+6}$ via length order n. More explicitly: for any problem $P \in k$-DEXPTIME and any x one can construct a term t_x of $STLC^{\leq k+6}$ of size linear in $|x|$ such that t_x converts to a fixed fourth-order normal form s if and only if $x \in P$. Consequently, $STLC^{\leq k+6}$ is k-DEXPTIME hard and requires k-iterated exponential time to decide.*

This result improves over (Statman 1979, Mairson 1992, Hillebrand & Kanellakis 1996, Vorobyov 1997) and is 'optimal', when compared with the upper bounds on the STLC reducibility, modulo possible improvements of the additive constant 7; see (Fortune, Leivant & O'Donnell 1983, Schwichtenberg 1991).

1.2 Lower Bounds for Higher-Order Matching

Huet (1976) raised the following, *today still open*, decidability problem for the STLC, called the *higher-order matching problem* and referred to as the *range question* by Statman (1982):

> *Given a term t of type $\sigma_1 \to \ldots \to \sigma_n \to \tau$ and a term u of type τ (both closed and in normal forms), do there exist terms s_i of types σ_i (for $1 \leq i \leq n$) such that $ts_1 \ldots s_n =_{\beta\eta} u$?*

Huet's conjecture is that the problem is decidable, but the proof is probably hard. Dowek (1994) showed that the third-order matching is decidable. Statman (1982) reduced the higher-order matching to the so-called definability problem: *given an element of a finitely generated type hierarchy, does there exist a closed simply typed λ-term denoting this element?* The positive answer was long known as Plotkin-Statman's conjecture. Loader (1993) settled it in the negative, thus leaving the higher-order problem *open*.

We contributed to the settlement of the higher-order matching problem by proving the following strong lower bound (it is a valid approach to settle strong lower bounds for the open decidability problems; see (Compton & Henson 1990)):

Theorem 1.1 ***(Lower Bound for Higher-Order Matching, Vorobyov (1997)).*** *Any algorithm for the higher-order matching in the STLC should necessarily make a number of steps exceeding*

$$\exp_\infty(cn/\log(n)) = 2^{2^{\cdot^{\cdot^{2}}}} \Big\} \; cn/\log(n)$$

for some constant $c > 0$ and infinitely many inputs of size n. □

It follows immediately that the problem is not elementary recursive and one can ask whether there is any difference, from the practical viewpoint, between an undecidable problem and a problem requiring as much as $\exp_\infty(cn/\log(n))$ time to decide.

The proof of Theorem 1.1 in (Vorobyov 1997) does not imply, however, any nontrivial lower bounds for the higher-order matching of bounded order. The order of an instance of the matching problem is the maximal order of types σ_i's of sought terms s_i's in the instance (see definition above). For fixed bounded orders the lower bounds on matching are only known for order 3, *NP*, (Wolfram 1993), and *NEXPTIME* for order 4 (Wierzbicki 1999). No lower bounds are known for higher orders. In this paper we settle the following generic lower bound for the k-order matching:

> k-*DEXPTIME* is linearly reducible to the *higher-order matching problem* (in STLC) of order $k+7$. Thus, order $k+7$ matching *requires* (lower bound) k-level exponential time.

Although this does not cover the *NEXPTIME* lower bound for matching of order 4 by Wierzbicki (1999) obtained by specialized techniques (which do not seem to generalize for the higher levels of exponential hierarchy), our larger additive constant is explained by the genericity of reduction and uniform representation of inputs by terms of order 4.

1.3 Direct Proof of Statman's Theorem

As a technical contribution of our paper we offer a new direct proof of Statman's theorem. Previous proofs in (Statman 1979, Mairson 1992, Vorobyov 1997) were based on a two-step reduction: proving a non-elementary lower bound for Henkin's higher-order theory Ω of propositional types and then encoding it in the STLC. We give a direct reduction from k-*DEXPTIME* to the STLC, which is conceptually much simpler, and gives stronger and more informative lower bounds for the fixed-order STLC, in contrast with the previous proofs.

2 Preliminaries

We assume the reader is familiar with the basics of the STLC. Our definitions and notation are consistent with Statman (1979), Statman (1982), Schwichtenberg (1982), Fortune et al. (1983), Barendregt (1984), Hindley & Seldin (1986), Schwichtenberg (1991), Mairson (1992), (Wolfram 1993), Troelstra & Schwichtenberg (1996). In particular, (simple) types are defined inductively: ι is a single base type; if σ and τ are types, then $\sigma \to \tau$ is a type; these are all types. In writing types we omit parentheses assuming that $\to$ associates to the right. The *order of the type* is defined by: $ord(\iota) = 1$ for the base type and $ord(\sigma \to \tau) = \max\{1 + ord(\sigma), ord(\tau)\}$. Simply typed terms are represented in Church style, with explicit type annotations in λ-abstractions. Applications associate to the left. The order of an explicitly typed term is the largest order of the type of its subterms. STLC$^{\leq k}$ is a fragment of STLC restricted to terms of orders up to k.

3 Refinement of Statman's Theorem

Technically, the main part of the paper is devoted to the construction of the generic reduction from *k-DEXPTIME* to STLC$^{\leq k+6}$. This constitutes, in familiar terms, the straightforward proof of Statman's theorem. More precisely, we prove the following:

Theorem 3.1 *Given a (description of a) deterministic Turing machine M working in time bounded by the k times iterated exponential function and an input $\bar{s}$ of size n one can construct in time polynomial in n a term t of size $O(n)$ and order $k+6$ such that t converts to some fixed normal form s if and only if M accepts $\bar{s}$. It follows that deciding equality in STLC$^{\leq k+6}$ requires the k-iterated exponential time (lower bound).* □

In the sequel we develop the encoding of the generic Turing machine computation within low-order STLC.

3.1 Types

We start by defining the types essential for our construction.

3.1.1 Type of Projections

For $m \in \omega$ define the *second-order* type:

$$\pi_m \equiv \underbrace{\iota \to \ldots \to \iota}_{m} \to \iota \tag{1}$$

Type π_m is inhabited by m projection functions (for $1 \leq 1 \leq m$):

$$p_i \equiv \lambda x_1 \colon \iota \ldots \lambda x_m \colon \iota \,.\, x_i \tag{2}$$

We will use projections of type π_m (with m to be determined and fixed in the sequel) to encode Turing machine's tape symbols and states.

3.1.2 Lists and List Iteration

For a thorough explanation of lists, list iteration, and numerous highly non-trivial examples the reader is referred to (Mairson 1992, Hillebrand et al. 1993, Hillebrand & Kanellakis 1994, Hillebrand & Kanellakis 1996). We only provide basic explanations for the reader's convenience.

Given a set $\{t_1, t_2, t_3, \ldots, t_k\}$ of simply typed terms of the same type α, consider the term

$$L \equiv \lambda c \colon \alpha \to \tau \to \tau \,.\, \lambda n \colon \tau \,.\, c\, t_1\, (c\, t_2\, (c\, t_3\, \ldots\, (c\, t_{k-1}\, (c\, t_k\, n)))) \tag{3}$$

of type

$$(\alpha \to \tau \to \tau) \to \tau \to \tau \tag{4}$$

for an *arbitrary but a priori fixed simple type* τ. The term L represents a list of terms $[t_1, t_2, t_3, \ldots, t_k]$ of type α, the variables c and n abstract over the list constructors `cons` and `nil`. Lists are useful to implement primitive recursion. The above cited papers give numerous elaborate examples. We also extensively use list iteration and will provide many examples below.

3.1.3 Type of Lists of Projections

Define the *fourth-order* type of *lists of projections* (by fixing α and τ in (4) to be π_m):

$$\pi_m^* \equiv (\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m \tag{5}$$

Given p terms $t_1, \ldots, t_p$, all of type π_m, define the *list* $[t_1, \ldots, t_p]$ of type π_m^* as follows:

$$\lambda c\colon \pi_m \to \pi_m \to \pi_m \,.\, \lambda n\colon \pi_m \,.\, c\, t_1 (c\, t_2 (c\, t_3 (\ldots (c\, t_{p-1} (c\, t_p\, n)) \ldots))) \tag{6}$$

Such lists will be used to encode Turing machine tape configurations.

3.2 Turing Machines

For $k \in \omega$ let k-*DEXPTIME*(cn) denote the class of sets (problems) recognizable by one-tape deterministic Turing machines (DTM) in time bounded by the k times iterated exponential function $\left.2^{2^{\cdot^{\cdot^{2^{cn}}}}}\right\}k+1$, where $c > 0$ is a constant and $n \in \omega$ is the length of input. For example, 1-*DEXPTIME*$(cn) =$ *DTIME*(2^{cn}), 2-*DEXPTIME*$(cn) =$ *DTIME*$(2^{2^{cn}})$, etc.

Given a problem $P \in k$-*DEXPTIME*(cn), let M denote the corresponding DTM recognizing P. Suppose that Γ and Q are M's tape alphabet (containing the end marker \$) and the set of states, and denote by N the cardinality of $\Gamma \cup Q$. Without loss of generality we may suppose that:

1. M given an input $s_1 \ldots s_n \in \Gamma^*$ of length n starts in the initial configuration $\$q_0 s_1 \ldots s_n\$$, where $q_0 \in Q$ is the initial state;

2. M stops either with the configuration $\$q_{a\sqcup\sqcup\cdots\sqcup\sqcup}\$$ (accepts) or $\$q_{r\sqcup\sqcup\cdots\sqcup\sqcup}\$$ (rejects), where q_a, $q_r \in Q$ are two distinguished states, after erasing the tape contents;

3. M extends the tape (by adding blanks) only on the right end of the tape (replacing \$ with $_\sqcup\$$), and never extends it on the left end;

4. in every computation M writes at least 2 symbols on its tape.

These technical conventions greatly simplify the subsequent presentation.

3.2.1 Representing Turing Machine Configurations

Let us describe the representation of a DTM configurations or *instantaneous descriptions* (ID for short). Such an ID keeps the contents of the machine tape together with the position and state of the read/write head. It is convenient for our purposes to represent an ID as a list of type π_m^*, see (5), for an appropriate $m \in \omega$ to be specified later. For example, the initial ID in which the machine observes the leftmost symbol of the input word *abba* will be represented, according to our conventions, as the term

$$\lambda c\colon \pi_m \to \pi_m \to \pi_m \,.\, \lambda n\colon \pi_m \,.$$
$$c\,\lceil \$ \rceil (c\,\lceil q_0 \rceil (c\,\lceil a \rceil (c\,\lceil b \rceil (\ldots (c\,\lceil b \rceil (c\,\lceil a \rceil\,(c\,\lceil \$ \rceil\, n)))\ldots))))$$

or as a short-hand $[\lceil \$ \rceil, \lceil q_0 \rceil, \lceil a \rceil, \lceil b \rceil, \lceil b \rceil, \lceil a \rceil, \lceil \$ \rceil]$, where $\lceil \$ \rceil$, $\lceil q_0 \rceil$, $\lceil a \rceil$, $\lceil b \rceil$ are representations of the tape and state symbols by projections of the type π_m, to be discussed below in detail.

Remark. Thus the DTM input data is always represented uniformly by closed lambda terms of order four; see the *language uniform typed inputs convention* of (Hillebrand & Kanellakis 1996).

3.2.2 Representing Turing Machine Moves

In one move (step) a DTM M transforms its current ID into the next ID according to the transition rules of M. Therefore, given an M we need to represent this transformation as a closed λ-term (of order 5)

$$\mathit{Move}_M \;:\; \pi_m^* \to \pi_m^*$$

The implementation of the term Move_M is quite tricky and we split the construction into several stages. The key idea is to perform a move in two main stages, as explained by the following diagram:

$$\begin{array}{ccccccccccc}
c & \lceil s_1 \rceil & (c & \lceil s_2 \rceil & (c & \lceil s_3 \rceil & \ldots & (c & \lceil s_i \rceil & \ldots))) \\
c & \lceil s_1 s_2 s_3 s_4 \rceil & (c & \lceil s_2 s_3 s_4 s_5 \rceil & (c & \lceil s_3 s_4 s_5 s_6 \rceil & \ldots & (c & \lceil s_i s_{i+1} s_{i+2} s_{i+3} \rceil & \ldots))) \\
c & \lceil s'_2 \rceil & (c & \lceil s'_3 \rceil & (c & \lceil s'_4 \rceil & \ldots & (c & \lceil s'_{i+1} \rceil & \ldots)))
\end{array}$$

The first stage replaces the encoding of each symbol s_i in the list with the encoding of the quadruple $s_i s_{i+1} s_{i+2} s_{i+3}$ (i.e., of s_i together with its three successors in the list; we agree that for the last three symbols s_{n-1}, s_n and \$ in an ID the corresponding quadruples are $\lceil s_{n-1} s_n \$\$ \rceil$, $\lceil s_n \$\$\$ \rceil$ and $\lceil \$\$\$\$ \rceil$ respectively).

The second stage implements a well known trick; see, e.g., (Stockmeyer 1974, pp. 38–39). It is easy to see that the contents of the $i+1$-th tape cell in the next ID is uniquely determined (for a given DTM) by the contents of the i-th, $i+1$-st, $i+2$-nd, and $i+3$-rd cells in the current ID. Indeed, if all s_i, s_{i+1}, s_{i+2} are symbols from Σ, then s'_{i+1} equals s_{i+1}. And if one of s_i, s_{i+1}, s_{i+2} is a state symbol from Q, then s'_{i+1} is uniquely determined from s_i, s_{i+1}, s_{i+2}, s_{i+3}, as the reader can readily check. For example, if $s_i = a$, $s_{i+1} = b$, $s_{i+2} = q$, $s_{i+3} = a$ and M has rule $q, a \to b, q', \mathit{left}$, then $s_{i+1} = q'$.

Therefore, the 'global' move from the current to the succeeding ID is performed as a series of 'local' transformations based on observations of quadruples of adjacent symbols. This is one of the basic encoding tricks underlying our construction.

3.2.3 Encoding Tape and State Symbols

We now explain the encoding of tape and state symbols from the alphabet $\Gamma \cup Q$. The cardinality of this alphabet depends on a DTM M chosen, but becomes a *constant*, once M is *fixed*[1]. From the constant cardinality of the alphabet we must determine and fix the parameter m in the projection type π_m. This will allow us to statically type all the lists and the list transformation terms in the sequel. Indeed, as we explained previously, we need to have enough projections to encode not only the symbols from $\Gamma \cup Q$, but also all possible quadruples of such symbols. Therefore, it suffices to choose m equal to $|\Gamma \cup Q|^4$. We stress that this is a *constant*, once the machine M is fixed *a priori*. Fixing m will allow us to write a fixed λ-term defining a DTM step, independent of the input and its length.

Let us agree that the symbols from $\Gamma \cup Q$ are encoded by the first $|\Gamma \cup Q|$ projection functions, and that the remaining projections are used appropriately to encode tuples, triples, and quadruples of symbols from $\Gamma \cup Q$. We will denote such encoding projections by using appropriate subscripts, e.g., $p_{a,b,c,q}$ to encode $abcq$, $p_{q,a,\$,\$}$ for $qa\$\$$. Not all projections are actually used, some of them, like $p_{q,q,a,a}$, do not make sense.

3.3 Comparing Projections, Booleans, Conditionals

We use the standard encodings for the *second-order* type $\mathsf{Bool} \equiv \iota \to \iota \to \iota$, boolean constants $\mathsf{true}\colon \mathsf{Bool} \equiv \lambda x : \iota \,.\, \lambda y \colon \iota \,.\, x$, $\mathsf{false}\colon \mathsf{Bool} \equiv \lambda x : \iota \,.\, \lambda y \colon \iota \,.\, y$, operations

$$\begin{array}{lcllcl}
\mathsf{and} & : & \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} & \equiv & \lambda u \colon \mathsf{Bool} \,.\, \lambda v \colon \mathsf{Bool} \,.\, \lambda x \colon \iota \,.\, \lambda y \colon \iota \,.\, u(vxy)y, \\
\mathsf{or} & : & \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} & \equiv & \lambda u \colon \mathsf{Bool} \,.\, \lambda v \colon \mathsf{Bool} \,.\, \lambda x \colon \iota \,.\, \lambda y \colon \iota \,.\, ux(vxy), \\
\mathsf{not} & : & \mathsf{Bool} \to \mathsf{Bool} & \equiv & \lambda u \colon \mathsf{Bool} \,.\, \lambda x \colon \iota \,.\, \lambda y \colon \iota \,.\, uyx.
\end{array}$$

We define the equality predicate on projections following (Hillebrand & Kanellakis 1996) as

$$\begin{aligned}
&\mathsf{eq} \colon \pi_m \to \pi_m \to \mathsf{Bool} \equiv \\
&\quad \lambda p \colon \pi_m \,.\, \lambda q \colon \pi_m \,.\, \lambda x \colon \iota \,.\, \lambda y \colon \iota \,. \\
&\qquad p(qx\underbrace{y \ldots y}_{m-1})(qyx\underbrace{y \ldots y}_{m-2}) \ldots (q\underbrace{y \ldots y}_{m-2}xy)(q\underbrace{y \ldots y}_{m-1}x)
\end{aligned} \tag{7}$$

eq applied to two projections $p_i, p_j \colon \pi_m$, converts to true or false depending on whether $i = j$ or $i \neq j$.

[1]Note that we first select a DTM and immediately fix it, so the size of the alphabet is always considered a constant.

3.4 Computing Symbols Pairing

As a preparation step for computing encodings of the quadruples of tape symbols, we first define *pairing*:

$$pair\colon \pi_m \to \pi_m \to \pi_m$$

The intention of this function is to combine encodings in the following way, for any $s, s_1, s_2, s_3, s_4 \in \Gamma \cup Q$:

$$\lceil s \rceil + \lceil s_1 s_2 s_3 s_4 \rceil = \lceil s\, s_1\, s_2\, s_3 \rceil,$$

i.e, the last element in the encoding of the second argument is forgotten, the remaining symbols are shifted right, and the first argument becomes the first in the encoding. We do not need to define pairing of, say, $\lceil s_1 s_2 s_3 s_4 \rceil + \lceil s'_1 s'_2 s'_3 s'_4 \rceil$, when the first operand is not a single symbol, we let it be defined arbitrarily.

By using the encodings of the booleans and equality described previously, one can easily write down the λ-definition of *pair* as a straightforward tedious enumeration of cases (tabulation):

$$
\begin{array}{l}
pair\colon \pi_m \to \pi_m \to \pi_m \equiv \\
\quad \lambda p\colon \pi_m\,.\, \lambda q\colon \pi_m\,.\, \lambda x_1\colon \iota\,.\ \ \ldots\ \ .\lambda x_m\colon \iota\,. \\
\qquad \begin{array}{lll}
\mathsf{if} & \mathsf{eq}\ p\ p_{s_1} & \\
\mathsf{then} & \mathsf{if} & \mathsf{eq}\ q\ p_{s_1 s_1 s_1 s_1} \\
 & \mathsf{then} & x_{s_1 s_1 s_1 s_1} \\
 & \mathsf{else\ if} & \mathsf{eq}\ q\ p_{s_1 s_1 s_1 s_2} \\
 & \mathsf{then} & x_{s_1 s_1 s_1 s_1} \\
 & \quad\ldots & \\
 & \mathsf{else\ if} & \mathsf{eq}\ q\ p_{s_i s_j s_l s_n} \\
 & \mathsf{then} & x_{s_1 s_i s_j s_l} \\
 & \quad\ldots & \\
 & \mathsf{else\ if} & \mathsf{eq}\ q\ p_{s_m s_m s_m s_m} \\
 & \mathsf{then} & x_{s_1 s_m s_m s_m} \\
 & \mathsf{else} & x_{s_1 s_m s_m s_m} \\
\mathsf{else\ if} & \mathsf{eq}\ p\ p_{s_2} & \\
\mathsf{then} & \mathsf{if} & \mathsf{eq}\ q\ p_{s_1 s_1 s_1 s_1} \\
 & \mathsf{then} & x_{s_2 s_1 s_1 s_1} \\
 & \mathsf{else\ if} & \mathsf{eq}\ q\ p_{s_1 s_1 s_1 s_2} \\
 & \mathsf{then} & x_{s_2 s_1 s_1 s_1} \\
 & \quad\ldots & \\
 & \mathsf{else\ if} & \mathsf{eq}\ q\ p_{s_i s_j s_l s_n} \\
 & \mathsf{then} & x_{s_2 s_i s_j s_l}
\end{array}
\end{array}
$$

$$\begin{array}{ll} & \dots \\ \mathsf{else\ if} & \mathsf{eq}\ q\ p_{s_m s_m s_m s_m} \\ \mathsf{then} & x_{s_2 s_m s_m s_m} \\ \mathsf{else} & x_{s_2 s_m s_m s_m} \\ & \dots \end{array}$$

where $\lambda b\colon \mathsf{Bool}\,.\,\lambda x\colon \iota\,.\,\lambda y\colon \iota\,.\,\mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y \equiv ((b)(x)(y))$

3.5 Computing the Encoding of the First Four Symbols in a List

Our first application of the list iteration consists in λ-defining a term, which being applied to a list of symbols (encoded as projections) converts to the encoding of the first four symbols of the list as follows. Applied to

$$\lambda c\,.\,\lambda n\,.\,c\,\lceil s_1\rceil\ (c\,\lceil s_2\rceil\ (c\,\lceil s_3\rceil\ (c\,\lceil s_4\rceil\ \dots\ (c\,\lceil s_i\rceil\ \dots\ (c\,\lceil \$\rceil\ n))))) \qquad (8)$$

it yields $\lceil s_1 s_2 s_3 s_4\rceil$, where $\lceil\ \ \rceil$ is an appropriate encoding of symbols and quadruples of symbols by projections. The definition of this term of *order five* is as follows:

$$\begin{array}{l} \mathit{First\text{-}Four}\colon ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \to \pi_m \equiv \\ \qquad \lambda L\colon (\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m\,.\,L\ \mathit{pair}\ p_{\$\$\$\$} \end{array}$$

It is easy to see that when *First-Four* applies to the list (8), it produces the body of the list, where c is replaced with *pair* and n replaced with $p_{\$\$\$\$}$, i.e.,

$$\mathit{pair}\ \lceil s_1\rceil\ (\mathit{pair}\ \lceil s_2\rceil\ (\mathit{pair}\ \lceil s_3\rceil\ (\mathit{pair}\ \lceil s_4\rceil\ \dots\ (\mathit{pair}\ \lceil s_i\rceil\ \dots\ (\mathit{pair}\ \lceil \$\rceil\ p_{\$\$\$\$})))))$$

Immediate reductions with *pair* afterwards produce the desired result $\lceil s_1 s_2 s_3 s_4\rceil$.

3.6 Computing the List of Quadruples

Our next aim is to define the closed λ-term *List-of-Quadruples*: $\pi_m^* \to \pi_m^*$ (see (5)) of *order five*, which is intended to convert to the second list, when being applied to the first one, as shown below:

$$\begin{array}{llllllllll} c & \lceil s_1\rceil & (c & \lceil s_2\rceil & (c & \lceil s_3\rceil & \dots & (c & \lceil s_i\rceil & \dots))) \\ c & \lceil s_1 s_2 s_3 s_4\rceil & (c & \lceil s_2 s_3 s_4 s_5\rceil & (c & \lceil s_3 s_4 s_5 s_6\rceil & \dots & (c & \lceil s_i s_{i+1} s_{i+2} s_{i+3}\rceil & \dots))) \end{array}$$

Let us define this term as follows:

$$
\begin{aligned}
&\mathit{List\text{-}of\text{-}Quadruples}\colon ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \to \\
&\qquad\qquad\qquad\qquad ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \ \equiv \\
&\quad \lambda R\colon (\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m \,. \\
&\qquad \lambda c\colon \pi_m \to \pi_m \to \pi_m \,.\, \lambda n\colon \pi_m \,. \\
&\qquad\quad R \Big(\lambda r\colon \pi_m \,.\, \lambda T\colon \pi_m \,. \\
&\qquad\qquad c\ (\mathit{pair}\ r\ ((\lambda c\colon \pi_m \to \pi_m \to \pi_m \,.\, \lambda n\colon \pi_m \,.\, T)\ \mathit{pair}\ p_m)\ T\Big)\ n
\end{aligned}
$$

- When *List-of-Quadruples* is applied to a list R, all constructors c in applications $c\ \lceil s_i \rceil\, tail_i$ in its body are replaced with the fuction $\lambda r\colon \pi_m \,.\, \lambda T\colon \pi_m \,.\ \ \ldots$.
- The body of this function then creates the first element of the resulting list $\lceil s_i s_{i+1} s_{i+2} s_{i+3} \rceil$ by using the first element s_i bound to r and the $tail_i$ bound to t and using the *First-Four* described above, and iterates the same transformation along the list.

3.7 Computing a Turing Machine Step

Our next aim consists in λ-defining the *fifth-order* term

$$
\mathit{Step}_M\colon ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \to ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m)
$$

which implements the second part of the ID transformation by a given deterministic Turing machine, i.e.,

$$
\begin{array}{llllllll}
c & \lceil s_1 s_2 s_3 s_4 \rceil & (c & \lceil s_2 s_3 s_4 s_5 \rceil & (c & \lceil s_3 s_4 s_5 s_6 \rceil & \ldots & (c \quad \lceil s_i s_{i+1} s_{i+2} s_{i+3} \rceil \quad \ldots))) \\
c & \quad \lceil s'_2 \rceil & (c & \quad \lceil s'_3 \rceil & (c & \quad \lceil s'_4 \rceil & \ldots & (c \qquad \lceil s'_{i+1} \rceil \qquad\quad \ldots)))
\end{array}
$$

where s'_i in the next ID are uniquely determined by adjacent quadruples of adjacent symbols $s_{i-1} s_i s_{i+1} s_{i+2}$; see Section 3.2.2. Although the very first symbol s'_1 in the next ID remains undetermined, by our convention in Section 3.2, the leftmost tape symbol \$ remains always unchanged, therefore we can subsequently add it in the head of the list explicitly.

The definition of Step_M once again uses the list iteration:

$$
\begin{aligned}
&\mathit{Step}_M\colon ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \to \\
&\qquad\quad ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \equiv \\
&\quad \lambda L\colon (\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m \,. \\
&\qquad \lambda c\colon \pi_m \to \pi_m \to \pi_m \,.\, \lambda n\colon \pi_m \,. \\
&\qquad\quad L \Big(\lambda r\colon \pi_m \,.\, \lambda t\colon \pi_m \,.\, c\ (\mathit{trans}_M\ r)\ t\Big)\ n
\end{aligned}
$$

When applied to a list of quadruples of symbols, $Step_M$ walks down the list and transforms every quadruple encoding $\lceil s_{i-1}s_is_{i+1}s_{i+2}\rceil$ bounded to r into the encoding of $\lceil s_i\rceil'$, by using the function $trans_M$, which describes, for a given DTM M the desired uniquely determined transformation.

For any given DTM M this function $trans_M$ is easy to define by tabulation (finite tedious case analysis). For example, if M has command $q, a \to b, q', L$ (in state q seeing a write b and move left), the λ-definition of $trans_M$ will contain:

$$
\begin{array}{l}
trans_M \colon \pi_m \to \pi_m \equiv \\
\quad \lambda s \colon \pi \,.\, \lambda x_1 \colon \iota \ldots \lambda x_m \colon \iota \,. \\
\qquad\qquad \begin{array}{ll}
 & \ldots \\
\mathsf{if} & \mathsf{eq}\ s\ p_{baqa} \\
\mathsf{then} & p_{q'}\ x_1\ x_2 \ldots x_m \\
 & \ldots
\end{array}
\end{array}
$$

Indeed if we know that in the preceding ID $s_{i-1}s_is_{i+1}s_{i+2} = baqa$, then in the next ID s'_i is uniquely determined as q', encoded by the projection $p_{q'}$ according to our convention.

Finally, the desired *fifth-order* term $Move_M$ describing the full one-step ID transformation is defined by

$$
\begin{array}{l}
Move_M \colon ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \to \\
\qquad\qquad ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \equiv \\
\quad \lambda L \colon (\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m \,. \\
\qquad \lambda c \colon \pi_m \to \pi_m \to \pi_m \,.\, \lambda n \colon \pi_m \,. \\
\qquad\qquad c \lceil \$ \rceil\ ((Step_M\ (List\text{-}of\text{-}Quadruples\ L))\ c\ n)
\end{array}
$$

Recall from Section 3.2.2 that *List-of-Quadruples* first transforms L into a list of (encodings of) quadruples of adjacent tape symbols, then $Step_M$ performs the global ID transformation by 'local' transformations of quadruples. Finally, we add the fist unchanged end marker \$ on the left of the tape.

This finishes the definition of a single ID transformation step by a DTM within the STLC$^{\leq 5}$. Indeed, all terms defined are of order *at most five.*

3.8 Iterating Turing Machine Steps

Now we need to iterate the one-step ID transformations implemented by $Move_M$ the desired k-exponential number of times in order to be able to model any k-exponential time bounded computations. This is achieved by using the iterated exponentiation arithmetic on Church numerals over varying domains; see (Fortune et al. 1983).

Let B_0 be an initial type to be specified later, and define the types $B_{i+1} \equiv B_i \to B_i$. The type of representation of natural numbers over B_i is

$$I_i \equiv (B_i \to B_i) \to (B_i \to B_i)$$

A Church numeral $\bar{n}$ for $n \in \omega$ over B_i is represented by

$$\bar{n} \equiv \lambda f \colon B_i \to B_i \,.\, \lambda x \colon B_i \,.\, \underbrace{f(f(\ldots f}_{n}(x)\ldots))$$

When B_i is fixed, the only functions of types $I_i \to \ldots \to I_i$ are generalized polynomials generated by 0, successor, addition, multiplication, conditional 'if zero ... then ... else ...'; exponentiation and subtraction are not λ-definable (due to Schwichtenberg-Statman). However, when the domain and range of functions are allowed to vary, exponentiation and subtraction become representable. In particular, define

$$E_i \equiv \lambda m \colon I_{i+1} \,.\, \lambda n \colon I_i \,.\, (m\ n)$$

Then $E_i\ \bar{m}\ \bar{n}$ reduces to the representation of n^m over domain B_i, given that $\bar{m}$ is represented over domain B_{i+1} and $\bar{n}$ is represented over domain B_i.

Iterating this construction we get the term of type I_0 (where we superscript occurrences of numerals for n and 2 by their types)

$$k\text{-}EXP(n) \equiv (\ldots((n^{I_k}\ 2^{I_{k-1}})\ 2^{I_{k-2}})\ldots 2^{I_1})\ 2^{I_0}\ :\ I_0 \tag{9}$$

which converts to the representation of the numeral $\left.2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}\right\}k+1$ over domain B_0.

For a fixed $k \in \omega$ the size of the (non-normalized) term in (9) is $O(n)$ (but type annotations in lambda-abstractions are exponential in k, which is however considered a constant; recall that we consider slices of the term equality problem in STLC$^{\leq k}$ for *fixed* k).

Now let $B_0 \equiv (\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m$ of order five, let M be a DTM working in time $\left.2^{2^{\cdot^{\cdot^{\cdot^{2^{dn}}}}}}\right\}k+1$ (for some $d \in \omega$) recognizing a problem complete for $\cup_{c>0}$ *k-DEXPTIME*(cn) under polynomial time linearly bounded reductions (such problems and machines are known to exist), and let $s_1 s_2 \ldots s_{n-1} s_n$ be an input to M. Write a term

$$\begin{aligned}&\mathit{Result}\Big(\ k\text{-}EXP(\underbrace{add\ n(add\ n(\ldots(add}_{d-1}\ n\ n)\ldots)))\\ &\qquad\qquad \mathit{Move}_M\ [\ \lceil\$\rceil, \lceil s_1\rceil, \lceil s_2\rceil, \ldots \lceil s_{n-1}\rceil, \lceil s_n\rceil, \lceil\$\rceil\ ]\ \Big),\end{aligned} \tag{10}$$

where:

1. $add\colon I_k \to I_k \to I_k$ is λ-definable addition on Church numerals over B_k.
2. $[\ \ulcorner\$\urcorner, \ulcorner s_1\urcorner, \ulcorner s_2\urcorner, \ldots \ulcorner s_{n-1}\urcorner, \ulcorner s_n\urcorner, \ulcorner\$\urcorner\ ]\colon \underbrace{(\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m}_{B_0}$ is the list encoding of the initial machine's tape of size $O(n)$;
3. $Move_M\colon \underbrace{((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m)}_{B_0} \to$
$$\underbrace{((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m)}_{B_0}$$
is the lambda term described above implementing a single step of M;
4. $k\text{-}EXP(\underbrace{add\ n(add\ n(\ldots(add}_{d-1}\ \ n\ n)\ldots)))\colon (B_0 \to B_0) \to (B_0 \to B_0)$ is a Church numeral, which converts to the normal form representing $\left.2^{2^{\cdot^{\cdot^{2^{k+1}}}}}\right\}dn$ over domain B_0;
5. therefore, the term (10) iterates the moves of the DTM the required k-exponential number of times;
6. the order of the term (10) is therefore $k+6$; indeed, the order of B_0 is 4, the order of $I_k = (B_k \to B_k) \to (B_k \to B_k)$ is $ord(B_k) + 2$, and $ord(B_{i+1}) = ord(B_i) + 1$; therefore, the order of I_k (maximal type in (10)) is $k+6$.
7. $Result\colon ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m)$ is the lambda term extracting the result, either
   (a) $\lambda c\colon \pi_m \to \pi_m \to \pi_m\,.\,\lambda n\colon \pi_m\,.\,c\ \ulcorner\$\urcorner\ (c\ \ulcorner q_a\urcorner\ (c\ \ulcorner\$\urcorner\ n))$ (accept) or
   (b) $\lambda c\colon \pi_m \to \pi_m \to \pi_m\,.\,\lambda n\colon \pi_m\,.\,c\ \ulcorner\$\urcorner\ (c\ \ulcorner q_r\urcorner\ (c\ \ulcorner\$\urcorner\ n))$ (reject)

   from the lists remaining after DTM's computations (respectively)

   (a) $[\ \ulcorner\$\urcorner, \ulcorner q_a\urcorner, \ulcorner\sqcup\urcorner, \ulcorner\sqcup\urcorner, \ldots, \ulcorner\sqcup\urcorner, \ulcorner\$\urcorner\ ]$ or
   (b) $[\ \ulcorner\$\urcorner, \ulcorner q_r\urcorner, \ulcorner\sqcup\urcorner, \ulcorner\sqcup\urcorner, \ldots, \ulcorner\sqcup\urcorner, \ulcorner\$\urcorner\ ]$,

   where the last two lists may have varying length (recall that by convention DTMs are standardized and in the end of the work replace all occupied tape cells with blanks, move to the leftmost tape position, and enter accepting/rejecting state).

   The function *Result* is λ-defined immediately:

$$\begin{aligned} \mathit{Result}\colon & ((\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m) \to \mathsf{Bool} \equiv \\ & \lambda L\colon (\pi_m \to \pi_m \to \pi_m) \to \pi_m \to \pi_m \,.\, \mathsf{eq}\ p_{\$q_a\sqcup\sqcup}\ (L\ \mathit{pair}\ p_{\$\$\$\$}), \end{aligned}$$

(recall that we assume (wlog) that M uses at least two tape cells in any computation).

This finishes the linearly bounded reduction from *k-DEXPTIME* to deciding equality in $\mathrm{STLC}^{\leq k+6}$ between two terms one of which may be fixed to the fourth-order term

$$\lambda c\colon \pi_m \to \pi_m \to \pi_m \,.\, \lambda n\colon \pi_m \,.\, c\ \lceil\$\rceil\ (c\ \lceil q_a\rceil\ (c\ \lceil\$\rceil\ n)) \quad \text{(DTM accepts)} \tag{11}$$

Therefore, deciding equality in $\mathrm{STLC}^{\leq k+6}$ requires k-times iterated exponential time, or is *k-DEXPTIME*-hard. This completes the proof of Theorem 3.1.

4 Complexity of the Bounded-Order Higher-Order Matching

As an application of the preceding results we now turn to improving lower bounds for the higher-order matching problem of fixed bounded order. Recall that the problem consists, given closed normalized terms t of type $\sigma_1 \to \ldots \to \sigma_n \to \tau$ and u of type τ to determine whether there exist terms x_i of types σ_i (for $1 \leq i \leq n$) such that $tx_1 \ldots x_n =_{\beta\eta} u$. The order of the instance $tx_1 \ldots x_n \stackrel{?}{=} u$ of the problem is defined as $\max\{\mathit{ord}(\sigma_1), \ldots, \mathit{ord}(\sigma_n)\}$, i.e., the maximal order of type of a free variable on the left. Note that in an instance of the matching problem the instantiable variables are only allowed on the left (in contrast to unification, where variables are allowed on both sides and which is undecidable for order three without constants and for order two with constants, due to Huet-Goldfarb).

While the decidability status of the higher-order matching problem remains open, it makes sense to provide compelling evidence of the inherent difficulty of the problem by proving non-trivial lower bounds (Compton & Henson 1990, Problem 10.11, p. 75). As we mentioned in the Introduction (Theorem 1.1), if one does not impose the bound on the orders of variables in instances, then the higher-order matching problem has a strong non-elementary lower bound, but this result does not imply any lower bounds for the k-order bounded matching ($k \in \omega$). Now we have enough machinery to settle this problem.

Note that we spent additional effort (e.g., in Section 3.8 and in the whole construction) to guarantee that the term appearing on the right of equalities in the reduction class is *fixed* and does not depend neither on input, nor on the machine description, which is important for our current development.

For a fixed $k \in \omega$ consider the instances, for varying input strings $s_1 \ldots s_n$, of closed equations

$$(10) \stackrel{?}{=} (11) \tag{12}$$

where the term (11) on the right is normalized. We now turn in linear time the equation instance (12) into an instance of the higher-order matching problem. The key obstacle is that the term (10) on the left of the instance (12) is not normalized. The trick we already used in (Vorobyov 1997) is as follows. Let us work with finite systems of instances of the matching problem rather than with single instances. Start with the system S containing a single input equation (12). Replace every β-redex $(\lambda x\colon \sigma\,.\,M)^{\sigma\to\tau}\ N^{\sigma}$ in (12) with the (non-redex) term $((f^{(\sigma\to\tau)\to\sigma\to\tau}\ (\lambda x\colon \sigma\,.\,M)^{\sigma\to\tau})\ N^{\sigma})$, where $f\colon (\sigma\to\tau)\to\sigma\to\tau$ is a fresh free variable, and add equation $f = \lambda g\colon \sigma\to\tau\,.\,\lambda z\colon \sigma\,.\,gz$ to system S. The variable f plays the role of *apply*: it takes two arguments and applies the first one to the second one.

Similarly, replace every η-redex $\lambda x\colon \sigma\,.\,M^{\sigma\to\tau}\ x^{\sigma}$ (x not free in M) in (12) with the (non-redex) term $\lambda x\colon \sigma\,.\,f^{\sigma\to\sigma}(M^{\sigma\to\tau}\ x^{\sigma})$, where $f\colon \sigma\to\sigma$ is a fresh free variable, and add equation $f = \lambda u\colon \sigma\,.\,u$ to S.

The resulting system S contains free variables only on the left-hand sides and all terms on the left and right-hand sides are normalized. It remains to reduce the system of equations S to *just one equation* by tupling: $\bigwedge_{i=1}^{n} M_i = N_i \;\Leftrightarrow\; \lambda f.fM_1\ldots M_k = \lambda f.fN_1\ldots N_k$. Note that the resulting equation *does not contain variables on the right*, so we translated the initial equation (12) to the instance of the higher-order matching problem with normalized terms. Note that the right-hand side of (12) reduces to the right-hand side of (12) if and only if the obtained instance of the matching problem has a solution. It remains to notice that the order of the matching problem is *by one greater* than the maximal order of subterm in (10). This is because the maximal order subterm (redex) $(n^{I_k}\ 2^{I_{k-1}})$ (see (10), (9)) introduces, while eliminating the β-redex, the free variable $f\colon I_k\to I_{k-1}\to I_{k-1}$ Thus the maximal order of a variable in the resulting matching instance is by one larger the largest order of subterm in (10). Summarizing, we obtain the promised

Theorem 4.1 *k-DEXPTIME is linearly reducible to the* higher-order matching problem *(in STLC) of order $k+7$. Thus, order $k+7$ matching* requires *(lower bound) k-level exponential time.*

More precisely, given a description of a problem P and input x of size n one can construct in polynomial time an instance S of the higher-order matching problem of order $k+7$ such that $x \in P$ if and only if S has a solution. $\square$

This gives an improvement (lower bounds for the fixed order matching) over the result of (Vorobyov 1997), which does not imply any lower bounds for the bounded order matching.

5 Conclusions

In this paper we settled the optimal (up to an additive constant 6) lower bound for the equality in the order bounded fragments $\mathrm{STLC}^{\leq k}$ (for fixed k) of the simply typed lambda calculus STLC. It is now known that checking whether a given term of order $k+6$ normalizes to a fixed normal form is as hard as deciding a k-*DEXPTIME*-complete problem. Thus comparing $\beta\eta$-equality of terms of $\mathrm{STLC}^{\leq k+6}$ requires the k-times iterative exponential time.

This result improves upon previous results (Statman 1979, Mairson 1992, Vorobyov 1997), which did not imply any nontrivial lower bounds for fixed orders (however the strong lower bound of (Vorobyov 1997) for unbounded orders persists and is not superceded by the present result), and over the result of (Hillebrand & Kanellakis 1996), reducing k-*DEXPTIME* to the order $2k+3$ STLC (we thus get a double increase, in terms of heights of exponentiation towers, of lower complexity bounds).

As an application, we obtained new strong lower bound for the fixed-order higher-order matching, still an open problem. It turns out that for every $k \in \omega$ deciding matchability of order $k+7$ is at least as difficult as deciding an arbitrary (complete) problem in k-*DEXPTIME*. Thus $k+7$-order matching requires the k-times iterative exponential time. Earlier results either gave nontrivial lower bounds for unbounded order matching (Vorobyov 1997), or for the specific orders up to four only (Wierzbicki 1999).

Methodologically, we obtained our results as consequences of the streamlined and immediate proof of Statman's theorem by a single step generic reduction. This gives a conceptually simpler proof yielding stronger lower bounds and bound for fixed orders, in contrast with the previous two-step reduction proofs of (Statman 1979, Mairson 1992, Vorobyov 1997).

It remains open whether the additive constants 6 and 7 in our lower bounds could be diminished to give stronger results.

References

Barendregt, H. (1984), *The Lambda Calculus. Its Syntax and Semantics*, North-Holland.

Compton, K. J. & Henson, C. W. (1990), 'A uniform method for proving lower bounds on the computational complexity of logical theories', *Annals Pure Appl. Logic* **48**, 1–79.

Dowek, G. (1994), 'Third order matching is decidable', *Annals Pure Appl. Logic* **69**, 135–155. Preliminary version in LICS'92.

Fortune, S., Leivant, D. & O'Donnell, M. (1983), 'The expressiveness of simple and second-order type structures', *J. ACM* **30**(1), 151–185.

Hillebrand, G. & Kanellakis, P. (1994), Functional database query languages as typed lambda calculi of fixed order, *in* '13th ACM Symp. on Principles of Programming Languages (PODS'94)', pp. 222–231.

Hillebrand, G. & Kanellakis, P. (1996), On the expressive power of simply typed and let-polymorphic lambda calculi, *in* '11th Annual IEEE Symp. on Logic in Computer Science (LICS'96)', pp. 253–263.

Hillebrand, G., Kanellakis, P. & Mairson, H. (1993), Database query languages embedded in the typed lambda calculus, *in* '8th Annual IEEE Symp. on Logic in Computer Science (LICS'93)', pp. 332–343.

Hindley, J. & Seldin, J. (1986), *Introduction to Combinators and Lambda Calculus*, Cambridge Univ. Press.

Huet, G. (1976), *Résolution d'Équations dans les Langages d'Ordre 1, 2, ... ,* ω, Thèse de Doctorat d'État, Université de Paris VII.

Loader, R. (1993), The undecidability of λ-definability. "Types" electronic forum, to appear in *Church's Festschrift*.

Mairson, H. (1992), 'A simple proof of a theorem of Statman', *Theor. Comput. Sci.* **103**, 387–394.

Schwichtenberg, H. (1982), Complexity of normalization in the pure typed λ-calculus, *in* A. S. Troelstra & D. van Dalen, eds, 'L. E. J. Brouwer Centenary Symposium', North-Holland, pp. 453–458.

Schwichtenberg, H. (1991), 'An upper bound for reduction sequences in the typed λ-calculus', *Archive of Mathematical Logic* **30**, 405–408.

Statman, R. (1979), ‘The typed λ-calculus is not elementary recursive’, *Theor. Comput. Sci.* **9**, 73–81.

Statman, R. (1982), ‘Completeness, invariance, and λ-definability’, *J. Symb. Logic* **47**(1), 17–26.

Stockmeyer, L. J. (1974), The complexity of decision problems in automata theory and logic, PhD thesis, MIT Lab for Computer Science. (Also /MIT/LCS Tech Rep 133).

Troelstra, A. S. & Schwichtenberg, H. (1996), *Basic Proof Theory*, Vol. 43 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge Univ. Press.

Vorobyov, S. (1997), The “hardest” natural decidable theory, *in* G. Winskel, ed., ‘12th Annual IEEE Symp. on Logic in Computer Science (LICS’97)’, IEEE, pp. 294–305. Available from `http://www.mpi-sb.mpg.de/~sv`.

Wierzbicki, T. (1999), Complexity of the higher-order matching, *in* H. Ganzinger, ed., ‘16th International Conference on Automated Deduction (CADE’99)’, Vol. 1632 of *Lect. Notes Comput. Sci.*, pp. 82–96.

Wolfram, D. A. (1993), *The Clausal Theory of Types*, Vol. 21 of *Cambridge Tracts in Theoretical Computer Science Series*, Cambridge Univ. Press.

INFORMATIK

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Birgit Hofmann
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-1999-2-005	J. Wu	Symmetries in Logic Programs
MPI-I-1999-2-004	V. Cortier, H. Ganzinger, F. Jacquemard, M. Veanes	Decidable fragments of simultaneous rigid reachability
MPI-I-1999-2-003	U. Waldmann	Cancellative Superposition Decides the Theory of Divisible Torsion-Free Abelian Groups
MPI-I-1999-2-001	W. Charatonik	Automata on DAG Representations of Finite Trees
MPI-I-1999-1-002	N.P. Boghossian, O. Kohlbacher, H.-. Lenhof	BALL: Biochemical Algorithms Library
MPI-I-1999-1-001	A. Crauser, P. Ferragina	A Theoretical and Experimental Study on the Construction of Suffix Arrays in External Memory
MPI-I-98-2-018	F. Eisenbrand	A Note on the Membership Problem for the First Elementary Closure of a Polyhedron
MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-014	Y. Gurevich, M. Veanes	Partisan Corroboration, and Shifted Pairing
MPI-I-98-2-013	H. Ganzinger, F. Jacquemard, M. Veanes	Rigid Reachability
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid E-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-2-002	F. Jacquemard, C. Meyer, C. Weidenbach	Unification in Extensions of Shallow Equational Theories
MPI-I-98-1-031	G.W. Klau, P. Mutzel	Optimal Compaction of Orthogonal Grid Drawings
MPI-I-98-1-030	H. Brönniman, L. Kettner, S. Schirra, R. Veltkamp	Applications of the Generic Programming Paradigm in the Design of CGAL

MPI-I-98-1-029	P. Mutzel, R. Weiskircher	Optimizing Over All Combinatorial Embeddings of a Planar Graph
MPI-I-98-1-028	A. Crauser, K. Mehlhorn, E. Althaus, K. Brengel, T. Buchheit, J. Keller, H. Krone, O. Lambert, R. Schulte, S. Thiel, M. Westphal, R. Wirth	On the performance of LEDA-SM
MPI-I-98-1-027	C. Burnikel	Delaunay Graphs by Divide and Conquer
MPI-I-98-1-026	K. Jansen, L. Porkolab	Improved Approximation Schemes for Scheduling Unrelated Parallel Machines
MPI-I-98-1-025	K. Jansen, L. Porkolab	Linear-time Approximation Schemes for Scheduling Malleable Parallel Tasks
MPI-I-98-1-024	S. Burkhardt, A. Crauser, P. Ferragina, H. Lenhof, E. Rivals, M. Vingron	q-gram Based Database Searching Using a Suffix Array (QUASAR)
MPI-I-98-1-023	C. Burnikel	Rational Points on Circles
MPI-I-98-1-022	C. Burnikel, J. Ziegler	Fast Recursive Division
MPI-I-98-1-021	S. Albers, G. Schmidt	Scheduling with Unexpected Machine Breakdowns
MPI-I-98-1-020	C. Rüb	On Wallace's Method for the Generation of Normal Variates
MPI-I-98-1-019		2nd Workshop on Algorithm Engineering WAE '98 - Proceedings
MPI-I-98-1-018	D. Dubhashi, D. Ranjan	On Positive Influence and Negative Dependence
MPI-I-98-1-017	A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, E. Ramos	Randomized External-Memory Algorithms for Some Geometric Problems
MPI-I-98-1-016	P. Krysta, K. Loryś	New Approximation Algorithms for the Achromatic Number
MPI-I-98-1-015	M.R. Henzinger, S. Leonardi	Scheduling Multicasts on Unit-Capacity Trees and Meshes
MPI-I-98-1-014	U. Meyer, J.F. Sibeyn	Time-Independent Gossiping on Full-Port Tori
MPI-I-98-1-013	G.W. Klau, P. Mutzel	Quasi-Orthogonal Drawing of Planar Graphs
MPI-I-98-1-012	S. Mahajan, E.A. Ramos, K.V. Subrahmanyam	Solving some discrepancy problems in NC*
MPI-I-98-1-011	G.N. Frederickson, R. Solis-Oba	Robustness analysis in combinatorial optimization
MPI-I-98-1-010	R. Solis-Oba	2-Approximation algorithm for finding a spanning tree with maximum number of leaves
MPI-I-98-1-009	D. Frigioni, A. Marchetti-Spaccamela, U. Nanni	Fully dynamic shortest paths and negative cycle detection on diagraphs with Arbitrary Arc Weights
MPI-I-98-1-008	M. Jünger, S. Leipert, P. Mutzel	A Note on Computing a Maximal Planar Subgraph using PQ-Trees
MPI-I-98-1-007	A. Fabri, G. Giezeman, L. Kettner, S. Schirra, S. Schönherr	On the Design of CGAL, the Computational Geometry Algorithms Library
MPI-I-98-1-006	K. Jansen	A new characterization for parity graphs and a coloring problem with costs
MPI-I-98-1-005	K. Jansen	The mutual exclusion scheduling problem for permutation and comparability graphs
MPI-I-98-1-004	S. Schirra	Robustness and Precision Issues in Geometric Computation
MPI-I-98-1-003	S. Schirra	Parameterized Implementations of Classical Planar Convex Hull Algorithms and Extreme Point Compuations
MPI-I-98-1-002	G.S. Brodal, M.C. Pinotti	Comparator Networks for Binary Heap Construction
MPI-I-98-1-001	T. Hagerup	Simpler and Faster Static AC^0 Dictionaries
MPI-I-97-2-012	L. Bachmair, H. Ganzinger, A. Voronkov	Elimination of Equality via Transformation with Ordering Constraints
MPI-I-97-2-011	L. Bachmair, H. Ganzinger	Strict Basic Superposition and Chaining
MPI-I-97-2-010	S. Vorobyov, A. Voronkov	Complexity of Nonrecursive Logic Programs with Complex Values