Semantic Evaluation, Intersection Types and Complexity of Simply Typed Lambda Calculus

Kazushige Terui[1]

1 **RIMS, Kyoto University**
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, Japan
terui@kurims.kyoto-u.ac.jp

Abstract

Consider the following problem: given a simply typed lambda term of Boolean type and of order r, does it normalize to "true"? A related problem is: given a term M of word type and of order r together with a finite automaton D, does D accept the word represented by the normal form of M? We prove that these problems are n-EXPTIME complete for $r = 2n + 2$, and n-EXPSPACE complete for $r = 2n + 3$.

While the hardness part is relatively easy, the membership part is not so obvious; in particular, simply applying β reduction does not work. Some preceding works employ semantic evaluation in the category of sets and functions, but it is not efficient enough for our purpose.

We present an algorithm for the above type of problem that is a fine blend of β reduction, Krivine abstract machine and semantic evaluation in a category based on preorders and order ideals, also known as the Scott model of linear logic. The semantic evaluation can also be presented as intersection type checking.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases simply typed lambda calculus, computational complexity, denotational semantics, intersection types

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.323

Category Regular Research Paper

1 Introduction

Beta reduction vs. semantic evaluation. Let us begin with a simple puzzle. Consider the simply typed lambda calculus. For every simple type σ, let $\mathbf{N}(\sigma) := (\sigma \to \sigma) \to (\sigma \to \sigma)$. Then every Church numeral $\mathbf{n}$ has type $\mathbf{N}(\sigma)$, and the exponentiation function 2^x has type $\mathbf{N}(\sigma \to \sigma) \to \mathbf{N}(\sigma)$ for an arbitrary σ. Let $\mathbf{B} := o \to o \to o$ (where o is the base type) and $\mathbf{tt} := \lambda xy.x : \mathbf{B}$.

Now the question is as follows. Fix a type σ and a closed term M of type $\mathbf{N}(\sigma) \to \mathbf{B}$. Then what is the computational complexity of deciding whether $M\mathbf{n} =_\beta \mathbf{tt}$, when n ranges over the set $\mathbb{N}$ of natural numbers?

Since arbitrary hyperexponential functions are available in M, one might be tempted to answer that it requires hyperexponential time in n in general. Although it is true if one uses β reduction, there is actually a much more efficient algorithm, according to which it can be decided in *linear* time in n, for fixed M, with a huge coefficient depending only on σ.

Such an algorithm is most easily provided by *semantic evaluation* in the category **Set** of sets and functions. Define $[\![o]\!] = \{0, 1\}$ and $[\![\sigma \to \tau]\!] = [\![\tau]\!]^{[\![\sigma]\!]}$. The denotation $[\![\mathbf{n}]\!]$ of $\mathbf{n}$ can be computed by induction on n, and each inductive step requires only constant time (which

23rd International Conference on Rewriting Techniques and Applications (RTA'12).
Editor: A. Tiwari; pp. 323–338

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


Order	2	3	4	5	6	7
Complexity	P	PSPACE	EXPTIME	EXPSPACE	2-EXPTIME	2-EXPSPACE

Figure 1 Time-space alternation in simply typed lambda calculus.

depends on σ). We then apply the function $[\![M]\!]$ (which may be precomputed independently of n) to $[\![n]\!]$. Since the semantics is sound and $[\![\mathbf{tt}]\!] \neq [\![\mathbf{ff}]\!]$, we can check if $M\mathbf{n} =_\beta \mathbf{tt}$ by comparing $[\![M\mathbf{n}]\!]$ with $[\![\mathbf{tt}]\!]$. Clearly the total runtime is linear in n.

This demonstrates that β reduction is desperately inefficient for computing a finite-valued function in simply typed lambda calculus. It may be ascribed to the fact that β reduction is a versatile machinery independent of typing, so that it does not take advantage of the type information at all. In contrast, semantic evaluation fully exploits the type information and that is the main reason why it is so efficient for some problems.

Order and complexity. Turing to a more general situation, it has been observed by Statman [20] that the problem of deciding β-equivalence (or $\beta\eta$-equivalence) of two simply typed terms is not elementary recursive. The explosion of complexity is caused by the increase of the order of the input terms. Hence it is interesting to study the complexity of Statman's problem with restricted order. In this context, Schubert [19] has shown that the problem of deciding β-equivalence is PTIME complete for terms of order up to 2, and is PSPACE complete for terms of order up to 3, provided that one of the input terms is in normal form.

In the meantime, a curious phenomenon on the order and complexity has been observed in some extensions of simply typed lambda calculus. Goerdt and Seidl [8] have studied interpretations of higher type primitive recursive definitions (i.e. System T terms) in finite structures, and proved that terms of order 2r+2 characterize r-EXPTIME predicates, while terms of order 2r+3 characterize r-EXPSPACE predicates. Namely, time and space complexity classes *alternate* when the order increases. Similar results have been shown by Kristiansen and Voda [14, 13] for System T withour successor and by Hillebrand and Kanellakis [9] for simply typed lambda calculus with equality test. A remarkable similarity among all these works is that they employ a *hybrid* algorithm that mixes β reduction and semantic evaluation in **Set**.

About this paper. The goal of this paper is to show a similar time-space alternation result in plain simply typed lambda calculus. We consider the following decision problems (see the next section for precise definitions of order r and word type $\mathbf{W}$).

1. Bool(r): Given a closed term $M : \mathbf{B}$ of order r, does M reduce to $\mathbf{tt}$?
2. RegLang(r): Given a closed term $M : \mathbf{W}$ of order r and a nondeterministic finite automaton D, does D accept the word represented by the normal form of M? (The size of D is defined to be the number of states.)
3. Term(r): Given a closed term $M : \tau$ of order r and another term $N : \tau$ *in normal form*, do we have $M =_{\beta\eta} N$?

We prove:

▶ **Theorem 1** (Time-Space Alternation)**.**

1. Bool$(2r+2)$ *and* RegLang$(2r+2)$ *are complete for* r-EXPTIME.
2. Bool$(2r+3)$ *and* RegLang$(2r+3)$ *are complete for* r-EXPSPACE *(see Figure 1).*

The third problem, that is essentially Schubert's problem, is subtle and will be discussed in Section 5.

To show the membership part, we basicaly follow [8, 9, 14] and employ a hybrid algorithm. However, it turns out that evaluation in the category **Set** is not efficient enough for our purpose (see Subsection 4.2). We are thus led to work in another cartesian closed category that arises by the Kleisli construction from a model of linear logic based on prime algebraic complete lattices and lub-preserving maps [10], or equivalently based on preorders and order ideals [23, 24]. It is simply called the *Scott model of linear logic* in [6]. Evaluation in this category can be expressed as type checking in an intersection type system that is essentially due to [18], and is similar to the essential type assignment system [21, 22]. Our algorithm also employs the mechanism of Krivine's abstract machine (KAM, [15]).

The rest of this paper is organized as follows. Section 2 is a preliminary, Section 3 introduces the semantics and the associated intersection type system, and Section 4 discusses some optimization techniques. Sections 5 proves the membership part of Theorem 1, while Section 6 proves the hardness part. Finally, Section 7 concludes the paper.

2 Preliminary

Given a set Q, $\sharp Q$ denotes its cardinality. Similarly, $\sharp\Gamma$ denotes the length if Γ is a list. The hyperexponential function hyp is defined by: $hyp(0,n) := n$ and $hyp(r+1,n) := 2^{hyp(r,n)}$. We let $hyp(-1,n) := \log n$ for convenience.

We write $poly(x)$ to denote a function bounded by some polynomial in x that depends on the context. The hyperexponential complexity classes are defined by r-EXPTIME $:= \bigcup_{c>0} \mathrm{DTIME}[hyp(r,n^c)]$ and r-EXPSPACE $:= \bigcup_{c>0} \mathrm{DSPACE}[hyp(r,n^c)]$. In particular, we have 0-EXPTIME = P and 0-EXPSPACE = PSPACE. The classes r-AEXPTIME and r-AEXPSPACE are similarly defined by means of alternating Turing machines. The following relations are fundamental:

$$r\text{-AEXPTIME} = r\text{-EXPSPACE}, \qquad r\text{-AEXPSPACE} = r+1\text{-EXPTIME}.$$

We work on the simply typed lambda calculus with a single base type o and without any term constants; the restriction to the single base type is inessential. So types are of the form o or $\sigma \to \tau$. Provided that countably many variables $x^\sigma, y^\sigma, z^\sigma, \ldots$ are given for each type σ, terms are generated by the following rules, where $\overline{x} \rhd M : \tau$ indicates that M is a term of type τ whose free variables are among the list $\overline{x}$:

$$\frac{\overline{x} = x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}}{\overline{x} \rhd x_i : \sigma_i} \qquad \frac{\overline{x}, y^\sigma \rhd M : \tau}{\overline{x} \rhd \lambda y^\sigma.M : \sigma \to \tau} \qquad \frac{\overline{x} \rhd M : \sigma \to \tau \quad \overline{x} \rhd N : \sigma}{\overline{x} \rhd MN : \tau}$$

We often omit type superscripts and adopt the variable convention. We write $M : \sigma$ or M^σ to indicate that M is of type σ. By a *subtype* of σ, we simply mean a type occurring in σ as a subexpression (which has nothing to do with subtyping disciplines). We omit parentheses in the standard way, and write $\sigma^n \to \tau$ for $\sigma \to \cdots \sigma \to \tau$ (n times).

Data types for Boolean values, natural numbers and binary words are given by:

$$\begin{array}{lcllcll}
\mathbf{B} & := & o \to o \to o & \mathbf{tt} & := & \lambda xy.x & : \mathbf{B} \\
 & & & \mathbf{ff} & := & \lambda xy.y & : \mathbf{B} \\
\mathbf{N} & := & (o \to o) \to (o \to o) & \mathbf{n} & := & \lambda f x.f^n x & : \mathbf{N} \\
\mathbf{W} & := & (o \to o)^2 \to (o \to o) & \mathbf{w} & := & \lambda f_0 f_1 x.f_{i_n}(\cdots f_{i_2}(f_{i_1} x)\cdots) & : \mathbf{W}.
\end{array}$$

where $n \in \mathbb{N}$ and $w = i_1 \cdots i_n \in \{0,1\}^*$.

Suppose that a term M has a subterm of type σ, and τ is a subtype of σ. The set of all such subtypes τ is denoted by $Type(M)$.

The *order* and *arity* of a type σ are inductively defined by:

$$\begin{array}{rclcrcl} \mathit{Order}(o) & := & 0 & & \mathit{Order}(\sigma \to \tau) & := & \max(\mathit{Order}(\sigma)+1, \tau) \\ \mathit{Arity}(o) & := & 1 & & \mathit{Arity}(\sigma \to \tau) & := & \mathit{Arity}(\tau)+1. \end{array}$$

(We define $\mathit{Arity}(o) = 1$ rather than 0 just to make calculation easier.)

The *order* (resp. *arity*) of a term M is the maximal order (resp. arity) of any type in $\mathit{Type}(M)$. The *size* $|M|$ of M is the number of nodes in the term formation tree for M. $|\sigma|$ is similarly defined. We denote by $\Lambda(s,a,r)$ the set of terms of size $\leq s$, arity $\leq a$ and order $\leq r$.

Let $M : \sigma$. Our definition of $|M|$ does not take into account the size of a type occurring in it. However, this does not cause any practical problem, since the size of such a type can be bounded, up to some optimization, in terms of $|M|$, $|\sigma|$ and the order of M. In more detail, let $\tau_1 \to \tau_2 \in \mathit{Type}(M)$. Suppose that $\tau_1 \to \tau_2$ is not a subtype of σ, and that M does not contain any subterm of the form $(\lambda x.N)^{\tau_1 \to \tau_2}$ nor $N^{\tau_1 \to \tau_2} K^{\tau_1}$. In that case, we may safely replace $\tau_1 \to \tau_2$ occurring in M by o preserving the type of M. Repeat this until such a type $\tau_1 \to \tau_2$ does not exist in $\mathit{Type}(M)$. The resulting term is said to be *optimally typed* (it has to do with *principal typing*).

▶ **Lemma 2.** *Let $M : \sigma$ be an optimally typed term of arity a and of order r. Then we have $a \leq |M| + 3|\sigma|$. As a consequence, for any $\tau \in \mathit{Type}(M)$ we have $|\tau| \leq a^r \leq (|M| + 3|\sigma|)^r$.*

Hereafter we assume that terms are optimally typed. In practice, the result type σ and the order r will be fixed. Hence the actual size of M taking types into account is polynomial in $|M|$. Since we are only interested in complexity classes above P, we may safely ignore the size of a type occurring in a term.

Given a term M, we denote by $\langle\!\langle M \rangle\!\rangle_r$ the term of order r obtained by reducing all the redexes of order $> r$ in M. The following result is standard.

▶ **Lemma 3.** *Suppose that $M : \sigma$ and $\mathit{Order}(\sigma) \leq r$. If $M \in \Lambda(s,a,r+1)$, then $\langle\!\langle M \rangle\!\rangle_r \in \Lambda(2^s, a, r)$. $\langle\!\langle M \rangle\!\rangle_r$ can be computed in time $2^{O(s)}$.*

See [2] for a precise analysis of β reduction length in simply typed lambda calculus.

3 Semantic evaluation and intersection types

3.1 Category of preorders and order ideals

We now introduce a cartesian closed category $\mathbf{ScottL}_!$ (the notation is taken from [6]). While it can be directly presented as the category of prime algebraic complete lattices and Scott continuous functions, we prefer a more elementary description due to Winskel [23] (see also [24, 6]), which immediately suggests a correspondence with intersection types. We first explain, assuming some acquaintance with models of linear logic (see, e.g., [3, 17]), how it naturally arises from simple building blocks. But this part is not needed for the rest. We later describe an interpretation of simply typed lambda calculus in $\mathbf{ScottL}_!$ concretely, which can be understood without any prerequisite.

Let $\mathbf{ScottL}$ be the category in which each object is a preorder $A = (A, \sqsupseteq_A)$ and each morphism $R : A \nrightarrow B$ is an *order ideal*, that is a binary relation $R \subseteq A \times B$ such that $a' \sqsupseteq_A a \; R \; b \sqsupseteq_B b'$ implies $a' \; R \; b'$. Composition of relations is standard, while the identity $1_A : A \nrightarrow A$ on object A is given by $\sqsupseteq_A$. $\mathbf{ScottL}$ is a compact closed category with

biproducts, where the dual A^*, the tensor product $A \otimes B$, the unit I, the biproduct $A \,\&\, B$ and the zero object $\top$ for $A = (A, \sqsupseteq_A)$ and $B = (B, \sqsupseteq_B)$ are given by:

$$\begin{array}{rclrclrcl} A^* & := & (A, \sqsubseteq_A) & A \otimes B & := & (A \times B, \sqsupseteq_A \times \sqsupseteq_B) & I & := & (\{*\}, =) \\ & & & A \,\&\, B & := & (A \uplus B, \sqsupseteq_A \uplus \sqsupseteq_B) & \top & := & (\emptyset, \emptyset), \end{array}$$

where $\uplus$ denotes the disjoint union (see, e.g., [3]).

Up to now, **ScottL** looks an obvious variant of the category **Rel** of sets and relations. A great advantage of **ScottL** is that it naturally admits a *set*-based (rather than *multiset*-based) interpretation of exponentials, in contrast to **Rel** (see [17] for discussion; see also [6] which shows that **ScottL** is the *extensional collapse* of **Rel**). This allows us to evaluate terms in finite structures.

Specifically, given a set X and a binary relation $R \subseteq A \times B$, $\mathcal{P}_f X$ denotes the set of finite subsets of X and $\mathcal{P}_f R \subseteq \mathcal{P}_f A \times \mathcal{P}_f B$ is defined by

$$Y \; \mathcal{P}_f R \; X \iff \forall \alpha \in X \; \exists \beta \in Y.\; \beta \; R \; \alpha.$$

A comonad $(!, \epsilon, \delta)$ in **ScottL** as well as Seely isomorphisms $m^2_{A,B}$, m^0 (i.e., exponential isomorphisms) are given as follows (where $A = (A, \sqsupseteq_A)$ and $R : A \nrightarrow B$):

$$\begin{array}{rcll} !A & := & (\mathcal{P}_f A, \mathcal{P}_f \sqsupseteq_A) & \\ !R & := & \mathcal{P}_f R & : !A \nrightarrow !B \\ \epsilon_A & := & \{(X, \alpha) : \exists \alpha'.\; X \ni \alpha' \sqsupseteq_A \alpha\} & : !A \nrightarrow A \\ \delta_A & := & \{(X, \{Y_1, \ldots, Y_n\}) : X \sqsupseteq_{!A} Y_1 \cup \cdots \cup Y_n\} & : !A \nrightarrow !!A \\ m^2_{A,B} & := & \{((X, Y), Z) : X \uplus Y \sqsupseteq_{!(A \& B)} Z\} & : !A \otimes !B \nrightarrow !(A \& B) \\ m^0 & := & \{(*, \emptyset)\} & : I \nrightarrow !\top. \end{array}$$

It is routine to verify that these data indeed define a model of linear logic (a *Seely category* [17]). Hence by the Kleisli construction, we obtain a cartesian closed category $\mathbf{ScottL}_!$.

Rather than specifying the exact structure of $\mathbf{ScottL}_!$, we will concretely describe how to interpret lambda terms in it.

Let Q be a finite set. To each type σ, we associate a poset $[\![\sigma]\!]_Q = ([\![\sigma]\!]_Q, \sqsupseteq_\sigma)$ as follows (we omit subscript Q for readability).

$$[\![o]\!] := (Q, =), \qquad [\![\sigma \to \tau]\!] := (\mathcal{P}_f [\![\sigma]\!] \times [\![\tau]\!], (\mathcal{P}_f \sqsupseteq_\sigma)^{-1} \times \sqsupseteq_\tau).$$

Concretely, we have $(X, \alpha) \sqsupseteq_{\sigma \to \tau} (Y, \beta)$ iff $\forall \gamma \in X \; \exists \delta \in Y.\; \gamma \sqsubseteq_\sigma \delta$ and $\alpha \sqsupseteq_\tau \beta$. In particular, $X \subseteq Y$ implies $(X, \alpha) \sqsupseteq_{\sigma \to \tau} (Y, \alpha)$. We occasionally write $X \to \alpha$ to denote a pair (X, α), and $(X_1, \ldots, X_n \rhd \alpha)$ to denote an $n+1$ tuple $(X_1, \ldots, X_n, \alpha)$.

To each $\overline{x} \rhd M : \tau$ with $\overline{x} = x_1^{\sigma_1} \ldots, x_n^{\sigma_n}$, we associate a set

$$[\![\overline{x} \rhd M]\!] \subseteq \mathcal{P}_f [\![\sigma_1]\!] \times \cdots \times \mathcal{P}_f [\![\sigma_n]\!] \times [\![\tau]\!]$$

as follows.

$$\begin{array}{rcl} [\![\overline{x} \rhd x_i]\!] & := & \{(\overline{X} \rhd \alpha) : \exists \alpha' \in [\![\sigma_i]\!].\; X_i \ni \alpha' \sqsupseteq_{\sigma_i} \alpha\} \\ [\![\overline{x} \rhd \lambda y^\sigma.M]\!] & := & \{(\overline{X} \rhd Y \to \alpha) : (\overline{X}, Y \rhd \alpha) \in [\![\overline{x}, y^\sigma \rhd M]\!]\} \\ [\![\overline{x} \rhd M^{\sigma \to \tau} N^\sigma]\!] & := & \{(\overline{X} \rhd \alpha) : \exists Y \in \mathcal{P}_f [\![\sigma]\!].(\overline{X} \rhd Y \to \alpha) \in [\![\overline{x} \rhd M]\!] \text{ and} \\ & & \qquad \forall \beta \in Y.(\overline{X} \rhd \beta) \in [\![\overline{x} \rhd N]\!]\} \end{array}$$

We write $[\![M]\!]$ instead of $[\![\rhd M]\!]$ when M is closed. Since $\mathbf{ScottL}_!$ is cartesian closed, we have:

▶ **Theorem 4.** *If* $M^\sigma =_{\beta\eta} N^\sigma$, *then* $[\![\overline{x} \rhd M]\!] = [\![\overline{x} \rhd N]\!]$.

3.2 ScottL$_!$ as an intersection type system

As the notation suggests, an element $X \to \alpha \in [\![\sigma \to \tau]\!]$ can be seen as an implication. When X is a set $\{\beta_1, \dots, \beta_n\}$, it is natural to think of it as an intersection type $\beta_1 \wedge \dots \wedge \beta_n \to \alpha$.

More precisely, we rewrite a statement

$$(X_1, \dots, X_n \rhd \alpha) \in [\![x_1^{\sigma_1}, \dots, x_n^{\sigma_n} \rhd M^\sigma]\!],$$

where $X_i \in \mathcal{P}_f[\![\sigma_i]\!]$ and $\alpha \in [\![\sigma]\!]$, as a typing judgement

$$x_1^{\sigma_1} : X_1, \dots, x_n^{\sigma_n} : X_n \vdash M : \alpha.$$

The inductive definition of $[\![\overline{x} \rhd M]\!]$ may be rewritten as the following type inference rules:

$$\frac{\exists \alpha'.\ X_i \ni \alpha' \sqsupseteq_{\sigma_i} \alpha}{x_1^{\sigma_1} : X_1, \dots, x_n^{\sigma_n} : X_n \vdash x_i^{\sigma_i} : \alpha}\ (\text{var}) \qquad \frac{\Gamma, y^\sigma : Y \vdash M : \alpha}{\Gamma \vdash \lambda y^\sigma.M : Y \to \alpha}\ (\text{abs})$$

$$\frac{\Gamma \vdash M : X \to \alpha \quad \Gamma \vdash N : \beta \text{ for every } \beta \in X}{\Gamma \vdash MN : \alpha}\ (\text{app}), \quad \text{in particular} \quad \frac{\Gamma \vdash M : \emptyset \to \alpha}{\Gamma \vdash MN : \alpha}\ (\text{app})$$

$$\frac{}{\alpha \sqsupseteq_o \alpha} \qquad \frac{\forall \alpha \in X\ \exists \beta \in Y.\ \beta \sqsupseteq_\sigma \alpha}{Y \sqsupseteq_{!\sigma} X} \qquad \frac{Y \sqsupseteq_{!\sigma} X \quad \alpha \sqsupseteq_\tau \beta}{X \to \alpha \sqsupseteq_{\sigma\to\tau} Y \to \beta}$$

Since the type system comes from a cartesian closed category, typing is preserved under $\beta\eta$ equivalence. Also, $x^\sigma : X \vdash M^\tau : \alpha$, $Y \sqsupseteq_{!\sigma} X$ and $\alpha \sqsupseteq_\tau \beta$ imply $x : Y \vdash M : \beta$, since $[\![x \rhd M]\!]$ is a morphism in **ScottL$_!$**.

▶ Remark. The above type system is essentially the same as that of [18]. It is also very close to the essential type assignment of [21], although we only deal with *simply typed* lambda terms, and thus our intersection types always conform to the shape of simple types; for instance, we do not have a type like $p \wedge (q \to r)$. It is for this reason that our types are preserved under η expansion, which does not usually hold for intersection types. It should be noted that a reversed notation is very often used for subsumption: our $a \sqsupseteq b$ corresponds to $a \leqslant b$ of [21].

It has been observed by Carvalho [4] that **Rel** leads to a (useful) intersection type system. More recently, **ScottL$_!$** is presented as an intersection type system by Ehrhard, that is similar to ours but for a different purpose. It could be interesting to explore this connection between relation-based semantics and intersection type systems from a more general perspective. Notice that this connection is entirely different from that between intersection types and filter models (see, e.g., [22]).

3.3 Applications

Let us illustrate the use of **ScottL$_!$** and the companion intersection type system by three examples.

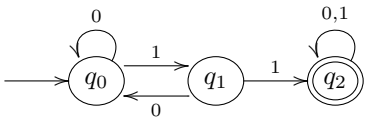
1. Suppose that $[\![o]\!] = Q = \{*\}$. Then both $\mathsf{True} := \{*\} \to \emptyset \to *$ and $\mathsf{False} := \emptyset \to \{*\} \to *$ belong to $[\![\mathbf{B}]\!]$. We have $\mathsf{True} \in [\![\mathbf{tt}]\!]$ but not $\mathsf{True} \in [\![\mathbf{ff}]\!]$. As a consequence:

▶ **Theorem 5.** *Let M be a closed term of type $\mathbf{B}$. Then $M =_{\beta\eta} \mathbf{tt}$ iff $\mathsf{True} \in [\![M]\!]$.*

In this way checking $\beta\eta$ equivalence boils down to checking membership in **ScottL$_!$**, or equivalently to type checking in the intersection type system.

2. Let $D = (Q, \{0,1\}, \delta, q_0, q_f)$ be a nondeterministic finite automaton (NFA) where Q is the set of states, $\delta \subseteq \{0,1\} \times Q \times Q$ is the transition relation, $q_0, q_f \in Q$ are respectively

(a)



(b)

$$\dfrac{\dfrac{F_1 \ni \{q_1\} \to q_2 \quad \dfrac{F_1 \ni \{q_0\} \to q_1 \quad \{q_0\} \ni q_0}{\Gamma \vdash f_1 : \{q_0\} \to q_1} \quad \dfrac{}{\Gamma \vdash x : q_0}}{\Gamma \vdash f_1 : \{q_1\} \to q_2} \quad \dfrac{}{\Gamma \vdash f_1 x : q_1}}{\dfrac{\Gamma \vdash f_1(f_1 x) : q_2}{\vdash \mathbf{11} : \mathsf{D}_0}}$$

Figure 2 (a) Finite automaton D_0. (b) A derivation (where $\Gamma = f_0 : F_0, f_1 : F_1, x : \{q_0\}$).

the initial and final states (we may assume w.l.o.g. that an NFA has exactly one final state). This can be expressed by an element

$$\mathsf{D} := X_0 \to X_1 \to q_0 \to q_f \quad \in [\![\mathbf{W}]\!]_Q,$$

where $X_0 = \{\{q_i\} \to q_j : (0, q_i, q_j) \in \delta\}$, and $X_1 = \{\{q_i\} \to q_j : (1, q_i, q_j) \in \delta\}$.

For instance, to the automaton $D_0 = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, q_2)$ described in Figure 2 (a) corresponds the element $\mathsf{D}_0 := F_0 \to F_1 \to \{q_0\} \to q_2$, where $F_0 = \{\{q_0\} \to q_0, \{q_1\} \to q_0, \{q_2\} \to q_2\}$ and $F_1 = \{\{q_0\} \to q_1, \{q_1\} \to q_2, \{q_2\} \to q_2\}$.

Corresponding to the run accepting the word 11, we have $\mathsf{D}_0 \in [\![\mathbf{11}]\!]$, i.e., $\vdash \mathbf{11} : \mathsf{D}_0$, as demonstrated by the derivation in Figure 2 (b); recall that $\mathbf{11} = \lambda f_0 f_1 x. f_1(f_1 x)$. Conversely, given a derivation for $\vdash \mathbf{w} : \mathsf{D}_0$, we can read off a successful run on w. Hence we have:

▶ **Theorem 6.** *Let D be a nondeterministic finite automaton and $w \in \{0,1\}^*$. Then D accepts w iff $\mathsf{D} \in [\![\mathbf{w}]\!]$. Hence if M is a closed term of type $\mathbf{W}$, D accepts the word represented by the normal form of M iff $\mathsf{D} \in [\![M]\!]$.*

▶ Remark. We can conversely show that every element of $[\![\mathbf{w}]\!]$ corresponds to an NFA accepting w. So the set $\bigcup_{Q:\text{finite}} [\![\mathbf{w}]\!]_Q$ can be thought of as the set of NFA's accepting w.

The same idea is behind Kobayashi's use of intersection types for model checking higher order recursion schemes [11]. Adapted to the current setting, it amounts to the following. Let M be a closed term of infinite tree type $\mathbf{T}^\omega := (o^2 \to o)^2 \to o$, involving the fixpoint operator μ. Then M possibly describes an infinite binary tree $\langle\!\langle M \rangle\!\rangle$ labelled with $\{0, 1\}$. For every tree automaton D with a trivial acceptance condition [1], there is an element $\mathsf{D} \in [\![\mathbf{T}^\omega]\!]$ such that $\langle\!\langle M \rangle\!\rangle$ is accepted by D iff $\mathsf{D} \in [\![M]\!]$.

The intersection type system of [11] is later expanded by [12] so that the new type system covers all the properties expressible by modal μ-calculus formulas. It would be interesting to see whether the latter can also be seen as a model of lambda calculus like **ScottL**$_!$.

3. Given a closed term N in η-long normal form, let $Q = \{\mathsf{L}_1, \ldots, \mathsf{L}_m\}$ be the set of symbols, each L_i corresponding to a subterm occurrence of base type o in N. We can then associate an element $\mathsf{M} \in [\![\sigma]\!]_Q$ to each subterm occurrence M^σ in N by induction on the structure of M. If $M = xK_1 \cdots K_j$ that is a subterm of $L_i^o = xK_1 \cdots K_n$ $(0 \le j < n)$, we define $\mathsf{M} := \{\mathsf{K}_{j+1}\} \to \cdots \{\mathsf{K}_n\} \to \mathsf{L}_i$. In particular, $\mathsf{x} := \{\mathsf{K}_1\} \to \cdots \{\mathsf{K}_n\} \to \mathsf{L}_i$. If $M = \lambda x.K$ and $x^{(1)}, \cdots, x^{(n)}$ are the occurrences of variable x in K, we define $\mathsf{M} := \{\mathsf{x}^{(1)}, \cdots, \mathsf{x}^{(n)}\} \to \mathsf{K}$.

For instance, consider the Kirstead terms:

$$K_1 := \lambda f.f(\lambda y.f(\lambda z.y)), \qquad K_2 := \lambda f.f(\lambda y.f(\lambda z.z)) \qquad : ((o \to o) \to o) \to o.$$

Let $F_1 := f(\lambda z.y), G_1 := f(\lambda y.f(\lambda z.y)), F_2 := f(\lambda z.z)$ and $G_2 := f(\lambda y.f(\lambda z.z))$. We have

$$\begin{aligned} \mathsf{K}_1 &:= \{\{\emptyset \to \mathsf{y}\} \to \mathsf{F}_1, \{\{\mathsf{y}\} \to \mathsf{F}_1\} \to \mathsf{G}_1\} \to \mathsf{G}_1, \\ \mathsf{K}_2 &:= \{\{\{z\} \to \mathsf{z}\} \to \mathsf{F}_2, \{\emptyset \to \mathsf{F}_2\} \to \mathsf{G}_2\} \to \mathsf{G}_2. \end{aligned}$$

We can verify that these elements distinguish K_1 from K_2, as $\vdash K_i : \mathsf{K}_j$ holds iff $i = j$. More generally, we have the following theorem, which is due to Salvati [18].

▶ **Theorem 7.** *Let N be a closed term of type σ in η-long normal form. For every closed term K of type σ, $K =_{\beta\eta} N$ iff $\mathsf{N} \in [\![K]\!]$.*

As a consequence, interpretation of lambda terms in **ScottL**$_!$ is *injective*:

▶ **Corollary 8.** *Let M and N be closed terms of the same type. Then $M =_{\beta\eta} N$ iff $[\![M]\!]_Q = [\![N]\!]_Q$ holds for all finite sets Q iff $[\![M]\!]_Q = [\![N]\!]_Q$ holds for an infinite set Q.*

Notice that injectivity also holds for **Set** [7], **Rel** and **Coh** (the category of coherent spaces and stable maps) [5].

4 Optimizations

4.1 Krivine type system

The type inference rules in the previous subsection suggest a natural alternating algorithm for type checking. For instance, the rule (app) says that $\Gamma \vdash MN : \alpha$ holds iff there *exists* a set X such that $\Gamma \vdash M : X \to \alpha$ and for *every* $\beta \in X$, $\Gamma \vdash N : \beta$. Unfortunately, the algorithm is not efficient enough. The reason is that, when applying the rule (app) bottom-up, we have to guess a set X, which is sometimes too big. To avoid a too big guess, we incorporate the mechanism of Krivine's abstract machine [15] into the type system.

Throughout this subsection, we fix a set Q, $r \in \mathbb{N}$, and only consider types and terms of order $\leq r+1$. First, for each simple type σ, let $CL(\sigma)$ be the set $\{(n, N) : n \in \mathbb{N},\ N : \sigma\}$ of *closures*. Next, notice that each simple type σ of order $\leq r+1$ can be uniquely written as $\sigma = \sigma_1 \to \cdots \sigma_k \to \tau$ with $Order(\sigma_k \to \tau) = r+1$ and $Order(\tau) \leq r$ ($k = 0$ if $Order(\sigma) \leq r$). We define the set $ST(\sigma)$ of *stacks* by:

$$ST(\sigma) := \{(n_1, N_1) \to \cdots (n_k, N_k) \to \alpha : (n_i, N_i) \in CL(\sigma_i) \text{ for } 1 \leq i \leq k \text{ and } \alpha \in [\![\tau]\!]\}.$$

An *environment* Γ is a list of the form $x_1^{\sigma_1} : C_1, \ldots, x_n^{\sigma_n} : C_n$ with $Order(\sigma_i) \leq r$ such that $C_i \in CL(\sigma_i)$ if $Order(\sigma_i) = r$ and $C_i \in CL(\sigma_i) \cup \mathcal{P}_f[\![\sigma_i]\!]$ if $Order(\sigma_i) < r$ for every $1 \leq i \leq n$. A *judgement* is of the form $\Gamma \vdash_r M : S$ with $M : \sigma$, $Order(\sigma) \leq r+1$ and $S \in ST(\sigma)$. Here we put subscript r to emphasize that the definition depends on the order r.

The following rules define the *Krivine type system*:

$$\frac{\exists \alpha'.\ X \ni \alpha' \sqsupseteq_\sigma \alpha}{\Delta, x^\sigma : X, \Sigma \vdash_r x^\sigma : \alpha}\ (\text{var } 1) \qquad \frac{\Gamma \vdash_r N : \alpha}{\Gamma, \Delta, x^\sigma : (\sharp\Gamma, N), \Sigma \vdash_r x^\sigma : \alpha}\ (\text{var } 2)$$

$$\frac{\Gamma, x^\sigma : X \vdash_r M : \alpha}{\Gamma \vdash_r \lambda x^\sigma.M : X \to \alpha}\ (\text{abs } 1) \qquad \frac{\Gamma, x^\sigma : (n, N) \vdash_r M : S}{\Gamma \vdash_r \lambda x^\sigma.M : (n, N) \to S}\ (\text{abs } 2)$$

$$\frac{\Gamma \vdash_r M : X \to \alpha \quad \Gamma \vdash_r N : \beta\ (\forall \beta \in X)}{\Gamma \vdash_r MN : \alpha}\ (\text{app } 1) \qquad \frac{\Gamma \vdash_r M : (\sharp\Gamma, N) \to S}{\Gamma \vdash_r MN : S}\ (\text{app } 2)$$

Note that the system is syntax-directed, in the sense that at most one rule can be applied bottom-up to each judgement.

The main purpose of the above system is to avoid guessing a set $X \in \mathcal{P}_f[\![\sigma]\!]$ when $Order(\sigma) = r$ in the case of application. We instead invoke a KAM computation. Judgements correspond to KAM states, and rules (var 2), (abs 2) and (app 2) correspond to KAM transition rules. Indeed, when read bottom-up, (app 2) forms a closure (Γ, N) and pushes it

to the stack, and (abs 2) pops a closure (Γ, N) from the stack and updates the environment with $x : (\Gamma, N)$. Finally, (var 2) looks up the value of x in the current environment.

The only difference is that our closure does not record an environment Γ itself, but only its length $\sharp\Gamma$. Due to our restricted use of KAM, the former can be recovered from the latter. Formally, let us call a judgement $\Gamma \vdash_r M : S$ *well-formed* when the following holds:

1. If $\Gamma = \Delta, x : (n, N), \Sigma$ then $n \leq \sharp\Delta$.
2. If (n, N) occurrs in S, then $n \leq \sharp\Gamma$.

One can immediately see that if a well-formed judgement is derivable, then it always has a derivation in which all judgements are well-formed.

The two derivations below, the left one in the previous system and the right one in the new system, illustrate how the KAM machinery works effectively (recall that $\Gamma \vdash N : \beta$ and $\beta \sqsupseteq \beta'$ imply $\Gamma \vdash N : \beta'$):

$$\dfrac{\dfrac{\dfrac{X \ni \beta \sqsupseteq \beta'}{\Gamma, x : X, \Delta \vdash x : \beta'} \quad \vdots}{\Gamma, x : X \vdash M : \alpha} \quad \vdots}{\Gamma \vdash \lambda x.M : X \to \alpha \qquad \Gamma \vdash N : \beta \ (\forall \beta \in X)} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash_r N : \beta'}}{\Gamma, x : (\sharp\Gamma, N), \Delta \vdash_r x : \beta'} \quad \vdots}{\Gamma, x : (\sharp\Gamma, N) \vdash_r M : \alpha}}{\Gamma \vdash_r \lambda x.M : (\sharp\Gamma, N) \to \alpha}$$

$$\Gamma \vdash (\lambda x.M)N : \alpha \qquad\qquad \Gamma \vdash_r (\lambda x.M)N : \alpha$$

▶ **Theorem 9.** *Let $M : \sigma$ be a closed term of order $r + 1$ and $\alpha \in [\![\sigma]\!]$. Then $\alpha \in [\![M]\!]$ iff $\vdash_r M : \alpha$ is derivable in the Krivine type system.*

Proof. Recall that $\alpha \in [\![M]\!]$ iff $\vdash M : \alpha$ holds in the intersection type system in Subsection 3.2. For the forward direction, we tentatively work in an intermediate system in which the definition of $ST(\sigma)$ is relaxed to $ST(\sigma) := \{C_1 \to \cdots C_n \to \alpha : C_i \in CL(\sigma_i) \cup \mathcal{P}_f[\![\sigma_i]\!] \text{ for } 1 \leq i \leq n \text{ and } \alpha \in [\![o]\!]\}$ for $\sigma = \sigma_1 \to \cdots \sigma_n \to o$. Suppose that $\Gamma \vdash N : \beta$ holds for every $\beta \in X$. We then have:

1. If $\Gamma, \Delta, x : X, \Sigma \vdash M : S$ is derivable in the intermediate system, then so is $\Gamma, \Delta, x : (\sharp\Gamma, N), \Sigma \vdash M : S$.
2. If $\Gamma, \Delta \vdash M : \overline{C} \to X \to S$ is derivable in the intermediate system (where $\overline{C}$ is a list of closures/intersection types), then so is $\Gamma, \Delta \vdash M : \overline{C} \to (\sharp\Gamma, N) \to S$.

We can now replace each instance of (app 1) in a derivation with that of (app 2). Indeed, if we have $\Gamma \vdash M : X \to S$ and $\Gamma \vdash N : \beta$ for every $\beta \in X$, then $\Gamma \vdash M : (\sharp\Gamma, N) \to S$ holds by 2 above, so $\Gamma \vdash MN : S$ holds by (app 2).

For the backward direction, we evaluate terms M in environments Γ. Given an environment Γ and $k \leq \sharp\Gamma$, we write Γ_k to denote its initial segment of length k, and $\hat{\Gamma}$ to denote Γ with all members of the form $x : (n, N)$ removed. We define evaluation $M[\Gamma]$ by:

$$M[\emptyset] := M, \qquad M[\Gamma, x : X] := M[\Gamma], \qquad M[\Gamma, x : (k, N)] := (M[\Gamma])[\ N[\Gamma_k]\ /x],$$

where $[N/x]$ denotes the standard substitution. We then have the following property.

- If a well-formed judgement $\Gamma \vdash_r M : (n_1, N_1) \to \cdots M : (n_k, N_k) \to \alpha$ is derivable in the Krivine type system, then $\hat{\Gamma} \vdash M[\Gamma]N_1[\Gamma_{n_1}] \cdots N_k[\Gamma_{n_k}] : \alpha$ is derivable in the previous type system.

The proof employs the fact that typing is preserved by β equivalence. ◀

4.2 Thin types

To give an efficient algorithm for problems $\textsc{Bool}(r)$ and $\textsc{RegLang}(r)$, it is useful to restrict the set of elements in $[\![\sigma]\!]$ (= intersection types) slightly. Let $\sigma = \sigma_1 \to \cdots \sigma_m \to o^k \to o$

with $\sigma_m \neq o$ (or $\sigma = o^k \to o$ and $m = 0$). An intersection type $\beta = X_1 \to \cdots X_{m+k} \to \alpha$ in $[\![\sigma]\!]$ is said to be *thin* if $\biguplus_{1 \leq i \leq k} X_{m+i}$ is either empty or a singleton, and $X_1, \ldots, X_m$ consist of thin types (whenever $m > 0$). The set of all thin elements in $[\![\sigma]\!]$ is denoted by $[\![\sigma]\!]^*$.

For instance, both $\mathsf{True} = \{*\} \to \emptyset \to *$ and $\mathsf{False} = \emptyset \to \{*\} \to *$ are thin, while $\{*\} \to \{*\} \to *$ is not. The type $\mathsf{D} \in [\![\mathbf{W}]\!]$ for a finite automaton D is also thin.

▶ **Lemma 10.** *Let σ be either $\mathbf{B}$ or $\mathbf{W}$. Then for every closed term $M : \sigma$ and $\alpha \in [\![\sigma]\!]^*$, $\vdash_r M : \alpha$ holds if and only if it has a thin derivation (i.e., a derivation which involves only thin types).*

Proof. We prove the lemma for the intersection type system in Subsection 3.2. We can then transform the derivation into that of the Krivine type system without introducing a non-thin type (see the proof of Theorem 9). For simplicity, we only consider the case $\sigma = \mathbf{B}$.

Suppose that $\vdash M : \mathbf{B}$ holds but it does not admit a thin derivation. By strong nomalization, we may assume that M has exactly one redex $(\lambda x_1.K)N_1$, and its reduct admits a thin derivation. Observe that $\lambda x_1.K$, which is responsible for non-thinness, is of type $o^k \to o$, and that substituting N_1 for the variable x_1 of type o does not create a new redex in K. As a consequence, M actually contains a generalized redex $(\lambda x_1^o \cdots x_k^o.K_0)N_1 \cdots N_l$ $(1 \leq l \leq k)$, and it reduces to a subterm of $\mathbf{tt}$ or $\mathbf{ff}$ after l steps of β reduction. Hence $K_0, N_1, \ldots, N_l$ are variables of type o. But then we can directly verify that $(\lambda x_1^o \cdots x_k^o.K_0)N_1 \cdots N_l$ admits a thin derivation, contradicting the assumption. ◀

▶ **Lemma 11.** *Let Q be a finite set such that $\sharp Q = q$, σ be a simple type of order $r \geq 1$ and of arity a. Then $\sharp [\![\sigma]\!]^*_Q \leq hyp(r-1, O(a^2q^2))$. Hence each element in $[\![\sigma]\!]^*_Q$ can be effectively coded by a string of length $hyp(r-2, O(a^2q^2))$.*

Proof. If $r = 1$, then σ is of the form $o^k \to o$. Since we only count thin types, $\sharp [\![\sigma]\!]^* = kq^2 + q \leq aq^2$. If $r = 2$ and $\sigma = \sigma_1 \to \cdots \sigma_k \to o$, then

$$\sharp [\![\sigma]\!]^* \;\leq\; 2^{\sharp [\![\sigma_1]\!]^* + \cdots + \sharp [\![\sigma_k]\!]^*} \cdot q \;\leq\; 2^{(a-1)(aq^2) + \log q} \;\leq\; 2^{a^2q^2}.$$

The rest can be calculated by induction. ◀

▶ Remark. If we naively interpret types in **Set**, the upper bound on the number of elements in $[\![\sigma]\!]$ would be something like: q^{q^a} for $Order(\sigma) = 1$, and $q^{q^{q^{a \log a}}}$ for $Order(\sigma) = 2$, that is too much sensitive to the arity a. This is the main reason why we work with $\mathbf{ScottL}_!$ rather than **Set**.

Restriction to thin elements only works for "unary" data types such as $\mathbf{B}$, $\mathbf{N}$ and $\mathbf{W}$; typically, it does not work for tree type $\mathbf{T} := (o^2 \to o)^2 \to o^2 \to o$, since tree automata cannot be expressed by thin types.

5 Complexity of evaluation

5.1 Order $2r + 3$

We are now ready to estimate the complexity of semantic evaluation in $\mathbf{ScottL}_!$.

▶ **Theorem 12.** *Let $M : \sigma$ be a closed term in $\Lambda(s, a, r+3)$, where σ is either $\mathbf{B}$ or $\mathbf{W}$ and $r \geq 0$. Let $\alpha \in [\![\sigma]\!]^*_Q$ and $q = \sharp Q$. Then it can be decided by an alternating algorithm in time $s^3 \cdot hyp(r, O(a^2q^2))$ whether $\vdash_{r+2} M : \alpha$ holds or not.*

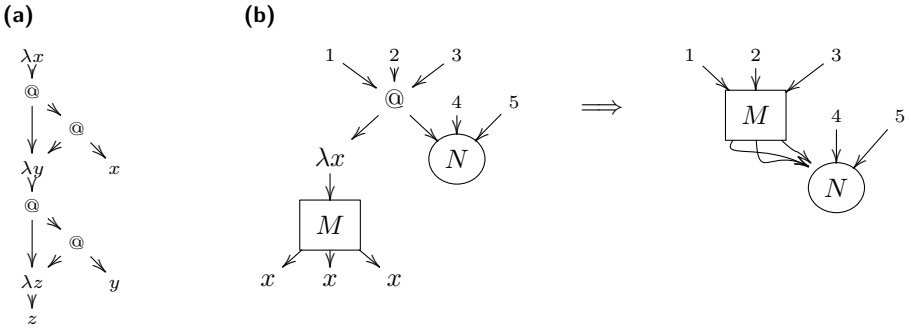


Figure 3 (a) A dag-representation of $\lambda x.(\lambda y.(\lambda z.z)((\lambda z.z)y))((\lambda y.(\lambda z.z)((\lambda z.z)y))x)$. (b) A graph rewriting rule corresponding to $(\lambda x.M)N \longrightarrow M[N/x]$.

Proof. We apply the natural alternating algorithm inherent to the definition of the Krivine type system, where stacks are built upon thin types.

It is clear that the height of any derivation is at most $s + ar$, where ar takes into account the height of subderivations for subsumption. Each step of applying a rule (bottom up) costs linear time in the size of a judgement. Hence it suffices to give an upper bound to the size of any judgement occurring in an (attempted) derivation of $\vdash_{r+2} M : \alpha$. Let

$$x_1^{\sigma_1} : C_1, \cdots, x_m^{\sigma_m} : C_m \vdash_{r+2} N^{\sigma} : C_{m+1} \to \cdots C_n \to \alpha$$

be such a judgement, where $\sigma = \sigma_{m+1} \to \cdots \sigma_n \to o$ and $\mathit{Order}(\sigma) \leq r + 3$. We have either $C_i \in \mathcal{P}_f[\![\sigma_i]\!]^*$ with $\mathit{Order}(\sigma_i) \leq r + 1$, or $C_i = (k, K) \in CL(\sigma_i)$. In the former case, C_i can be coded by a string of length $\mathit{hyp}(r, O(a^2q^2))$, while in the latter case the size is at most $2s$. Since n is bounded by $s + a$, the total size of the judgement is $(s + a) \cdot \max(\mathit{hyp}(r, O(a^2q^2)), 2s) + s = s^2 \cdot \mathit{hyp}(r, O(a^2q^2))$ (note that $a = O(s)$ by Lemma 2).

Therefore, the algorithm runs in time $s^3 \cdot \mathit{hyp}(r, O(a^2q^2))$. ◀

▶ **Corollary 13.** *Both* $\textsc{Bool}(2r + 3)$ *and* $\textsc{RegLang}(2r + 3)$ *belong to* r-EXPSPACE.

Proof. Let us consider $\textsc{Bool}(2r + 3)$. Given an (optimally typed) closed term $M : \mathbf{B}$ of size s and of order $2r + 3$, we compute $M' = \langle\!\langle M \rangle\!\rangle_{r+3}$, and check if $\vdash_{r+2} M' : \mathsf{True}$ with $Q = \{*\}$. The algorithm is correct since we have $M =_{\beta\eta} \mathbf{tt}$ iff $M' =_{\beta\eta} \mathbf{tt}$ iff $\mathsf{True} \in [\![M']\!]$ iff $\vdash_{r+2} M' : \mathsf{True}$ by Theorems 5 and 9.

The cost of computing M' is $\mathit{hyp}(r, O(s))$ and $M' \in \Lambda(\mathit{hyp}(r, s), a, r + 3)$ by Lemma 3. Type checking of $\vdash_{r+2} M' : \mathsf{True}$ costs alternating time $\mathit{hyp}(r, s)^3 \cdot \mathit{hyp}(r, O(a^2)) = \mathit{hyp}(r, \mathit{poly}(s))$ by Theorem 12. Hence the whole algorithm works in alternating time $\mathit{hyp}(r, \mathit{poly}(s))$, so in deterministic space $\mathit{hyp}(r, \mathit{poly}(s))$.

The problem $\textsc{RegLang}(2r + 3)$ can be decided similarly; this time the polynomial also depends on q, the number of states of the input automaton D (see Theorem 6). ◀

5.2 Order $2r + 2$

Let us now address the problems $\textsc{Bool}(2r + 2)$ and $\textsc{RegLang}(2r + 2)$. Here a more delicate space management is required. In particular, type checking has to be done with dag-representations of terms (see Figure 3(a)).

A term M is of *dag-size* s, if M can be represented as a directed acyclic graph which has s vertices. Let us denote by $\Lambda^{\mathsf{dag}}(s, a, r)$ the set of terms of dag-size $\leq s$, arity $\leq a$ and order $\leq r$. Lemma 3 can be refined as follows:

▶ **Lemma 14.** *Suppose that $M : \sigma$ and $Order(\sigma) \leq r$. If $M \in \Lambda(s, a, r+1)$, then $\langle\langle M \rangle\rangle_r \in \Lambda^{\mathsf{dag}}(s, a, r)$. Given an ordinary representation of M, a dag-representation of $\langle\langle M \rangle\rangle_r$ can be computed in time $O(s^2)$.*

Proof. Represent M as a tree, and successively apply the graph rewriting rule of Figure 3(b) (that corresponds to β reduction) to each "redex" of order $r+1$. The rule is sound, provided that (*) the indicated vertex λx (of order $r+1$) does not have any incoming edge other than the indicated one from @. This property (*) is indeed maintained during the whole process of reduction, since the reduction rule applied to a redex of order $r+1$ never creates a vertex of order $r+1$ with multiple incoming edges. Since the reduction strictly decreases the number of vertices, the computation terminates in s steps, so in time $O(s^2)$. ◀

Below, we suppose that the Krivine type system operates on terms represented as dags.

▶ **Theorem 15.** *Let $M : \sigma$ be a closed term in $\Lambda^{\mathsf{dag}}(s, a, r+3)$, where σ is either $\mathbf{B}$ or $\mathbf{W}$, and $r \geq 0$. Let $\alpha \in [\![\sigma]\!]$ and $q = \sharp Q$. Then it can be decided by an alternating algorithm in space $s^2 \cdot hyp(r, O(a^2q^2))$ whether $\vdash_{r+2} M : \alpha$ holds or not.*

Proof. It suffices to show that each judgement $\Gamma \vdash N : S$ occurring in the derivation has size $s^2 \cdot hyp(r, O(a^2q^2))$. The only delicate point is to show that the length of Γ is bounded by s, the *dag-size* of M. To see this, notice that we can identify a subterm of M with a vertex in the dag-representation of M. Then a thread of a derivation in the type system of Subsection 3.2 corresponds to a path in the dag. Now the length of an environment increases only when a vertex with label λx is visited, but the same vertex cannot be visited twice due to acyclicity. Hence it is bounded by s. The derivation can be then translated into one in the Krivine type system keeping the length bound (see the proof of Theorem 9). ◀

▶ **Corollary 16.** *Both* $\textsc{Bool}(2r+2)$ *and* $\textsc{RegLang}(2r+2)$ *belong to* r-EXPTIME.

Proof. Let us consider $\textsc{Bool}(2r+2)$. Since the case $r = 0$ can be easily decided by simple graph rewriting [19], we assume $r = r' + 1$, so that $2r + 2 = 2r' + 4$.

Given a closed term $M : \mathbf{B}$ of size s and of order $2r'+4$, we first compute $M' := \langle\langle M \rangle\rangle_{r'+4}$, then a dag representation of $M'' := \langle\langle M' \rangle\rangle_{r'+3}$, and check if $\vdash_{r'+2} M'' : \mathsf{True}$ with $Q = \{*\}$. The cost of computing M' is as before, while M'' can be computed in space $hyp(r', O(s^2))$, and $M'' \in \Lambda^{\mathsf{dag}}(hyp(r', s), a, r'+3)$ by Lemma 14. Type checking of $\vdash_{r'+2} M'' : \mathsf{True}$ costs alternating space $hyp(r', s)^2 \cdot hyp(r', O(a^2)) = hyp(r', poly(s))$ by Theorem 15. Hence the whole algorithm works in alternating space $hyp(r', poly(s))$, so in deterministic time $hyp(r, poly(s))$. ◀

▶ Remark. Theorem 15 nearly (though not exactly) conforms to the puzzle at the beginning of the introduction. The trick there is indeed that the arity is constant for all $M\mathbf{n}$ when M is fixed, so that the number $hyp(r, O(a^2q^2))$ in Theorem 15 becomes a constant. Hence the computation can be done in alternating space $O(s^2)$, that is in deterministic time $2^{O(s)}$.

Finally, let us discuss the problem $\textsc{Term}(r)$. Since we cannot assume that types are thin in general, the upper bounds on the number and size of intersection types get worse than estimated by Lemma 11. Namely, if σ is of order $r \geq 1$ and arity a, then $\sharp[\![\sigma]\!] \leq hyp(r, O(aq))$, and the size of each $\alpha \in [\![\sigma]\!]$ is $hyp(r-1, O(aq))$.

In addition, the Krivine type system $\vdash_r$ does not work when the order of the type of input terms M, N is greater than $r+1$. Hence natural complexity bounds are as follows.

▶ **Theorem 17.** *Under the assumption that the type of input terms M, N is of order $\leq r+1$:*
1. $\textsc{Term}(2r+1)$ *belongs to* r-EXPTIME.

2. $\textsc{Term}(2r+2)$ *belongs to* r-EXPSPACE.

It can be proved by employing Theorem 7; just notice that q is bounded by the size of the η-long normal form of N.

▶ Remark. Schubert [19] proved that $\textsc{Term}(2)$ is complete for P and $\textsc{Term}(3)$ is complete for PSPACE, which are sharper than our results. We leave it open whether we may extend the sharper bounds of [19] to higher order.

6 Hardness

6.1 Encoding of Turing machines

Let us outline a proof of the hardness part of Theorem 1. For orders 2 and 3, it can be easily proved by encoding Boolean circuits into order 2 terms, and quantified Boolean circuits into order 3 terms, respectively.

$$\begin{array}{lcll} \neg & := & \lambda bxy.byx & : \mathbf{B} \to \mathbf{B} \\ \wedge & := & \lambda b_1 b_2 xy.b_1(b_2 xy)y & : \mathbf{B}^2 \to \mathbf{B} \\ \forall & := & \lambda F.F\mathbf{tt} \wedge F\mathbf{ff} & : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \end{array}$$

Beyond order 3, we encode Turing machines with bounded time and space. In contrast to [8, 9, 14], which represent each TM by a single program, we just need to encode a TM by a *family* of lambda terms, one for each input length. A similar encoding was given by [16], but ours is more efficient with respect to the order (though less uniform). Encoding is even easier if we have product types. So we first outline an encoding with product types, and then explain how to eliminate them in the next subsection.

In the following, we write $\sigma \times \tau$ for a product type, $\langle M^\sigma, N^\tau \rangle^{\sigma\times\tau}$ for a pair, and $\pi_i(M^{\sigma_1\times\sigma_2})^{\sigma_i}$ for a projection $(i = 1, 2)$. The definition of order is extended by $\mathit{Order}(\sigma \times \tau) := \max(\mathit{Order}(\sigma), \mathit{Order}(\tau))$.

Define $\mathbf{N}_0 := \mathbf{N}$ and $\mathbf{N}_{r+1} := \mathbf{N}_r \to \mathbf{N}_r$, and let $\mathbf{N}_r(\sigma)$ be $\mathbf{N}_r$ with the base type o replaced by σ. Observe that $\mathit{Order}(\mathbf{N}_r(\sigma)) = r + 2 + \mathit{Order}(\sigma)$. A natural number $k = 2^n$ can be succinctly represented by $\mathbf{n2} : \mathbf{N}_r$ with $\mathbf{n} : \mathbf{N}_{r+1}$ and $\mathbf{2} : \mathbf{N}_r$, or by $\mathbf{2} \circ \cdots \circ \mathbf{2} : \mathbf{N}_r$ (n times) with $\mathbf{2} : \mathbf{N}_r$ (where $N \circ M := \lambda z.N(Mz)$ with z fresh). Hence the number $\mathit{hyp}(r, n)$ can be represented by

$$\mathbf{hyp}(r,n) := (\underbrace{\mathbf{2} \circ \cdots \circ \mathbf{2}}_{n \text{ times}})\ \underbrace{\mathbf{2} \cdots \mathbf{2}}_{r-1 \text{ times}} \quad : \mathbf{N}_0(\sigma)$$

for every type σ, where the highest order subterms are $\mathbf{2} \circ \cdots \circ \mathbf{2}$ ($k \le n$ times) of type $\mathbf{N}_{r-1}(\sigma)$, whose order is $r + 1 + \mathit{Order}(\sigma)$.

Let $\mathbf{B_n} := o^n \to o$ and $\mathbf{i} := \lambda x_1 \cdots x_n.x_i : \mathbf{B_n}$. We have $\mathbf{B} = \mathbf{B_2}$.

For every $r, n \ge 1$, define a type $\mathbf{Addr}_{r,n}$ (for *addresses*) of order r by:

$$\mathbf{Addr}_{1,n} := \mathbf{B} \times \cdots \times \mathbf{B}\ (n \text{ times}), \qquad \mathbf{Addr}_{r+1,n} := \mathbf{Addr}_{r,n} \to \mathbf{B}.$$

Each $\mathbf{Addr}_{r,n}$ represents the set of natural numbers below $\mathit{hyp}(r, n)$ in binary. For $r = 1$, $k < 2^n$ is represented by $\mathbf{k}_{1,n} := \langle \mathbf{i_1}, \cdots, \mathbf{i_n} \rangle : \mathbf{Addr}_{1,n}$, where $i_1 \cdots i_n$ is the binary representation of k. For $r = r' + 1$, $k < \mathit{hyp}(r, n)$ is represented by the closed normal term $\mathbf{k}_{r,n} : \mathbf{Addr}_{r,n}$ such that $\mathbf{k}_{r,n}\mathbf{i}_{r',n}$ gives the ith bit of k for every $i < \mathit{hyp}(r', n)$.

▶ **Lemma 18.** *For every* $r, n \ge 1$, *there are closed terms*

$$\mathbf{suc}_{r,n},\ \mathbf{pred}_{r,n} : \mathbf{Addr}_{r,n} \to \mathbf{Addr}_{r,n}, \qquad <_{r,n} : \mathbf{Addr}_{r,n} \times \mathbf{Addr}_{r,n} \to \mathbf{B}$$

of size $O(n^2)$ *and order* $2r$, *representing successor and predecessor modulo* $\mathit{hyp}(r, n)$ *and* $<$.

Proof. It is easy for $r = 1$. For $r = r' + 1$, we first build a term $G_{f,a_0} : \mathbf{B}$ with free variables $f : \mathbf{Addr}_{r,n}$ and $a_0 : \mathbf{Addr}_{r',n}$, which computes the carry to the a_0th bit when computing $f + 1$. The idea is to iterate the step function $F_{f,a_0} : \mathbf{B} \times \mathbf{Addr}_{r',n} \to \mathbf{B} \times \mathbf{Addr}_{r',n}$ defined by $F_{f,a_0} := \lambda ca.\ \mathbf{if}\ a <_{r',n} a_0\ \mathbf{then}\ \langle c \wedge f(a), \mathbf{suc}_{r',n}(a)\rangle\ \mathbf{else}\ \langle c, a\rangle$. We thus define $G_{f,a_0} := \pi_1(\mathbf{hyp}(r', n)\ F_{f,a_0}\ \langle \mathbf{1}, \mathbf{0}_{r',n}\rangle)$, and finally $\mathbf{suc}_{r,n} := \lambda f a_0.f(a_0) \oplus G_{f,a_0} : \mathbf{Addr}_{r,n} \to \mathbf{Addr}_{r,n}$. The highest order subterm is $\mathbf{hyp}(r', n) : \mathbf{N}_0(\mathbf{Addr}_{r',n} \times \mathbf{B})$, which is of order $r' + 1 + r' \le 2r$. $\mathbf{pred}_{r,n}$ and $<_{r,n}$ are defined similarly. ◀

▶ **Theorem 19.** *Let TM be a deterministic Turing machine with 1 tape and q states which works in space $hyp(r, n)$ and in time $hyp(r', n)$ with $1 \le r \le r'$. Then there exists a term $\mathbf{tm}_{r,r',n} : \mathbf{B}^n \to \mathbf{B}$ of order $r + r' + 2$ such that for every $w \in \{0, 1\}^n$, $\mathbf{tm}_{r,r',n}(\mathbf{w}_{1,n}) =_{\beta\eta} \mathbf{tt}$ iff TM accepts w. The term $\mathbf{tm}_{r,r',n}(\mathbf{w}_{1,n})$ can be constructed in time polynomial in n.*

Proof. Define types $\mathbf{Tape} := \mathbf{Addr}_{r,n} \to \mathbf{B_3}$ and $\mathbf{Config} := \mathbf{Tape} \times \mathbf{Addr}_{r,n} \times \mathbf{B_q}$. Each $\langle t, \mathbf{k}, \mathbf{i}\rangle : \mathbf{Config}$ represents the configuration with tape content t, head position k and state i. Note that $Order(\mathbf{Config}) = r + 1$.

With the help of $\mathbf{suc}_{r,n}$, $\mathbf{pred}_{r,n}$ and $<_{r,n}$, it is easy to represent the one-step transition of TM by a term $\mathbf{tran} : \mathbf{Config} \to \mathbf{Config}$ of size $O(n^2)$ and order $2r$.

Now the result of the computation after $hyp(r', n)$ steps can be computed by the term $\mathbf{tm}_{r,r',n} := \lambda x.\mathbf{output}(\mathbf{hyp}(r', n)\ \mathbf{tran}\ \mathbf{init}(x)) : \mathbf{B}^n \to \mathbf{B}$ where $\mathbf{init} : \mathbf{B}^n \to \mathbf{Config}$ and $\mathbf{output} : \mathbf{Config} \to \mathbf{B}$ are suitable terms for initialization and output extraction. $\mathbf{hyp}(r', n)$ is considered to be of type $\mathbf{N}_0(\mathbf{Config})$, thus of order $r' + 1 + Order(\mathbf{Config}) = r + r' + 2$. Since $r' \ge r$, this dominates the order $2r$ of $\mathbf{tran}$.

It remains to show that product types can be eliminated from the encoding with at most polynomial size overhead, that will be discussed in the next subsection. ◀

▶ **Corollary 20.**

1. *Both* $\textsc{Bool}(2r + 2)$ *and* $\textsc{RegLang}(2r + 2)$ *are hard for* r-EXPTIME.
2. *Both* $\textsc{Bool}(2r + 3)$ *and* $\textsc{RegLang}(2r + 3)$ *are hard for* r-EXPSPACE.

Proof. Let $r' = r$ for $\textsc{Bool}(2r + 2)$ and $r' = r + 1$ for $\textsc{Bool}(2r + 3)$. It is straightforward to reduce $\textsc{Bool}(r)$ to $\textsc{RegLang}(r)$. ◀

6.2 Elimination of product types

The elimination is based on two isomorphisms:

$$\sigma_1 \times \sigma_2 \to \tau \cong \sigma_1 \to \sigma_2 \to \tau, \qquad \tau \to \sigma_1 \times \sigma_2 \cong (\tau \to \sigma_1) \times (\tau \to \sigma_2).$$

For each type σ, we define a number $b(\sigma)$ (the *breadth*) and a list $\overline{\sigma}$ of length $b(\sigma)$ below. The ith member of the list $\overline{\sigma}$ will be denoted by $(\sigma)_i$.

$$\begin{array}{rclcrcl} b(o) & := & 1 & & \overline{o} & := & o \\ b(\sigma \to \tau) & := & b(\tau) & & \overline{\sigma \to \tau} & := & \overline{\sigma} \to (\tau)_1, \ldots, \overline{\sigma} \to (\tau)_m \\ b(\sigma \times \tau) & := & b(\sigma) + b(\tau) & & \overline{\sigma \times \tau} & := & \overline{\sigma}, \overline{\tau}, \end{array}$$

where $m = b(\tau)$, and $\overline{\sigma} \to (\tau)_i$ is a shorthand for $(\sigma)_1 \to \cdots (\sigma)_k \to (\tau)_i$ (with $k = b(\sigma)$).

For each term $M : \sigma$, we define a list $\overline{M}$ of length $b(\sigma)$ such that $(M)_i$ (the ith member) is of type $(\sigma)_i$. In the following, we assume that $b(\sigma) = m$ and $b(\tau) = k$.

$$\begin{array}{rclcrcl} \overline{x^\sigma} & := & x_1^{(\sigma)_1}, \ldots, x_m^{(\sigma)_m} & & \overline{\langle M, N\rangle} & := & \overline{M}, \overline{N} \\ \overline{\lambda x^\sigma.M^\tau} & := & \lambda\overline{x^\sigma}.(M)_1, \ldots, \lambda\overline{x^\sigma}.(M)_k & & \overline{\pi_1(M^{\sigma\times\tau})} & := & (M)_1, \ldots, (M)_m \\ \overline{M^{\sigma\to\tau}N^\sigma} & := & (M)_1\overline{N}, \ldots, (M)_k\overline{N} & & \overline{\pi_2(M^{\sigma\times\tau})} & := & (M)_{m+1}, \ldots, (M)_{m+k}, \end{array}$$

where $x_1, \ldots, x_m$ are new variables associated to x, $\lambda\overline{x^\sigma}.(M)_i$ and $(M_i)\overline{N}$ are respectively shorthands for $\lambda x_1 \cdots x_m.(M)_i$ and $(M_i)(N)_1 \cdots (N)_m$.

The translation preserves $\beta\eta$-equality:

▶ **Theorem 21.** *If $M : \sigma$ and $M =_{\beta\eta} N$, then $(M)_i =_{\beta\eta} (N)_i$ for every $1 \leq i \leq b(\sigma)$.*

Therefore, when applied to $\mathbf{tm}_{r,r',n}(\mathbf{u}_{1,n}) : \mathbf{B}$, the translated term returns the correct output. However, we have to be careful for size. Indeed, if $\mathbf{n} : \mathbf{N}(\sigma)$, we have $|(\mathbf{n})_i| = O(b(\sigma)^n)$. To avoid such an unexpected explosion, we define the *depth* $d(M)$ of each term M as follows:

$$\begin{array}{rclcrcl} d(x) & := & 1 & \quad d(\lambda x.M) & := & d(M) & \quad d(\langle M, N\rangle) := \max(d(M), d(N)) \\ & & & d(MN) & := & \max(d(M), d(N)+1) & \quad d(\pi_i M) := d(M). \end{array}$$

We can now estimate the size after translation.

▶ **Lemma 22.** *Let $M : \sigma$, $m = \max\{b(\tau) : \tau \in \mathit{Type}(M)\}$ and $d = d(M)$. Then $|(M)_i| \leq m^d \cdot |M|$ for every $1 \leq i \leq b(\sigma)$.*

For instance, $d(\mathbf{n}) = O(n)$ and this is the reason of the exponential explosion. It is important for our purpose that the order of association matters for iterated multiplications $\mathbf{2} \circ \cdots \circ \mathbf{2}$ (n times). Depending on whether we associate them to the right or to the left, the depth changes significantly:

$$d(\mathbf{2} \circ (\cdots (\mathbf{2} \circ (\mathbf{2} \circ \mathbf{2})))) = O(n), \qquad d((((\mathbf{2} \circ \mathbf{2}) \circ \mathbf{2}) \cdots) \circ \mathbf{2}) = O(1).$$

Hence if we choose the second one for the definition of $\mathbf{hyp}(r, n)$ that appears in $\mathbf{tm}_{r,r',n}$, then the depth stays constant (as far as r, r' are fixed), so that we obtain a term of polynomial size by translation. Therefore, Theorem 19 holds without product types.

7 Conclusion

We have studied the computational complexity of evaluating simply typed lambda terms of bounded order. As observed by [8, 9, 14], efficiency is achieved by mixing β reduction and semantic evaluation. In comparison with these previous works, the crux of our work lies in the use of $\mathbf{ScottL}_!$ rather than **Set**. Indeed, the former has led to an intersection type system, and thus provided a natural alternating algorithm. Moreover, it is flexible enough to incorporate Krivine's machinery and restriction to thin types. All these ideas combine into an optimal algorithm that comforms to the completeness results for the problems $\textsc{Bool}(r)$ and $\textsc{RegLang}(r)$, although there still remains a gap for $\textsc{Term}(r)$.

It should be mentioned that semantic evaluation is also quite effective in the setting of infinitary lambda calculus [1]. Incidentally, [11] refines [1] by replacing semantic evaluation in **Set** by intersection typing. This is strictly parallel to our shift from **Set** to $\mathbf{ScottL}_!$.

We believe that this way of using semantics for computation should be further developed, so let us conclude with a slogan: *better semantics, faster computation.*

Acknowledgements We would like to thank Masahito Hasegawa, Naohiko Hoshino and Shin'ya Katsumata for pointers to the literature, and anonymous referees for a lot of useful comments. This work is supported by JSPS KAKENHI 21700041.

References

1 Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

2 Arnold Beckmann. Exact bounds for lengths of reductions in typed lambda-calculus. *J. Symb. Log.*, 66(3):1277–1285, 2001.

3 Richard Blute and Philip Scott. Category theory for linear logicians. In *Linear Logic in Computer Science, LMS 316*. Cambridge University Press, 2004.

4 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, to appear.

5 Lorenzo Tortora de Falco. Obsessional experiments for linear logic proof-nets. *Mathematical Structures in Computer Science*, 13(6):799–855, 2003.

6 Thomas Ehrhard. The scott model of linear logic is the extensional collapse of its relational model. *Theor. Comput. Sci.*, to appear.

7 Harvey Friedman. Equality between functionals. In *Proceedings of Logic Colloquium'73, LNM 453*, pages 22–37, 1975.

8 Andreas Goerdt and Helmut Seidl. Characterizing complexity classes by higher type primitive recursive definitions, Part II. In *Proc. 6th IMYCS, LNCS 464*, pages 148–158, 1990.

9 Gerd G. Hillebrand and Paris C. Kanellakis. On the expressive power of simply typed and let-polymorphic lambda calculi. In *Proceedings of 11th LICS*, pages 253–263, 1996.

10 Michael Huth. Linear domains and linear maps. In *Proceedings of 9th MFPS, LNCS 802*, pages 438–453, 1993.

11 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of 36th POPL*, pages 416–428, 2009.

12 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of 24th LICS*, pages 179–188, 2009.

13 Lars Kristiansen. Complexity-theoretic hierarchies induced by fragments of Gödel's T. *Theory Comput. Syst.*, 43(3-4):516–541, 2008.

14 Lars Kristiansen and Paul J. Voda. Programming languages capturing complexity classes. *Nord. J. Comput.*, 12(2):89–115, 2005.

15 Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

16 Harry G. Mairson. A simple proof of a theorem of Statman. *Theor. Comput. Sci.*, 103(2):387–394, 1992.

17 Paul-Andre Mellies. Categorical semantics of linear logic. In *Interactive models of computation and program behaviour, Panoramas et Syntheses 27*. Soc. Math. de France, 2009.

18 Sylvain Salvati. On the membership problem for non-linear abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2):163–183, 2010.

19 Aleksy Schubert. The complexity of β-reduction in low orders. In *Proceedings of 5th TLCA, LNCS 2044*, pages 400–414, 2001.

20 Richard Statman. The typed lambda-calculus is not elementary recursive. *Theor. Comput. Sci.*, 9:73–81, 1979.

21 Steffen van Bakel. Intersection type assignment systems. *Theor. Comput. Sci.*, 151(2):385–435, 1995.

22 Steffen van Bakel. Strict intersection types for the lambda calculus. *ACM Comput. Surv.*, 43(3):20, 2011.

23 Glynn Winskel. A linear metalanguage for concurrency. In *Proceedings of 7th AMAST, LNCS 1548*, pages 42–58, 1998.

24 Glynn Winskel. Linearity and non linearity in distributed computation. In *Linear Logic in Computer Science, LMS 316*. Cambridge University Press, 2004.