BRICS

Basic Research in Computer Science

BRICS RS-96-46 Andreev & Soloviev: A Decision Algorithm for Linear Isomorphism of Types

A Decision Algorithm for Linear Isomorphism of Types with Complexity $Cn(log^2(n))$

Alexander E. Andreev
Sergei Soloviev

BRICS Report Series RS-96-46

ISSN 0909-0878 November 1996

Copyright © 1996, BRICS, Department of Computer Science
University of Aarhus. All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark

Telephone:	+45 8942 3360
Telefax:	+45 8942 3255
Internet:	BRICS@brics.dk

BRICS publications are in general accessible through World Wide Web and anonymous FTP:

`http://www.brics.dk/`
`ftp://ftp.brics.dk/pub/BRICS`
This document in subdirectory `RS/96/46/`

A Decision Algorithm for Linear Isomorphism of Types with Complexity $Cn(log^2(n))$.

Alexander E. Andreev
Department of Mechanics and Mathematics,
Moscow State University,
Moscow, 119899, Russia;
e-mail: andreev@matis.math.msu.su

Sergei Soloviev*
Computer Science Department,
Durham University, U.K.,
e-mail: Sergei.Soloviev@durham.ac.uk

November 1996

Abstract

It is known that ordinary isomorphisms (associativity and commutativity of "times", isomorphisms for "times" unit and currying) provide a complete axiomatisation for linear isomorphism of types. One of the reasons to consider linear isomorphism of types instead of ordinary isomorphism was that better complexity could be expected. Meanwhile, no upper bounds reasonably close to linear were obtained. We describe an algorithm deciding if two types are linearly isomorphic with complexity $Cn(log^2(n))$.

*The main part of this research was done while both authors were visiting the Computer Science Department of the University of Aarhus; the visits were funded by BRICS, a Centre of the Danish National Research Foundation, and the European CLICS grant (for the second author). The final version of this paper was done by S. Soloviev while employed by Durham University and funded by British ESPRC grant (on leave from S. Petersburg Institute for Informatics RAN).

1 Introduction

The problem of characterisation of isomorphism of types that holds in all models of certain system of typed lambda calculus is closely connected with mathematical semantics of datatypes [1], [2]. The problem allows many equivalent re-formulations, for example, a description of isomorphic objects in free closed categories of different classes: Cartesian Closed (CC) Categories, Symmetric Monoidal Closed (SMC) Categories, Biclosed Categories etc. A presentation of a free Closed Category is given by certain system of propositional calculus (with deductions as morphisms). Another (maybe, more familiar to computer scientists) is provided by lambda calculus.

To define the notion of isomorphism of types in a system of typed lambda calculus one needs a) the presence of functional types $A \to B$; b) a composition, for two terms of types $A \to B$ and $B \to C$ their composition is to be a term of the type $A \to C$; c) the terms $id : A \to A$ representing identity maps for every type A (usually, $\lambda x : A.x$); d)an equivalence relation on terms (which respects the composition).

Of course, these conditions look too abstract for the familiar systems of lambda calculus, but we just use an opportunity to present the problem in a more general setting. These conditions satisfied, one defines two types A, B to be isomorphic iff there are terms $t : A \to B$ and $s : B \to A$ such that the composite $t \cdot s$ is equivalent to $id : B \to B$ and the composite $s \cdot t$ to $id : A \to A$.

Suppose two types A, B are given and one would decide if they are isomorphic. The direct attempt based on the definition above will lead, in general, to the consideration of an infinite set of lambda terms living in the types $A \to B$ and $B \to A$. Usually, one is looking for the algorithms based on transformations of types without any explicit reference to lambda terms.

A complete axiomatization and decision algorithm for isomorphism of objects in free CC Categories was obtained in [3]. That result provided automatically a complete axiomatization and decision algorithm for the isomorphism of types in the First Order Typed Lambda Calculus with terminal object and surjective pairing. The precise formulation is the following:

Let $\sim$ be the equivalence relation on types generated by the following axioms

1. $A \sim A$;

2. $A \wedge B \sim B \wedge A$;

3. $A \wedge (B \wedge C) \sim (A \wedge B) \wedge C$;

4. $A \wedge I \sim A$;

5. $I \to A \sim A$;

6. $A \wedge B \to C \sim A \to (B \to C)$;

7. $A \to I \sim I$

8. $A \to B \wedge C \sim (A \to B) \wedge (A \to C)$

and rules

$$\frac{A \sim B}{C[A/X] \sim D[B/X]}(subst) \quad \frac{A \sim B}{B \sim A}(sym) \quad \frac{A \sim B \quad B \sim C}{A \sim C}(trans)$$

(here $[A/X]$ denotes substitution of A for type variable X).

The types A and B are isomorphic (in the above-mentioned calculus) iff $A \sim B$.

(In [3] a model-theoretic proof was given. A direct proof for typed lambda calculus was published in [4], where the applications to Computer Science were also discussed. Di Cosmo developed the method from [4] to obtain similar results for the Second Order Typed Lambda Calculus.)

Some weaker variants of isomorphism of types are of practical interest, the linear isomorphism of types in particular, because one can expect that the complexity will be reasonably low.

The linear isomorphism of types corresponds to the isomorphism of objects in free SMC category, and can also be described as the isomorphism of types in the system of lambda calculus which corresponds to intuitionistic multiplicative linear logic. (A description of this system can be found in [5], [6], [7], [8].)

In [7] it was shown that the subsystem of the axiom system above, consisting of the axioms 1)-6) (where $\wedge$ is understood as "times" and $\to$ as linear implication) with the same rules, defines an equivalence relation on types that coincides with the relation of linear isomorphism of types.

Of course, the use of linear logic or linear terms does not imply linear complexity of corresponding algorithms.

The deciding algorithms for the isomorphism in the First Order Typed Lambda Calculus with terminal object and surjective pairing used a reduction to some normal forms, which were, in general, subexponetially longer than the original types. In case of linear isomorphism there is no growth of the length. The main problem is (recursive) ordering of factors in the subformulas of the form $A_1 \wedge .... \wedge A_n$. More or less obvious algorithms have quadratic complexity. We propose an algorithm with complexity $C{\cdot}nlog^2(n)$.

2 The Algorithm

In this section we shall denote by $\sim$ the eqivalence relation on types defined by axioms (1)-(5) and the rules above. $\wedge$ can be understood as "times" , $\rightarrow$ as linear implication, and I is the unit of $\wedge$. The algorithm is based on the fact [7] that two types A, B are linearly isomorphic iff $A \sim B$ for this relation.

2.1 Regular Formulas

When low complexity bounds are considered, the form of presentation of information is quite important. Usually the types are presented by formulas of intuitionistic linear logic, i.e., by propositional formulas with two binary connectives $\wedge$ and $\rightarrow$ (and constant I). We shall use a kind of prefix notation (not exactly polish notation, because we prefer to have "$\wedge$" with a varying number of arguments; that does no harm because of the associativity axiom 3)). So, the formulas are defined inductively in the following way:

1. the symbols $X_1, ..., X_n, ...$(type variables) and the constant I are formulas;
2. if A, B are formulas then $(\rightarrow AB)$ is a formula;
3. if the $A_1...A_n$ are formulas ($n > 1$), then the $(\wedge A_1...A_n)$ is a formula.

Below we shall also use list notation in formulas with the agreement that the expressions like $(\wedge\Gamma), (\rightarrow\Gamma)$ when Γ contains just one member should be understood as that member itself ($\wedge$ and $\rightarrow$ should be omitted).

The syntactic axioms 1)-6) above that characterize the linear isomorphism of types can be replaced for this presentation by the following axioms:

(i) $A \sim A$ $(refl)$;

(ii) $(\wedge A_1 ... A_n) \sim (\wedge A_{\sigma(1)} ... A_{\sigma(n)})$ (com) where σ is a permutation of the set {1,...,n} $(n > 1)$;

(iii) $(\wedge\Gamma(\wedge\Delta)\Sigma) \sim (\wedge\Gamma\Delta\Sigma)$ (as) (with Γ, Δ, Σ being lists of formulas of appropriate length);

(iv) $(\wedge\Gamma I\Delta) \sim (\wedge\Gamma\Delta)$ (un) (with $\Gamma\Delta$ non-empty);

(v) $(\rightarrow IB) \sim B$ (un')

(vi) $\rightarrow A(\rightarrow BC) \sim \rightarrow (\wedge AB)C(cur)$.

The rules for $\sim$ in this syntax are still $(subst), (sym), (trans)$.

We shall write $\Rightarrow_k$ iff $A{\sim}B$ is derivable from an instance of the axiom labelled by k by single application of $(subst)$, i.e., by replacement of an occurrence of a left side of this axiom in A by the right side of the same axiom.

We shall write $A{\sim}_k B$ iff $A{\sim}B$ is derivable from the axiom labelled by k only (obviously, $A{\sim}_k B$ iff there exists a chain $A = A_1, A_2, ..., A_n = B$ such that $A_i{\Rightarrow}_k A_{i+1}$ or $A_{i+1}{\Rightarrow}_k A_i$).

We shall write $A{\sim}_{\rightarrow}B$ if it is derivable from reflexivity $(refl)$ and axioms $(un), (un')$ and (cur) and $A{\sim}_{com}B$, if only $(refl), (com)$ were used.

Now *regular formulas, regular $\wedge$-formulas and regular $\rightarrow$-formulas* are defined in the following way.

Definition 2.1

1. *the symbols $X_1, ..., X_n, ...$(type variables) and the constant I are regular formulas;*

2. *if A is a regular formula different from I and B is a regular $\wedge$-formula, variable or I then $\rightarrow AB$ is a regular $\rightarrow$-formula;*

3. *if each of $A_1, ..., A_n$ ($n > 1$) is a variable or regular $\rightarrow$-formula, then $(\wedge A_1...A_n)$ is a regular $\wedge$-formula;*

4. *regular $\wedge$-formulas and regular $\rightarrow$-formulas are regular formulas.*

Let A be a (sub)formula. We shall call its 1-extension any (sub)formula of the form $(\rightarrow \Gamma A \Delta)$, or $(\wedge \Gamma A \Delta)$ with Γ and/or Δ non-empty.

Remark 2.2 *(i) A regular formula is $\wedge$- or $\rightarrow$-regular, except the case when it is a variable or the constant I. (ii) All subformulas of a regular formula are regular formulas. (iii) All regular $\wedge$- subformulas of a regular formula A are maximal in the following sense: their 1-extensions (in A) are $\rightarrow$-subformulas (not $\wedge$-subformulas). (iv) All regular $\rightarrow$- subformulas of a regular formula A are maximal in the following sense: their 1-extensions (in A) are $\rightarrow$-subformulas (not $\rightarrow$-subformulas)*

Lemma 2.3 *For every formula A, there is a unique regular formula B such that $A \sim_{\rightarrow} B$.*

Proof. Consider the system of formula-reductions $\Rightarrow_{as}, \Rightarrow_{un}, \Rightarrow_{un'}, \Rightarrow_{cur}$ (an occurrence of the left side of corresponding axiom is to be replaced by the right side). By straightforward induction on the structure of formulas one shows that a formula is in normal form iff it is regular. Now (trivial check) a) the system is Church-Rosser's and b) terminating (since each step decreases the number of $\rightarrow, \wedge$ or I).

Denote by $R(A)$ the regular formula corresponding to A.

Lemma 2.4 *If $A \Rightarrow_k B$ where k is $(as), (un), (un'), (cur)$, then $R(A) = R(B)$. If $A \Rightarrow_k B$ where k is (com), then $R(A) \sim_{com} R(B)$.*

Proof by induction on the length of reduction sequence from A to $R(A)$ in the system of reductions described above.

As an immediate consequence of the two lemmas, we have

Lemma 2.5 *$A \sim B$ iff $R(A) \sim_{com} R(B)$.*

Let us call the length $l(A)$ of the formula A the number of occurrences of variables, I, $\wedge$ and $\rightarrow$ in A.

Lemma 2.6 *If $A \sim_{com} B$ then $l(A) = l(B)$.*

2.2 Computational Model

Our computational model is random access machine. The cell size is $O(log(n))$, where n is the length of the formulas that are considered. The addresses of cells are $1, 2, \ldots$.

For our goals we have to use a special presentation of formulas. As it is easily seen, with every occurrence of $\wedge, \rightarrow$, variable or I in a formula A a unique subformula of A is connected. This occurrence will be called the main symbol of a subformula.

Initially, let us represent each occurrence of $\wedge, \rightarrow$, variable or I (and corresponding subformula) in a more standard way, using the quadruples, described below. (We do not need to represent brackets.) It is supposed that each such quadruple is encoded in a certain way and occupies one cell of RAM. The address of a subformula is understood as the address in RAM-memory of its main symbol.

The quadruple, representing a symbol, is defined in the following way:

Definition 2.7

- *the first member is the symbol itself (one may think, that it is represented numerically in some way);*
- *the second member is* 0 *for* I *or a variable and the address of the left-most subformula, if the symbol is* $\wedge$ *or* $\rightarrow$*;*
- *the third member is* 0 *for* I *or a variable and the address of the last subformula, if the symbol is* $\wedge$ *or* $\rightarrow$ *;*
- *the fourth member is the address of the next subformula in the 1-extension of the subformula, corresponding to the considered symbol. If there is no next subformula, it is* 0.

The formula is presented in RAM-memory by collection of quadruples representing its symbols (except brackets). Let $[v]$ denote the address of the cell corresponding to an occurrence v.

Example 2.8 *Consider the formula* $\rightarrow_1 a \rightarrow_2 bc$ *(i.e.,* $a \rightarrow_1 (b \rightarrow_2 c)$ *in standard notation). Its presentation is*

$$\{\rightarrow, [a], [\rightarrow_2], 0, \}; \{a, 0, 0, [\rightarrow_2]\}; \{\rightarrow, [b], [c], 0\}; \{b, 0, 0, [c]\}; \{c, 0, 0, 0\}.$$

If we put the quadruiples in the nodes of the syntactic tree of our formula (direction of pointers shown by arrows), we'll have

$$\begin{tikzcd}
& \{\to, [a], [\to_2], 0, \} \arrow[dl] \arrow[dr] & & \\
\{a, 0, 0, [\to_2]\}^{\to} & & \{\to, [b], [c], 0\} \arrow[dl] \arrow[dr] & \\
& \{b, 0, 0, [c]\}^{\to} & & \{c, 0, 0, 0\}
\end{tikzcd}$$

Here $[\to_1]$ would be the address of the first, and $[\to_2]$ is the address of the third quadruple.

The length $l(A)$ of a (sub)formula is the number of cells in this presentation.

It is supposed that the cells, occupied by different formulas, are always disjoint.

Elementary operations are comparisons of quadriples and their members and standard arithmetical and control operations of RAM. Input information of the algorithms considered below consists of the addresses of the main symbols of processed formulas.

The following lemma is obvious.

Lemma 2.9 *Syntactical identity of two formulas A, B can be checked in time $O(min(l(A), l(B)))$.*

Lemma 2.10 *There is an algorithm transforming the presentation of an arbitrary formula A in the presentation of $R(A)$ with an upper time bound $C(l(A))$ for some constant C.*

Proof proceeds by induction on the structure of A and is standard, with one modification due to our use of multiple $\wedge$.

When the formula $\wedge A_1 ... A_n$ is considered, the algorithm should check if $n = 2$ or $n > 2$ (obviously, in constant number of steps).

If $n = 2$ then induction proceeds in ordinary way, if $n > 2$ then the inductive hypothesis (and the algorithm) should be applied to A_1 and $\wedge A_2 ... A_n$, and then in constant number of steps one can "assemble" the presentation of the regular form of the formula $\wedge A_1 ... A_n$. (One does not change registers, only pointers.)

Definition 2.11 *Let us call by extended presentation of a formula it presentation where each quadriple is extended to the quintiple, with the fifth symbol being the length of the subformula, whose main symbol is represented by the quadriple.*

Lemma 2.12 *There is an algorithm transforming the presentation of a formula into its extended presentation in less than $C(l(A))$ steps for some constant C.*

Proof. Consider the following trip. Start in the root of the tree, and go from node to node according to the following procedure: a) if there is a leftmost subformula (node) not yet visited, go there; b) if it was already visited, go to the next subformula the in 1-extension; c) if in case b) the node corresponds to the last subformula, go to the node, representing the main symbol of the 1-extension (one can keep "return address"), or stop, if you come to the root.

Obviously, each node is visited exactly twice, and (with a constant number of extra-operations in each node) one can keep count the number of symbols visited before coming to each node.

Then, if in the first visit this number is written in the 5-th position, one can replace it by the difference at the second visit, which will be the length of the subtree corresponding to the subformula.

2.3 Strongly Regular Formulas

Let us define inductively a relation $\leq$ on regular formulas. ($A < B$ is understood as $A \leq B$ and not $A{\sim_{com}}B$.) Remember that if $A \sim_{com} B$, they have the same length.

Definition 2.13

- *If $l(A) < l(B)$ then $A < B$. If $l(A) = l(B)$, then:*
- *For variables and constant I,*

$$I < a_1 < \ldots < a_n < \ldots.$$

- *If A is a regular $\wedge$-formula and B is a regular $\rightarrow$-formula, then $A < B$;*

- *If A and B are the regular $\rightarrow$-formulas,*

$$A = (\rightarrow A_1A_2),\ \ B = (\rightarrow B_1B_2),$$

and

$$A_1 < B_1$$

or

$$A_1 \sim_{com} B_1,\ \ A_2 \leq B_2$$

then $A \leq B$;

- *If A and B are regular $\wedge$-formulas,*

$$A = (\wedge A_1A_2...A_n),\ \ B = (\wedge B_1B_2...B_n),$$

and there exist bijections φ and ψ

$$\varphi\{1,2,...,m\} \rightarrow \{1,2,...,m\},$$

$$\psi\{1,2,...,r\} \rightarrow \{1,2,...,r\}$$

and the number k,

$$1 \leq k \leq min(m,r),$$

such that

$$A_{\varphi_1} \leq A_{\varphi_2} \leq ... \leq A_{\varphi_m},\ \ B_{\psi_1} \leq B_{\psi_2} \leq ... \leq B_{\psi_r}$$

$$A_{\varphi_1} \sim_{com} B_{\psi_1}\ \ A_{\varphi_2} \sim_{com} B_{\psi_2}, ..., A_{\varphi_{k-1}} \sim_{com} B_{\psi_{k-1}},\ \ A_{\varphi_k} < B_{\psi_k}$$

then $A < B$.

Lemma 2.14 *For every regular formula A, B, always $A \leq B$ or $B \leq A$. If $A \leq B$ and $B \leq A$, then $A \sim_{com} B$. (That is, the relation $\leq$ is linear order on $\sim_{com}$-equivalence classes of regular formulas.)*

Proof by induction on $l(A)$, taking into account that both $A \leq B$ and $B \leq A$ can hold only if A and B are both of the same type (variables, constants, $\wedge$- or $\rightarrow$-formulas), and have equal length.

Now we are in the position to define strongly regular formulas.

Definition 2.15 *A regular formula is strongly regular iff in each its subformula of the form $\wedge A_1 ... A_n$ there is $A_1 \leq ... \leq A_n$.*

Lemma 2.16 *For every regular formula A there exists exactly one strongly regular formula $S(A)$ in the $\sim_{com}$-equivalence class of A.*

Proof. An $S(A)$ can be obtained by ordering $\wedge$-subformulas of A, that is, using $\wedge_{com}$. By induction on the structure of A (using definition of $\leq$) we show uniqueness.

As an immediate consequence of this lemma and lemma 2.5, we have the following theorem:

Theorem 1 $A \sim B \iff S(R(A)) = S(R(B))$.

3 Complexity

Lemma 3.1 *There is an algorithm deciding if $A \leq B$ or $B \leq A$ for strongly regular formulas A, B in time bound by Cn, where $n = min(l(A), l(B))$.*

Proof by induction on n, taking into account that syntactic identity of two regular formulas can be checked in time proportional to (the minimum of) their lengths (2.9).

Theorem 2 *The complexity of construction of the strongly regular form for any regular formula A is at most*

$$O(1) l(A) \log^2(l(A)).$$

Proof. The formula is taken in its extended presentation. The addresses of the tuples, representing symbols, do not change (only pointers do).

We prove our theorem by induction on the formula length. If the formula is a variable of I, then the bound is, evidently, true.

Assume that there exist some constants C_0, C_1 such that for any formula with $l(A) \leq T$ one can construct (the extended presentation of) its strongly regular form in time at most

$$C_0 T \log^2 T - C_1.$$

Consider A with the length equal to $T+1$. Of course, $1 \leq T$. Let $L(A)$ denote complexity of constructiong the strongly regular form of A.

Case. $A = \rightarrow BD$.

We have to convert to strongly regular forms the subformulas B, D. The process has the following steps:

- conversion of B to the strongly regular form;
- computing of the address of the first symbol of D;
- conversion of D to the strongly regular form.

As we immediately see, there exists a constant C_2, such that

$$L(A) \leq L(B) + L(D) + C_2.$$

By induction hypothesis,

$$\begin{aligned} L(A) &\leq C_0 l(B) log^2(l(B)) - C_1 + C_0 l(D) log^2(l(D)) - C_1 + C_2 \\ &\leq C_0 l(A) log^2(l(A)) - 2C_1 + C_2. \end{aligned}$$

If $C_1 > C_2$, then we have

$$L(A) \leq C_0 l(A) log^2(l(A)) - C_1.$$

Case. $A = (\wedge A_1 ... A_m)$.

Let $Q(i)$ denote the following triple: (num_i, len_i, adr_i), where

- $num_i = i$;
- $len_i = l(A_i)$;
- adr_i is the address of the main symbol of the formula A_i.

At the first step we compute the number m and the array

$$\mathbf{Q} = Q(1)Q(2)...Q(m),$$

at the same time converting each A_i to the strongly regular form. The complexity of this process is at most

$$C_3 + \sum_{i=1}^{m} (C_0 l(A_i) log^2(l(A_i)) - C_1 + C_3).$$

At the second step we make sorting of the array **Q**. After this sorting we have on the i-th place the sequence $Q(\phi(i))$, where ϕ is a bijection

$$\phi : \{1, ..., m\} \to \{1, ..., m\},$$

such that

$$i \leq j \Rightarrow len_{Q(\phi(i))} \leq len_{Q(\phi(j))}.$$

Complexity of such sorting is at most

$$C_4 m log(m).$$

At the third step we compute the number r and the sequence of integers

$$s_0, ..., s_r$$

such that

$$1 = s_0 < s_1 < ... < s_r = m,$$
$$len_{Q(\phi(s_i))} = len_{Q(\phi(s_i)+1)} = ... = len_{Q(\phi(s_{i+1}-1))},$$
$$len_{Q(\phi(s_{i+1}-1))} < len_{Q(\phi(s_{i+1}))},$$
$$i = 0, 1, ..., r-1.$$

Complexity of this step is at most

$$C_5 m,$$

for some constant C_5.

At the fourth step we are sorting each array

$$Q(\phi(s_i))Q(\phi(s_i)+1)\dots Q(\phi(s_{i+1}-1)),$$

if it is nontrivial, i.e.,

$$s_{i+1} - s_i > 1.$$

The order of the new array should correspond to our ordering of formulas of the same length. Complexity of this sorting is at most

$$C_6 len_{Q(\phi(s_i))} log(s_{i+1} - s_i)$$

for some constant C_6, because the comparison of two strongly regular formulas of equal length has complexity $O(n)(3.1)$.

After this step we have an array

$$Q(\psi(1))...Q(\psi(m)),$$

such that

$$i \leq j \Rightarrow A_{num_\psi(i)} \leq A_{num_\psi(j)}.$$

At the fifth step we have to make permutation of the subformulas in the new order. The complexity of this step is at most

$$C_7 m$$

for some constant C_7. At this step we change only pointers (address labels) in the tuples, representing the main symbols of $A_1, ..., A_m$, but do not move any tuples. (If we would move the subformulas, the bound would not be true.)

Let $l^*(A) = \sum_{i, l(A_i) \leq (1/2) l(A)} l(A_i)$. We have

$$\sum_{i=0}^{r-1} len_{Q(\phi(s_i))} log(s_{i+1} - s_i) \leq l^*(A) log(m),$$

since in our formula there may be at most one subformula with more than half of the whole formula length. This fact implies also

$$m log(m) \leq l^*(A) log(m).$$

In accordance with the bounds (...) we have, that

$$L(A) \leq C_3 + \sum_{i=1}^{m} (C_0 l(A_i) log^2(l(A_i)) - C_1 + C_3) + C_8 l^*(A) log(m),$$

for some constant C_8.

Suppose that

$$2C_3 \leq C_1.$$

Then we have

$$L(A) \leq \sum_{i=1}^{m} C_0 l(A_i) log^2 l(A_i) - C_1 + C_8 l^*(A) log(m) \leq$$

$$C_0 (l^*(A) log^2 \frac{l(A)}{2} + (l(A) - l^*(A)) log^2 l(A)) -$$

$$-C_1 + C_8 l^*(A) log(m) \leq$$
$$\leq C_0(l^*(A)(log^2 l(A) - logl(A)) + (l(A) - l^*(A))log^2 l(A)) -$$
$$-C_1 + C_8 l^*(A) log(m) =$$
$$C_0 l(A) log^2 l(A) - C_0 l^*(A) log(l(A)) - C_1 + C_8 l^*(A) log(m).$$

Now, we can suppose that

$$C_8 \leq C_0,$$

and then

$$L(A) \leq C_0 l(A) log^2 l(A) - C_1,$$

so that, with the choice of constants as above, our theorem is true. In other words, C_0, C_1 should be chosen in such a way, that they make the conditions (...) true.

Acknowledgements

We would like to thank Michael Rittri and Roberto Di Cosmo for many stimulating discussions of the matter, and Glynn Winskel and Uffe Engberg for helpful attention to our work while at BRICS.

References

[1] M. Rittri. Retrieving library functions by unifying types modulo linear isomorphism. *Proceedings of Conference on Lisp and Functional Programming*, 1992.

[2] R. DiCosmo. Isomorphism of Types: from λ-calculus to information retrieval and language design. Birkhauser, 1995.

[3] S.V. Soloviev. The category of finite sets and cartesian closed categories. *Zapiski Nauchnych Seminarov Leningradskogo Otdelenya Matematicheskogo Instituta im.V.A.Steklova AN SSSR*, 105:174-194, 1981 (English translation in: *Journal of Soviet Mathematics*, 22(3):1387-1400, 1983).

[4] K. Bruce, R. Di Cosmo and G. Longo. Provable isomorphism of types. Preprint LIENS-90-14, Ecole Normale Superieure, Paris, 1990

[5] G.E. Mints. Closed categories and Proof Theory.*J.Soviet Math.*, 15:45-62, 1981.

[6] G.-Y. Girard, Y. Lafont. Linear logic and lazy computation. In: Proc. TAPSOFT 87 (Pisa), v.2, p.52-66, Lecture Notes in Comp. Sci. v. 250 , 1987.

[7] S.V. Soloviev. A complete axiom system for isomorphism of types in closed categories. *Lecture Notes in Artificial Intelligence*, v. 698 (1993), 380-392.

[8] P.H. Benton, G.M. Bierman, V.C.V. de Paiva and J.M.E. Hyland. A term calculus for Intuitionistic Linear Logic. In *Proceedings of Typed Lambda calculus and Applications*, Lecture Notes in Comp. Sci. v. 664, 1993.

Recent Publications in the BRICS Report Series

RS-96-46 Alexander E. Andreev and Sergei Soloviev. *A Deciding Algorithm for Linear Isomorphism of Types with Complexity $Cn(log^2(n))$*. November 1996. 16 pp.

RS-96-45 Ivan B. Damgård, Torben P. Pedersen, and Birgit Pfitzmann. *Statistical Secrecy and Multi-Bit Commitments*. November 1996. 30 pp.

RS-96-44 Glynn Winskel. *A Presheaf Semantics of Value-Passing Processes*. November 1996. 23 pp. Extended and revised version of paper appearing in Montanari and Sassone, editors, *Concurrency Theory: 7th International Conference*, CONCUR '96 Proceedings, LNCS 1119, 1996, pages 98–114.

RS-96-43 Anna Ingólfsdóttir. *Weak Semantics Based on Lighted Button Pressing Experiments: An Alternative Characterization of the Readiness Semantics*. November 1996. 36 pp. An extended abstract to appear in the proceedings of the *10th Annual International Conference of the European Association for Computer Science Logic*, CSL '96.

RS-96-42 Gerth Stølting Brodal and Sven Skyum. *The Complexity of Computing the k-ary Composition of a Binary Associative Operator*. November 1996. 15 pp.

RS-96-41 Stefan Dziembowski. *The Fixpoint Bounded-Variable Queries are PSPACE-Complete*. November 1996. 16 pp. Presented at the *10th Annual International Conference of the European Association for Computer Science Logic*, CSL '96.

RS-96-40 Gerth Stølting Brodal, Shiva Chaudhuri, and Jaikumar Radhakrishnan. *The Randomized Complexity of Maintaining the Minimum*. November 1996. 20 pp. To appear in a special issue of *Nordic Journal of Computing* devoted to the proceedings of SWAT '96. Appears in Karlson and Lingas, editors, *Algorithm Theory: 5th Scandinavian Workshop*, SWAT '96 Proceedings, LNCS 1097, 1996, pages 4–15.