Math. Struct. in Comp. Science (2008), *vol.* 18, *pp.* 753–793. © 2008 Cambridge University Press
doi:10.1017/S096012950800683X Printed in the United Kingdom

On the building of affine retractions

ALEKSY SCHUBERT[†]

Institute of Informatics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
Email: alx@mimuw.edu.pl

Received 29 November 2006; revised 21 February 2008

A simple type σ is retractable to a simple type τ if there are two terms $C^{\sigma\to\tau}$ and $D^{\tau\to\sigma}$ such that $D \circ C =_{\beta\eta} \lambda x.x$. The retractability of types is affine if the terms C and D are affine, that is, when every bound variable occurs in them at most once in the scope of its declaration. This paper presents a system that derives affine retractability for simple types. It also studies the complexity of constructing these affine retractions. The problem of affine retractability is NP-complete even for the class of types over a single type atom and having limited functional order. In addition, a polynomial algorithm for types of orders less than three is presented.

1. Introduction

The notion of isomorphism, which renders the idea of identicality, appears very frequently in many formal theories. In the simply typed lambda calculus it is defined as follows: two types σ, τ are isomorphic if there exist terms C of the type $\sigma \to \tau$ and D of the type $\tau \to \sigma$ such that $D \circ C =_{\beta\eta} \lambda x.x$ and $C \circ D =_{\beta\eta} \lambda x.x$. This notion has been studied since the 1980s – see, for example, Bruce and Longo (1985). A complete and effective characterisation of the relation is given in Di Cosmo (1995). This notion has also been studied for calculi with richer types – see, for example, Fiore *et al.* (2002) and Palsberg and Zhao (2001). In the untyped lambda calculus, it is not possible to study the isomorphism of types. Still, one can consider lambda terms that behave as isomorphisms, that is, such that they have inverses. Characterisations of these terms were already known in the 1970s (Dezani-Ciancaglini 1976).

The relation of type isomorphism in the case of the simply typed lambda calculus has already been successfully applied. The first notable application is in tools that support the searching of software libraries (Di Cosmo 1995; Rittri 1991), and efficient procedures for accomplishing this task have been reported (Considine 2000; Zibin *et al.* 2003). Another area of application arises in tools that suggest possible corrections in programs that do not typecheck (McAdam 2002).

The notion of retraction is a generalisation of isomorphism. In this case only $D \circ C =_{\beta\eta} \lambda x.x$ is required to hold instead of the two equalities mentioned earlier. This category theory based definition corresponds to a concept of coding – everything that is encoded by means of C can be decoded back by D without any loss of information.

[†] This work was partly supported by the Polish Government grant no 3 T11C 002 27 and a fellowship of The Foundation for Polish Science (FNP).

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

The subject of type retractions is undeveloped even for very simple formalisms, such as the simply typed lambda calculus. Although there is a complete and effective criterion for retractability in the calculus with respect to β-reduction (Spławski and Urzyczyn 1999), there is still no characterisation of this kind for retractability with respect to the semantically more interesting $\beta\eta$-reduction. There are a few sufficient conditions for $\beta\eta$-reduction based retractions in de'Liguoro *et al.* (1992), which also includes a complete characterisation of *affine retractions* in the lambda calculus with a single type atom, where the word *affine* means that each bound variable in terms C, D is used at most once (however, this characterisation is not effective in the sense that it does not lead to an algorithm). Padovani (2001) contains an algorithm that decides full $\beta\eta$ retractability in the simply typed lambda calculus with types that have a single type atom. Regnier and Urzyczyn (2001) gives characterisations of $\beta\eta$ retractability in the lambda calculus with many type atoms, but these characterisations are either not effective or not complete. Still, they give a complete characterisation of affine retractability, which implies the decidability of the problem.

Type retractions (but in a richer type system) have already been used to prove that recursive polymorphic types cannot be β-encoded in the polymorphic lambda calculus $\lambda 2$ (Spławski and Urzyczyn 1999). Type retractions may, in addition, turn out to be useful as a method for finding more general operations in software libraries – in a similar way to the previously mentioned applications of type isomorphisms (Rittri 1991). By the Curry–Howard isomorphism, retractions can also be used in automated provers as a vehicle supporting the reuse of already proved subproblems. Each of these applications requires an effective method for generating a link between the existing infrastructure (a software library, a library of proved lemmas) and the new requirement (a function to be found, a new formula to be proved). This link is provided by the terms C, D, mentioned in the definition of retraction. This paper presents a system for inferring *affine retractability* together with the accompanying terms C and D.

We can justify the restriction to affine retractions in three ways:

1 The type retraction problem can be defined in terms of the higher-order matching problem (a more detailed description can be found in Section 3.1). The higher-order matching problem is known to be non-elementarily hard (Vorobyov 1997). Moreover, the construction by Padovani in Padovani (2001), which employs a special, and known to be non-elementary, case of the higher-order matching problem suggests that the problem of finding retractions has high computational complexity (however, there is no proof of this fact). This suggests that the general solution may have high complexity. In fact, a similar kind of simplification has been used in the work on suggesting corrections in non-typing programs (McAdam 2002).

2 The type retractions may be used as a help in a proof or in a computer program development in a similar way to the type isomorphisms. However, the automatically generated terms of this kind are often subject to further manual modification by a human. In this light, it is more effective to supply the human with affine terms, which are less complicated and thus easier to understand and correctly modify.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

type atoms \ order	1	2	3	4	5	7
any	$O(n)$	$O(n)$	$O(n^6)$	NP-comp.		
k>1	$O(1)$	$O(n)$	$O(n^6)$	?	NP-comp.	
k=1	$O(1)$	$O(n)$	$O(n^6)$	?	?	NP-comp.

Fig. 1. A summary of the results on the complexity of deciding retractability. The columns give the results for various orders, and the rows the results for different numbers of type atoms: for unbounded number of atoms (any); for a fixed number of atoms ($k > 1$); and for a single atom ($k = 1$). The question marks indicate open cases.

3 The terms witnessing the type isomorphisms are actually affine, so it is natural to look for characterisations of retractions that are also affine.

As far as the complexity results are concerned, we can use general procedures for the higher-order matching when we are interested in solutions with a bounded number of variable occurrences (Wierzbicki and Dougherty 2002; Levy *et al.* 2006). These procedures already give a non-deterministic polynomial procedure for solving retractions (together with the simple reduction presented in Section 3.1). Dowek has also shown (Dowek 1993) that the second-order matching for linear terms has complexity $O(n)$. This, combined with the reduction to the higher-order matching problem presented later in the current paper, easily implies the linear complexity of the affine type retractions problem.

The current paper contains a proof that the relation of the affine retractability is polynomial for types of order at most 3. This provides a tight bound on the tractability of the problem as we prove here that the problem is already NP-complete for types of order 4. The latter result holds for the class of types with an unbounded number of atoms. We also present the complexity bounds when the number of type atoms is bounded by a fixed number $k > 1$ – the problem is NP-complete for types of order 5 and we have a polynomial procedure for types of order 3. Furthermore, if there is only one type atom, the problem is NP-complete for types of order 7, but we do not know the exact complexity for types of orders 4, 5 and 6. We also present very simple procedures for orders 1 and 2, which both run in time $O(n)$ (and can even be reduced to $O(1)$ and $O(n)$, respectively, when the number of type atoms is fixed). These results are summarised in Figure 1.

This paper is structured as follows. The basic notation is introduced in Section 2. Section 3 contains an exposition of the system to infer the retraction terms together with its foundational properties. The polynomial cases are discussed in Section 4. All the NP-completeness results mentioned in Figure 1 are proved in Section 5. We conclude with a discussion in Section 6.

This paper is a revised version of a preliminary conference paper (Schubert 2004).

2. Basic definitions

We assume familiarity with the simply typed lambda calculus, so we will only sketch the basic definitions to settle the notation. A more rigorous introduction can be found, for example, in Barendregt (1992).

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

We write $\{a_1, \ldots, a_n\}$ to denote a set or a multiset with the elements $a_1, \ldots, a_n$. Sequences are written $\langle a_1, \ldots, a_n \rangle$.

The simply typed lambda calculus $\lambda_{\rightarrow}$ (in the Church fashion) has types of the form

$$\mathscr{T} ::= \mathscr{B} \mid T \rightarrow T$$

where $\mathscr{B}$ is a non-empty set of type variables. We usually write type variables as a, b, and so on. while the types are written as σ, τ, and so on. The terms of the calculus have forms

$$x^{\sigma} \qquad \text{or} \qquad (M^{\sigma \rightarrow \tau} N^{\sigma})^{\tau} \qquad \text{or} \qquad (\lambda x^{\sigma}.M^{\tau})^{\sigma \rightarrow \tau}.$$

The λ operator binds the variable x^{σ} in the body of M^{τ} above, so we have to deal with the notions of free and bound variables, together with the α equivalence relation. We identify α-equivalent expressions. We use $\mathrm{FV}(M)$ to denote the set of variables free in a term M. We extend the notation to pairs, sequences, multisets and sets of terms in a natural way. In the version of the simply typed lambda calculus we use here, the types are part of the syntax, so we do not need the type inference rules. We will usually omit at least part of the type information when it does not introduce any essential value and is clear from the context. We use $M[x := N]$ to denote the usual capture avoiding substitution when we want to indicate that we substitute N on all the free occurrences of x in M. We use a sequence notation $M[x_i := S_i]_{i=k}^{l}$ to denote $M[x_k := S_k, \ldots, x_l := S_l]$. The notation C[] denotes a context with a single occurrence of a variable []. The term C[M] is C[] with the only occurrence of [] replaced by M. This replacement operation is not capture avoiding, so some variables in M may get bound as a result of the operation.

We will use several of the usual notational conventions for the presentation of terms. We write $\sigma_1 \rightarrow \sigma_2 \rightarrow \cdots \rightarrow \sigma_n$ to denote the type $\sigma_1 \rightarrow (\sigma_2 \rightarrow (\cdots \rightarrow \sigma_n) \cdots)$. We write $\lambda x_1 \ldots x_n.M$ for $\lambda x_1 . \ldots \lambda x_n.M$ and $M N_1 \ldots N_n$ for $(\ldots (M N_1) \ldots N_n)$. We also use a non-standard notation:

$$\begin{array}{lll}
M(N_i^{\sigma_i})_{i \in I} & \text{for } M N_1^{\sigma_1} \ldots N_n^{\sigma_n} & \text{when } I \text{ is a sequence } \langle n_1, \ldots, n_k \rangle \\
M(N_i^{\sigma_i})_{i=k}^{l} & \text{for } M N_k^{\sigma_k} \ldots N_l^{\sigma_l} & \\
\lambda(x_i^{\sigma_i})_{i \in I}.M & \text{for } \lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n}.M & \text{when } I \text{ is a sequence } \langle n_1, \ldots, n_k \rangle \\
\lambda(x_i^{\sigma_i})_{i=k}^{l}.M & \text{for } \lambda x_k^{\sigma_k} \ldots x_l^{\sigma_l}.M\,. &
\end{array}$$

We say that a term is *affine* whenever each variable (bound or free) occurs in it at most once. For instance, the terms $\lambda f^{a \rightarrow a} x^a.x^a$ and $\lambda f^{a \rightarrow a} x^a.z^b$ are affine, but the terms $\lambda f^{a \rightarrow a} x^a.f^{a \rightarrow a}(f^{a \rightarrow a} x^a)$ and $\lambda f^{a \rightarrow a} x^a.g^{a \rightarrow a}(g^{a \rightarrow a} x^a)$ are not.

In a term $\lambda(x_i)_{i=1}^{n}.z(M_i)_{i=1}^{p}$, the symbol z is called the *head* symbol or the *head of the term.*

If $\sigma = \sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow a$ or $\sigma = a$, then $\mathit{head}(\sigma) = a$. All the types $\sigma_1, \ldots, \sigma_n$ are called *arguments* of σ, and the type a is called the *result* type or the *head* type. The size $|\sigma|$ of a type σ is defined inductively as follows:

— $|a| = 1$ for $a \in \mathscr{B}$,
— $|\sigma_1 \rightarrow \sigma_2| = |\sigma_1| + |\sigma_2| + 1$.

In particular, the size of both the type a and b is 1, and the size of $a \rightarrow b \rightarrow c$ is 5. We sometimes use the notation $\Delta \rightarrow a$, where Δ is a multiset or a sequence of types $\delta_1, \ldots, \delta_n$, to denote $\delta_1 \rightarrow \cdots \rightarrow \delta_n \rightarrow a$. We can use this notation because the type retractability

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

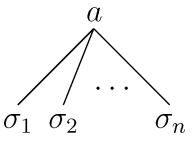


Fig. 2. A schematic representation of $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to a$

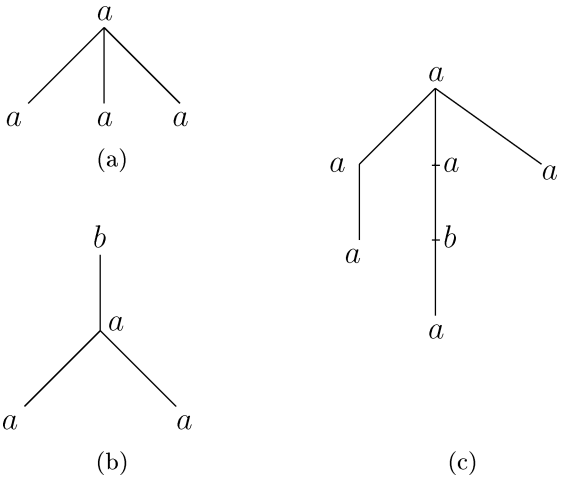


Fig. 3. A schematic representation of the types (a) $a \to a \to a \to a$ (b) $(a \to a \to a) \to b$, and (c) $(a \to a) \to ((a \to b) \to a) \to a \to a$

is invariant under type isomorphisms (de'Liguoro *et al.* 1992), which corresponds to argument permutation.

The types are categorised by an order. The order is defined as

— $\mathrm{ord}(a) = 1$ for $a \in \mathscr{B}$ and
— $\mathrm{ord}(\sigma \to \tau) = \max(\mathrm{ord}(\sigma) + 1, \mathrm{ord}(\tau))$.

In particular, both types a and b have order 1, the type $a \to a \to a \to a$ has order 2, the type $(a \to a \to a) \to b$ has order 3 and the type $(a \to a) \to ((a \to b) \to a) \to a \to a$ has order 4. The set of types having order k is denoted $\mathrm{T}^k_{\to}$. We also adopt a method for representing types in figures that is more convenient in dealing with type retractions. The idea of the representation is presented in Figure 2. In this representation a type $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to a$ is shown as a tree with the root labelled with a (the result type) and n children such that the tree starting at the i-th child represents the corresponding argument type σ_i. Examples of types in this representation are presented in Figure 3. This notation is borrowed from Regnier and Urzyczyn (2001).

The simply typed lambda calculus is accompanied by a notion of evaluation. This is called β-reduction. This relation is generated by the basic rule

$$(\lambda x^{\sigma}.M)N \to_{\beta} M[x := N]$$

where $[x := N]$ denotes the capture avoiding substitution. This relation is extended by the η-reduction reduction generated by $(\lambda x^{\sigma}.Mx) \to_{\eta} M$, where x does not occur freely in

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

M. The union of the two relations is denoted by $\rightarrow_{\beta\eta}$. We also use $\rightarrow^*_{\beta\eta}$ for the reflexive transitive closure of $\rightarrow_{\beta\eta}$, and $=_{\beta\eta}$ for the least equivalence containing $\rightarrow_{\beta\eta}$. The $\beta\eta$ reduction in the simply typed lambda calculus is strongly normalisable, that is, for any term M, each reduction of the term using the relation is finite. Moreover, there is only one result of such a reduction, which is called the *normal form*. We write $\mathrm{NF}(M)$ to denote the normal form of M.

We say that a redex MN is the *leftmost-outermost* in $\mathrm{C}[MN]$ when there is no redex in $\mathrm{C}[]$ that occurs to the left of []. In this paper we use the notion of the *leftmost-outermost* strategy in the $\beta\eta$ reduction. In this strategy, we choose the leftmost-outermost redex at each step. In the leftmost-outermost reduction

$$M \rightarrow_{\beta\eta} M_1 \rightarrow_{\beta\eta} \cdots \rightarrow_{\beta\eta} M_n,$$

the first term M_i of the form $\lambda x_1 \ldots x_k.yN_1 \cdots N_m$ is called the *head normal form* of M.

3. Retracts and their properties

Definition 1 (Retracts). We say that a type σ is a *retract* of a type τ, and write $\sigma \trianglelefteq_{\beta\eta} \tau$, if and only if there exists a pair of terms $C^{\sigma\rightarrow\tau}$ and $D^{\tau\rightarrow\sigma}$ such that $D \circ C =_{\beta\eta} I_\sigma = \lambda x^\sigma.x$. We say that a type σ is an *affine retract* of a type τ, written $\sigma \trianglelefteq^1_{\beta\eta} \tau$, if and only if C, D are affine terms (that is, terms with at most one occurrence of each bound variable). We often say that these C, D *witness* the (affine) retractability of σ and τ. We usually omit the subscripts $\beta\eta$ and write $\trianglelefteq$ or $\trianglelefteq^1$.

In order to illustrate the idea, we will now give a few examples of retracts, beginning with some simple cases.

Example 1. Consider the second-order types

$$\sigma = a \rightarrow a \qquad \tau = a \rightarrow b \rightarrow a\,.$$

Terms C, D that witness the retractability in this simple case are

$$C = \lambda x^\sigma u^a_1 u^b_2.xu_1 \qquad D = \lambda y^\tau v^a_1.yv_1c^b$$

where c^b is an arbitrary constant of the type b.

To give a feeling for how we can obtain terms C and D, we will briefly analyse the possibilities for the head symbol of C. In general, the head symbol of C can be one of:

— x;
— a free variable;
— u_1.

In the first case, there is a pair of retracts. In the second case, however, the normal form of $D \circ C$ would have the same free variable as the head symbol, so this case is impossible. In the third case, the variable x could not occur in the normal form of $D \circ C = \lambda x^\sigma u^\sigma_1.u_1$. Hence, the only possibility for constructing C is to start it with x as the head symbol.

Example 2. Consider the third-order types

$$\sigma = (a \rightarrow a) \rightarrow a \rightarrow a, \qquad \tau = (a \rightarrow b \rightarrow a) \rightarrow b \rightarrow a \rightarrow a\,.$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Terms C, D that witness the retractability in this simple case are

$$\begin{aligned} C &= \lambda x^{\sigma} u_1^{a\to b\to a} u_2^b u_3^a . x\big(\lambda z_1^a . u_1 z_1 c^b\big) u_3 \\ D &= \lambda y^{\tau} v_1^{a\to a} v_2^a . y\big(\lambda z_1^a z_2^b . v_1 z_1\big) c^b v_2 . \end{aligned}$$

Using β-reduction, the term $\lambda x.D(Cx)$ reduces to

$$\lambda x^{\sigma} v_1^{a\to a} v_2^a . x\big(\lambda z_1^a . v_1 z_1\big) v_2 . \tag{1}$$

It is easy to verify that this term η reduces to $\lambda x.x$. Note that u_1 cannot be the head symbol of C. This is because the only way to obtain $\lambda z_1^a . v_1 z_1$ as the first argument of x in (1) is to use $\lambda z_1^a z_2^b . v_1 z_1$ as the first argument of y in D. This, however, means that we cannot have u_1 as the head symbol of C since then the head symbol of the β-normal form of $\lambda x.D(Cx)$ would be v_1 (in the notation of (1)).

In general, for the retracts relating types of order less than or equal to 3, the head symbol of the term C is x (using the notation above). This is proved later as Proposition 4.1.

Example 3. Now consider an example of a fourth-order retraction:

$$\begin{aligned} \sigma &= a_1 \to a_2 \to a_3 \to a_4 \to a \\ \tau &= \tau_1 \to a_4 \to a_3 \to a \\ &= ((a_1 \to a_2 \to a) \to (a_1 \to a_3 \to a) \to a) \to a_4 \to a_3 \to a . \end{aligned}$$

Terms C, D that witness the retractability are

$$\begin{aligned} C &= \lambda x^{\sigma} u_1^{(a_1\to a_2\to a)\to(a_1\to a_3\to a)\to a} u_2^{a_4} u_3^{a_3} . u_1\big(\lambda z_1^{a_1} z_2^{a_2} . x z_1 z_2 u_3 u_4\big)\big(\lambda z_1^{a_1} z_2^{a_3} . c^a\big) \\ D &= \lambda y^{\tau} v_1^{a_1} v_2^{a_2} v_3^{a_3} v_4^{a_4} . y\big(\lambda z_1^{a_1\to a_2\to a} z_2^{a_1\to a_3\to a} . z_1 v_1 v_2\big) v_4 v_3 . \end{aligned}$$

It is worth stressing that the use of the first argument τ_1 of τ to encode a_1, a_2 blocks the possibility of using the argument to encode the argument a_3 of σ in τ_1. Intuitively, we can use this phenomenon to encode, for example, the problem of Hamiltonian circuits. This can be done by using the type atoms (such as a_1, a_2, a_3, a_4) as vertices and types similar to τ_1 as encodings of all the edges that go out of a particular vertex (in our example, $a_1 \to a_2 \to a$ would encode the edge from a_1 to a_2 and $a_1 \to a_3 \to a$ the edge from a_1 to a_3). In our proof in Section 5, we use a reduction of the 3-SAT problem, but the idea is essentially the same.

This encoding does not work when we have a limited number of atoms since then the number of vertices is fixed. That is why the NP-hardness of the fourth-order case when the number of atoms is fixed cannot use a similar technique. However, we can encode the case with an arbitrary number of vertices by using types of the form $a^k \to b^l \to a$, where a^k means k arguments a and b^l means l arguments b, but at the cost of higher order.

Example 4. Finally, consider an example of a non-affine retraction:

$$\begin{aligned} \sigma &= a \to a \to a \to a \\ \tau &= ((a \to a \to a) \to (a \to a \to a) \to a) \to a . \end{aligned}$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Terms C, D that witness the retractability are

$$C = \lambda x^{\sigma} u_1^{(a\to a\to a)\to(a\to a\to a)\to a}.u_1\big(\lambda z_1^a z_2^a.u_1\big(\lambda z_3^a z_4^a.xz_1z_2z_3\big)L\big)K$$
$$D = \lambda y^{\tau} v_1^a v_2^a v_3^a.y\big(\lambda z_1^{a\to a\to a} z_2^{a\to a\to a}.z_2(z_1v_1v_2)(z_1v_3v_3)\big)$$

where $K = \lambda xy.x$ and $L = \lambda xy.y$. These terms are indeed non-affine since C contains two occurrences of u_1 and D two occurrences of z_1 and v_3. Moreover, the pair σ, τ has no affine retractions, which we will prove in Example 8.

The problem of finding (affine) retracts is defined as follows.

Definition 2 (Problem of (affine) retractions).
Input: Types σ, τ.
Question: Is there a pair of (affine) terms $C^{\sigma\to\tau}$ and $D^{\tau\to\sigma}$ such that $D \circ C =_{\beta\eta} I_\sigma$?

3.1. *Connection with other problems*

The retractions problem is related to two other problems: the higher-order matching problem and the subtyping problem.

3.1.1. *The higher-order matching problem* We are interested here in the following question.

Definition 3 (Higher-order matching problem).
Input: An equation $M = N$ where M, N are simply typed lambda terms.
Question: Is there a substitution S such that $S(M) =_{\beta\eta} N$?

When there exists a substitution S such that $S(M) =_{\beta\eta} N$, we say that the equation $M = N$ is *solvable*.

Example 5. Consider the higher-order matching equation

$$\lambda x^{a\to a\to a}.x(X^{(a\to a\to a)\to(a\to a\to a)\to a}KL)(X^{(a\to a\to a)\to(a\to a\to a)\to a}LK) = \lambda x^{a\to a\to a}.xc_1^a c_2^a$$

where $K = \lambda xy.x$, $L = \lambda xy.y$ and X, c_1, c_2 are pairwise different free variables. This equation has a solution where

$$X := \lambda y_1^{a\to a\to a} y_2^{a\to a\to a}.y_1c_1c_2\,.$$

However, the equation

$$\lambda x^{a\to a\to a}.x(X^{((a\to a)\to a\to a)\to a}(\lambda f^{a\to a} x^a.fx))(X(\lambda f^{a\to a} x^a.f(fx))) = \lambda x.xc_1^a c_2^a$$

has no solution. Intuitively, the only way to obtain different variables c_1, c_2 based on the same term substituted for X would be to use different arguments, but the terms substituted on f can, roughly speaking, either be equivalent to I_a or to a term that starts with the same variable.

Recently, Stirling (Stirling 2006) presented a new algorithm that solves instances of the higher-order matching problem (when the types have only one type atom), which, hopefully, answers the decidability question. Moreover, the results of Wierzbicki and Dougherty (Wierzbicki and Dougherty 2002) and Schmidt-Schauß (Schmidt-Schauß 2004)

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

$$\text{(C)} \qquad C \cup \{\sigma \preceq \tau\} \vdash \sigma \preceq \tau$$

$$\text{(R)} \qquad C \vdash \sigma \preceq \sigma$$

$$\text{(T)} \qquad \frac{C \vdash \sigma \preceq \tau, \qquad C \vdash \tau \preceq \gamma}{C \vdash \sigma \preceq \gamma}$$

$$\text{(Arrow)} \qquad \frac{C \vdash \sigma \preceq \tau, \qquad C \vdash \rho \preceq \gamma}{C \vdash \tau \to \rho \preceq \sigma \to \gamma}$$

Fig. 4. The subtype inference system

imply that the problem is decidable when we constrain the substitution S to substitute the affine terms only. These results can be applied to the type retractions problem as the problem can be reduced to the higher-order matching problem using the following theorem.

Theorem 3.1 (From retractions to higher-order matching). For each pair of simple types σ, τ there exists a higher-order matching equation $M = N$, the size of which is proportional to the size of σ, τ such that the equation is solvable if and only if $\sigma \trianglelefteq \tau$.

Moreover, if the solution substitutes affine terms, then $\sigma \trianglelefteq^1 \tau$ holds.

Proof. Let σ, τ be simple types. Consider an equation $\lambda x^\sigma.X^{\tau\to\sigma}(Y^{\sigma\to\tau}x) = \lambda x^\sigma.x$. Suppose we have a solution S that substitutes X by D and Y by C. By Definition 3, this means that

$$\lambda x^\sigma.X^{\tau\to\sigma}(Y^{\sigma\to\tau}x)[X := D, Y := C] = \lambda x^\sigma.D(C)x =_{\beta\eta} I_\sigma.$$

The final equivalence means that $D \circ C =_{\beta\eta} I_\sigma$, which is exactly the condition from Definition 1 for C and D to witness the retractability.

The converse follows immediately from the definition of the retraction.

When C, D are affine they also witness the affine retractability. □

This simple theorem immediately implies that the affine retractions problem reduces in logarithmic space to the higher-order matching problem.

3.1.2. *The subtyping problem* The inequalities $\trianglelefteq$ and $\trianglelefteq^1$ are also related to the subtyping problem. The subtyping problem in the classical approach is defined as follows. Suppose we have a set C of type inequalities on type atoms that have the form $a \preceq b$. The inequalities on the atoms usually come equipped with coercions that are inclusions. These inclusions can be lifted to the higher-order types. As it is reasonable here to follow the idea of containment, the system must result from a semantical inclusion of type representations (Mitchell 1988). Based on this, we obtain a system to infer the subtype inequality of compound types using the rules in Figure 4 (Lincoln and Mitchell 1992).

The type retractions problem is similar to the problem of checking the subtyping relation in that both problems rely on the principle that there must be a transformation from the smaller type to the bigger one (the coder C is the transformation in the case of the retractions problem). However, the subtype problem does not require the existence of a reverse transformation, while this is necessary for retractions. This, in fact, gives rise to

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

$$\text{(Ax)} \qquad a \trianglelefteq^1 a$$

$$\text{(H)} \qquad \frac{\sigma \trianglelefteq^1 \sigma', \quad \tau \trianglelefteq^1 \tau'}{\sigma \to \tau \trianglelefteq^1 \sigma' \to \tau'}$$

$$\text{(N)} \qquad \frac{\sigma \trianglelefteq^1 \tau}{\sigma \trianglelefteq^1 \Sigma \to \tau}$$

$$\text{(D)} \qquad \frac{\Delta_1 \to a \trianglelefteq^1 \sigma_1, \ \cdots, \ \Delta_n \to a \trianglelefteq^1 \sigma_n}{\Delta_1 \cup \cdots \cup \Delta_n \to a \trianglelefteq^1 \{\Sigma_1 \to \sigma_1 \to a, \cdots, \Sigma_n \to \sigma_n \to a\} \to a}$$

Fig. 5. The syntax directed system (RU)

a crucial difference between these systems as the type retractions can be inferred with the use of the covariant rule (see rule (H) in Figure 5):

$$\text{(Arrow}') \qquad \frac{\vdash \sigma \trianglelefteq \tau, \qquad \vdash \rho \trianglelefteq \gamma}{\vdash \sigma \to \rho \trianglelefteq \tau \to \gamma}.$$

This kind of rule is typical for subtype inference systems with the product constructor $\times$ (see, for example, Bono and Tiuryn (2002)).

Another difference is that the subtype problem requires the existence of a preorder over type atoms. This requirement is essential in the simply typed calculus for obtaining non-trivial inequalities. The preorder is not necessary for the type retractions problem and non-trivial inequalities already arise when there is no preorder over atoms.

3.2. *A system for inference of retraction terms*

We begin this section with an earlier system (RU) to infer the $\trianglelefteq^1$ inequalities. We then propose another system – the term system – which, besides the inequality, infers terms in a combinator algebra. We subsequently give a mapping of the terms to lambda terms so that the results of derivations are mapped to the respective coders and decoders that witness the retractability. This gives a counterpart of the Curry–Howard isomorphism result for retractability inference systems.

We consider a system to infer inequalities $\trianglelefteq^1$. The system was proposed by Regnier and Urzyczyn (RU) and is presented in Figure 5. This system is sound and complete. The soundness means here that any inequality $\sigma \trianglelefteq^1 \tau$ inferred using the system actually holds, while the completeness means that for each inequality $\sigma \trianglelefteq^1 \tau$ we can find a derivation in the system.

Example 6. Let us present a derivation in the system (RU). We continue Example 3 here. A derivation for the inequality $\sigma \trianglelefteq^1 \tau$ looks like

$$\text{(D)}\ \dfrac{\dfrac{\text{(Ax)}\ \dfrac{}{a_1 \trianglelefteq^1 a_1} \quad \text{(H)}\ \dfrac{\text{(Ax)}\ \dfrac{}{a_2 \trianglelefteq^1 a_2} \quad \text{(Ax)}\ \dfrac{}{a \trianglelefteq^1 a}}{a_2 \to a \trianglelefteq^1 a_2 \to a}}{a_1 \to a_2 \to a \trianglelefteq^1 a_1 \to a_2 \to a} \quad \mathsf{D}}{a_1 \to a_2 \to a_3 \to a_4 \to a \trianglelefteq^1 \tau_1 \to a_4 \to a_3 \to a}$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

$$\text{(Ax)} \qquad \vdash I_a \bullet I_a : a \trianglelefteq^1 a \qquad\qquad a \in \mathcal{B}$$

$$\text{(H)} \qquad \frac{\vdash T_1 \bullet T_2 : \sigma \trianglelefteq^1 \sigma', \quad \vdash T_1' \bullet T_2' : \tau \trianglelefteq^1 \tau'}{\vdash H_{\sigma,\tau,\sigma',\tau'} T_1' T_2 \bullet H_{\sigma',\tau',\sigma,\tau} T_2' T_1 : \sigma \to \tau \trianglelefteq^1 \sigma' \to \tau'}$$

$$\text{(N)} \qquad \frac{\vdash T_1 \bullet T_2 : \sigma \trianglelefteq^1 \tau}{\vdash N_{\sigma,\tau,\Sigma} T_1 \bullet N'_{\sigma,\tau,\Sigma} T_2 : \sigma \trianglelefteq^1 \Sigma \to \tau}$$

$$\text{(D)} \qquad \frac{\begin{array}{c}\vdash T_1 \bullet T_1' : \Sigma \to a \trianglelefteq^1 \tau', \\ \vdash T_2 \bullet T_2' : \sigma \trianglelefteq^1 \tau \\ head(\tau') = head(\sigma) = head(\tau) = a\end{array}}{\vdash D T_1 T_2 \bullet D' T_1' T_2' : \Sigma \to \sigma \trianglelefteq^1 (\Delta_1 \to \tau' \to \Delta_2 \to a) \to \tau}$$

$$\text{(P)} \qquad \frac{\vdash T \bullet T' : \sigma_1 \to \cdots \to \sigma_n \to a \trianglelefteq^1 \tau_1 \to \cdots \to \tau_m \to a}{\begin{array}{l}\vdash P^{\pi_1}_{\pi_2} T \bullet P^{\pi_2}_{\pi_1} T' : \sigma_{\pi_1(1)} \to \cdots \to \sigma_{\pi_1(n)} \to a \trianglelefteq^1 \\ \qquad\qquad \tau_{\pi_2(1)} \to \cdots \to \tau_{\pi_2(m)} \to a\end{array}}$$

Fig. 6. The term system for affine retractions

where the derivation D is

$$\text{(H)}\ \frac{\text{(Ax)}\ \dfrac{}{a_3 \trianglelefteq^1 a_3} \qquad \text{(H)}\ \dfrac{\text{(Ax)}\ \dfrac{}{a_4 \trianglelefteq^1 a_4} \qquad \text{(Ax)}\ \dfrac{}{a \trianglelefteq^1 a}}{a_4 \to a \trianglelefteq^1 a_4 \to a}}{a_3 \to a_4 \to a \trianglelefteq^1 a_4 \to a_3 \to a}$$

Note that the lowest use of rule (H) in the derivation D involves an implicit permutation of the arguments a_3 and a_4.

A very important property of the system (RU) is the lack of any cut-like rule. This kind of rule is natural in such a system as $\trianglelefteq^1$ is transitive. In the absence of such a rule, we obtain a system that is a basis for an algorithm for checking the retractability. Moreover, the cut-like transitivity rule is admissible in the system as it is complete.

The system (RU) is not designed to infer terms that witness retractability, so we propose a system to do so. The system (RU) is a good starting point in the design of the term system. We will modify the system (RU) since its original form has several notational abbreviations that are inadequate in the context of generating retracting terms. The term system presented in Figure 6 infers sequents $\vdash C \bullet D : \sigma \trianglelefteq^1 \tau$ such that C, D correspond to terms that witness the retraction $\sigma \trianglelefteq^1 \tau$. The differences between the system (RU) and the term system are that in the former:

(a) terms are not presented;
(b) rule (P) is hidden in the multiset notation;
(c) rule (D) is applicable for many arguments in one shot.

We chose to avoid the implicit permutations and multiset notation, especially in rule (D), as the composition of the resulting terms based on the terms from premises in our system is more straightforward.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

We begin by presenting the signature of the terms.

Definition 4 (Signature of combinators). Here we define algebraic combinators, which will be interpreted as terms of specific types later in the paper. We provide the types here as an aid to understanding.

— I_a (to be interpreted as a term) of the type $a \to a$ for any $a \in \mathscr{B}$.
— $H_{\sigma,\tau,\sigma',\tau'}$ of the type $(\tau \to \tau') \to (\sigma' \to \sigma) \to (\sigma \to \tau) \to \sigma' \to \tau'$ for any simple types σ, τ, σ' and τ'.
— $N_{\sigma,\tau,\Sigma}$, $N'_{\sigma,\tau,\Sigma}$ of the types

$$(\sigma \to \tau) \to \sigma \to \Sigma \to \tau$$
$$(\tau \to \sigma) \to (\Sigma \to \tau) \to \sigma,$$

respectively, where σ along with τ are any simple types and $\Sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ is a sequence of simple types.
— D_{ρ_1,ρ_2}, D'_{ρ_1,ρ_2} of the types

$$((\Sigma \to a) \to \tau') \to (\sigma \to \tau) \to \rho_1 \to \rho_2$$
$$(\tau' \to \Sigma \to a) \to (\tau \to \sigma) \to \rho_2 \to \rho_1,$$

respectively, where

$$\rho_1 = \Sigma \to \sigma$$
$$\rho_2 = (\Delta_1 \to \tau' \to \Delta_2 \to a) \to \tau$$

with $\Sigma, \Delta_1, \Delta_2$ possibly empty sequences of simple types and σ, τ, τ' any simple types.
— $P^{\pi_1,\rho_1}_{\pi_2,\rho_2}$ of the type $(\rho_1 \to \rho_2) \to \rho_3 \to \Sigma \to \rho_2$ where

$$\begin{aligned} \rho_1 &= \sigma_1 \to \cdots \to \sigma_n \to a \\ \rho_2 &= \tau_1 \to \cdots \to \tau_m \to a \\ \rho_3 &= \sigma_{\pi_1(1)} \to \cdots \to \sigma_{\pi_1(n)} \to a \\ \Sigma &= \langle \tau_{\pi_2(1)}, \ldots, \tau_{\pi_2(m)} \rangle \end{aligned}$$

and π_1 is a permutation of $\{1, \ldots, n\}$, and π_2 is a permutation of $\{1, \ldots, m\}$.

We will usually omit the type subscripts in D, D' and $P^{\pi_1}_{\pi_2}$ to avoid the notation becoming too laborious.

Now consider the rules of the system (RU) and the corresponding rules for the term system.

Rule (Ax) handles the case when a base type can be identically embedded into the same type. The pair C, D is simply a pair of identities. Note that the axiom can be applied to type atoms only.

Rule (H) works when the types $\sigma = \sigma_1 \to \sigma_2$ and $\tau = \tau_1 \to \tau_2$ can be decomposed using the $\to$ symbol so that the left subtype σ_1 of σ can be retracted from the left subtype τ_1 of τ and the right subtype σ_2 from the right subtype τ_2. The term that combines the pairs for the subtypes just lifts the arguments to a higher level, then applies a transformation and then retracts back the result.

Rule (N) works when certain arguments of the type τ cannot be taken into consideration when the retract is constructed. In this case, the coder $N_{*,*,*}$ simply does

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

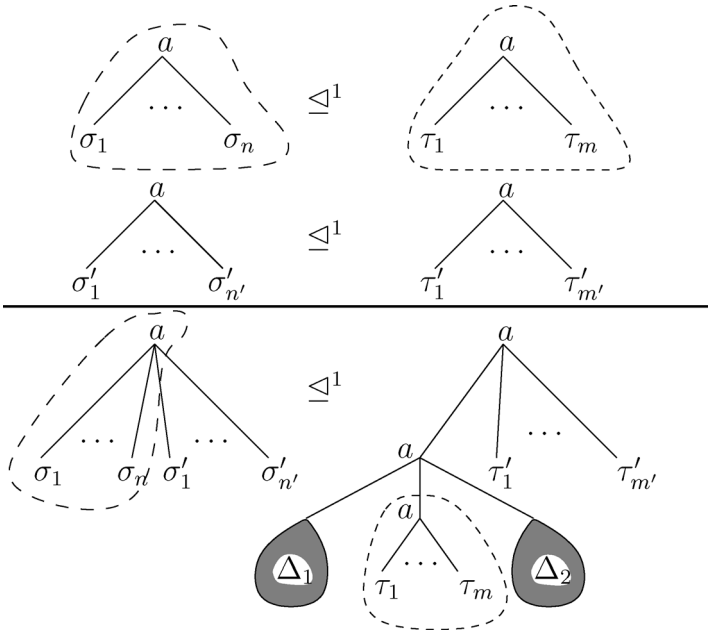


Fig. 7. Rule (D) from the term system with emphasised deep inequality

not use some of the arguments while the decoder $N'_{*,*,*}$ puts dummy free variables in places where the values of the omitted types are required. This is the first of the two rules that are absent from the type isomorphisms systems and make the retractions different.

Rule (D) in both the system (RU) and the term system are a bit difficult to grasp. Figure 7 may help in understanding them. The rule is in a sense similar to rule (H) as it relates the initial part of σ to the initial part of τ. However, the actual work is more complicated as the subtypes of σ are embedded under two levels of arrows. In a sense this operation corresponds to taking the double negation over the subtypes. This is the second of the two rules that are absent from the type isomorphisms systems and make the retractions different.

Rule (P) is, in fact, implicit in the system (RU), as the type arguments there are treated as multisets. We need to make it explicit when we are interested in the extraction of the terms as it does non-trivial work. Its role can be compared to the use of commutativity in the AC-unification. As a consequence of the use of the rule, when we are discussing the term system, the capital Greek letters will denote sequences of types rather than multisets.

It is worth stressing that this system is relatively simple and easy to understand compared with the system implicit in the solution by Padovani (Padovani 2001), where a part of the machinery relies on a version of the higher-order matching problem, which has non-elementary complexity.

We will now present a mapping, $\mathsf{lambdify}(\cdot)$, that translates the algebraic terms into the terms of the simply typed calculus. These terms are in fact extracted from the completeness

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

proof of Regnier and Urzyczyn (Regnier and Urzyczyn 2001). We will rework their proofs later in this paper to obtain the soundness and completeness of our term system.

Definition 5 (Terms from the term system). We define a transformation $\mathsf{lambdify}(\cdot)$ for the retraction combinators used in Figure 6. This transformation translates terms inferred using the rules in Figure 6 to well-typed lambda-terms.

— $\mathsf{lambdify}(I_a) = \lambda x^a.x$, for $a \in \mathscr{B}$

— $\mathsf{lambdify}(H_{\sigma,\tau,\sigma',\tau'}) = \lambda T_1^{\tau\to\tau'}.\lambda T_2^{\sigma'\to\sigma}.\lambda f^{\sigma\to\tau}.\lambda x^{\sigma'}.T_1(f(T_2x))$

— $\mathsf{lambdify}(N_{\sigma,\tau,\Sigma}) = \lambda T^{\sigma\to\tau}.\lambda f^{\sigma}.\lambda(x_i^{\sigma_i})_{i=1}^n.Tf$, where $\Sigma = \{\sigma_1,\dots,\sigma_n\}$

— $\mathsf{lambdify}(N'_{\sigma,\tau,\Sigma}) = \lambda T^{\tau\to\sigma}.\lambda f^{\Sigma\to\tau}.T(f(z_i)_{i=1}^n)$, where $\Sigma = \{\sigma_1,\dots,\sigma_n\}$ and $z_i^{\sigma_i}$ are fresh variables for $i = 1,\dots,n$.

— $\mathsf{lambdify}(D) = \lambda T_1^{(\Sigma\to a)\to\tau'}.\lambda T_2^{\sigma\to\tau}.\lambda f^{\Sigma\to\sigma}.\lambda y^{\Delta_1\to\tau'\to\Delta_2\to a}.$

$$\lambda(z_i^{\tau_i})_{i=1}^p.y(c_i^{\delta_i})_{i=1}^{q_1}\overline{D}(c_i^{\delta_i})_{i=q_1+1}^{q_2}.$$

— $\mathsf{lambdify}(D') = \lambda T_1'^{\tau'\to\Sigma\to a}.\lambda T_2'^{\tau\to\sigma}.\lambda g^{(\Delta_1\to\tau'\to\Delta_2\to a)\to\tau}.\lambda(x_i^{\rho_i})_{i=1}^r.T_2'\overline{D'}$, where

$$\begin{aligned}
\tau &= \tau_1\to\cdots\to\tau_p\to a\\
\Delta_1 &= \langle\delta_1,\dots,\delta_{q_1}\rangle\\
\Delta_2 &= \langle\delta_{q_1+1},\dots,\delta_{q_2}\rangle\\
\Sigma &= \langle\rho_1,\dots,\rho_r\rangle\\
\overline{D} &= T_1\lambda(y_i^{\rho_i})_{i=1}^r.T_2(f(y_i)_{i=1}^r)(z_i)_{i=1}^{z_p}\\
\overline{D'} &= g\lambda(z_i^{\delta_i})_{i=1}^{q_1}.\lambda y^{\tau'}.\lambda(z_i^{\delta_i})_{i=q_1+1}^{q_2}.T_1'y(x_i)_{i=1}^r
\end{aligned}$$

with $c_1,\dots,c_{q_2}$ being fresh variables of the types δ_k for $k = 1,\dots,q_2$.

— $\mathsf{lambdify}(P_{\pi_2}^{\pi_1}) = \lambda T^{\sigma\to\tau}.\lambda x^{\sigma_{\pi_1(1)}\to\cdots\to\sigma_{\pi_1(n)}\to a}.\lambda(y_i^{\tau_i})_{i\in I_1}.$

$$T(\lambda(v_i^{\sigma_i})_{i=1}^n.x(v_i)_{i\in I_2})(y_i)_{i=1}^m,$$

where π_1,π_2 are permutations of the sets $\{1,\dots,n\}$ and $\{1,\dots,m\}$, respectively, and

$$\begin{aligned}
I_1 &= \langle\pi_2(1),\dots\pi_2(m)\rangle\\
I_2 &= \langle\pi_1(1),\dots\pi_1(n)\rangle\\
\sigma &= \sigma_1\to\cdots\to\sigma_n\to a\\
\tau &= \tau_1\to\cdots\to\tau_m\to a.
\end{aligned}$$

Recall that the type subscripts of D, D' and $P_{\pi_2}^{\pi_1}$ are

$$\begin{aligned}
&D, D' \text{ with } \Sigma\to\sigma, \text{ and } (\Delta_1\to\tau'\to\Delta_2\to a)\to\tau\\
&P_{\pi_2}^{\pi_1} \text{ with } \sigma_1\to\cdots\to\sigma_n\to a, \text{ and } \tau_1\to\cdots\to\tau_m\to a.
\end{aligned}$$

We will conclude the presentation of our system with an example.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Example 7. This example shows a derivation in the term system and is a continuation of Examples 3 and 6. A derivation D of the sequent $\vdash T \bullet T' : \sigma \trianglelefteq^1 \tau$ looks like

$$(\mathrm{D})\ \frac{\mathsf{D}_L \qquad \mathsf{D}_R}{\begin{array}{l}\vdash D(H(HI_aI_{a_2})I_{a_1})(P^{\pi_1}_{\pi_2}(H(HI_aI_{a_4})I_{a_3}))\bullet \\ \quad D'(H(HI_{a_2}I_a)I_{a_1})(P^{\pi_2}_{\pi_1}(H(HI_{a_4}I_a)I_{a_3})) \\ \quad : a_1 \to a_2 \to a_3 \to a_4 \to a \trianglelefteq^1 \tau_1 \to a_4 \to a_3 \to a\end{array}}$$

where we have omitted the type annotations in terms to make the presentation easier, $\pi_1(n) = n$ for $n = 1, 2$ and $\pi_2(n) = (n \bmod 2) + 1$, and the derivation D_L is

$$(\mathrm{H})\ \frac{(\mathrm{Ax})\ \dfrac{}{\vdash I_{a_1} \bullet I_{a_1} : a_1 \trianglelefteq^1 a_1} \qquad \mathsf{D}_{LR}}{\vdash H(HI_aI_{a_2})I_{a_1} \bullet H(HI_{a_2}I_a)I_{a_1} : a_1 \to a_2 \to a \trianglelefteq^1 a_1 \to a_2 \to a}$$

with D_{LR} being

$$(\mathrm{H})\ \frac{(\mathrm{Ax})\ \dfrac{}{\vdash I_{a_2} \bullet I_{a_2} : a_2 \trianglelefteq^1 a_2} \quad (\mathrm{Ax})\ \dfrac{}{\vdash I_a \bullet I_a : a \trianglelefteq^1 a}}{\vdash HI_aI_{a_2} \bullet HI_{a_2}I_a : a_2 \to a \trianglelefteq^1 a_2 \to a}$$

and the derivation D_R is

$$(\mathrm{P})\ \frac{(\mathrm{H})\ \dfrac{\mathsf{D}_{RR}}{\vdash a_3 \to a_4 \to a \trianglelefteq^1 a_3 \to a_4 \to a}}{\vdash P^{\pi_1}_{\pi_2}(H(HI_aI_{a_4})I_{a_3}) \bullet P^{\pi_2}_{\pi_1}(H(HI_{a_4}I_a)I_{a_3}) : a_3 \to a_4 \to a \trianglelefteq^1 a_4 \to a_3 \to a}$$

for some derivation D_{RR}, which is very similar to D_L, and the details of which we leave for the interested reader to construct.

We would like to point out that the subterms $HI_aI_{a_2}$ of T and $HI_{a_2}I_a$ of T' result in the same term $\lambda y^{a_2 \to a} x^{a_2}.yx$. This is because we, in fact, use two different terms H here and the distinction disappears as the type annotations are dropped. It is instructive to see the result of $\mathsf{lambdify}(\cdot)$ for the final terms T, T' (the details of the translation and reduction are left for the interested reader):

$$\begin{aligned}\mathsf{lambdify}(T) &= \lambda x^{\sigma} u_1^{\tau_1} u_2^{a_4} u_3^{a_3}.u_1\left(\lambda z_1^{a_1} z_2^{a_2}.f z_1 z_2 u_2 u_3\right)\underline{c^{a_1 \to a_3 \to a}} \\ \mathsf{lambdify}(T') &= \lambda y^{\tau} v_1^{a_1} v_2^{a_2} v_3^{a_3} v_4^{a_4}.y\left(\lambda z_1^{a_1 \to a_2 \to a} z_2^{a_1 \to a_3 \to a}.z_1 v_1 v_2\right) v_4 v_3\end{aligned}$$

(recall that $\tau_1 = (a_1 \to a_2 \to a) \to (a_1 \to a_3 \to a) \to a$). Note that the small difference compared with the terms presented in Example 3 have been underlined.

3.3. *The soundness of the system*

We can now prove that the system we have presented infers exactly the coders and decoders for the affine retractions problem. We start with the proof that the terms displayed in Figure 6 do indeed form the retraction pairs.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Proposition 3.2 (Soundness). If $\vdash T \bullet T' : \sigma \trianglelefteq^1 \tau$ is derivable, then $\mathsf{lambdify}(T)$ is affine and has type $\sigma \to \tau$, $\mathsf{lambdify}(T')$ is affine and has type $\tau \to \sigma$, and $\mathsf{lambdify}(T) \circ \mathsf{lambdify}(T') =_{\beta\eta} \lambda x^\sigma.x^\sigma$.

Proof. The proof is by induction on the derivation of $\vdash T \bullet T' : \sigma \trianglelefteq^1 \tau$. In each case this just amounts to a routine check of the reduction. □

3.4. *The completeness of the system*

The system we introduce here is complete in the sense that when the inequality $\trianglelefteq^1$ holds one can infer it and obtain the terms that witness the retraction. This system, however, does not derive all the possible retractions. In particular, the translation of the combinator N' substitutes fresh variables in places where any inhabitant of the appropriate type could be inserted (see Examples 3 and 7). In fact, this is not the only reason why some retraction pairs cannot be inferred, but a proper characterisation of such terms is highly non-trivial, and is beyond the scope of this paper.

In order to prove completeness, we need a few definitions and technical lemmas proved in Regnier and Urzyczyn (2001). As usual in papers dealing with $\beta\eta$-reduction, it is handy to deal with the long normal forms.

Definition 6 (Long forms). We say that a term M is in *long form* if and only if for any context $\mathrm{C}[]$ and term $N^{\tau\to\tau'}$, the equality $\mathrm{C}[N^{\tau\to\tau'}] \equiv M$ ($\equiv$ means here syntactical equality) implies that $\mathrm{C}[] \equiv C'[[]P^\tau]$ or that $N^{\tau\to\tau'} \equiv \lambda x^\tau.P^{\tau'}$. A term is in *long normal form* when it is in β-normal form and is in long form.

Each simply typed term has its unique long normal form. For example, the long form of $(\lambda x^{(a\to a)\to(a\to a)\to a}.x)\lambda x^{a\to a}y^{a\to a}.x$ is

$$(\lambda x^{(a\to a)\to(a\to a)\to a} z_1^{a\to a} z_2^{a\to a}.x(\lambda u_1^a.z_1u_1)(\lambda u_2^a.z_2u_2))\lambda x^{a\to a}y^{a\to a}z_1^a.xz_1\,,$$

and the long normal form is $\lambda x^{a\to a}y^{a\to a}z_1^a.xz_1$.

The following observations concerning the form of the decoder and the form of retractable types will be used extensively in the proof of completeness.

Lemma 3.3 (Shapes).

(1) If $T'^{\tau\to\sigma}(T^{\sigma\to\tau}x^\sigma) =_{\beta\eta} x^\sigma$ and $x^\sigma \notin \mathrm{FV}(T, T')$, then the long normal form of T' has the shape $\lambda f^\tau.\lambda\vec{z}.f^\tau\vec{M}$.
(2) If $\sigma \trianglelefteq^1 \tau$, then $head(\sigma) = head(\tau)$.

Part (1) of the lemma gives us the general shape of the decoder and Part (2) allows us to check immediately that $\sigma \to a \not\trianglelefteq^1 \tau \to b$.

The following lemma helps in dealing with the reduction.

Lemma 3.4 (Head variable extensions). Let

$$M = M^0\big[X := x(V_i)_{i=1}^k\big] \to^*_{\beta\eta} x(A_i)_{i=1}^k(B_i)_{i=1}^r$$

where M is an affine term and $x \notin \mathrm{FV}(M^0)$. Then $M^0 \to^*_{\beta\eta} X(B_i)_{i=1}^r$.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

This lemma can now be used to obtain the following proposition.

Proposition 3.5 (Head variable extensions II). Let

$$M = M^0\left[X := \lambda(z_i)_{i=1}^r.x(z_i)_{i=1}^r(V_i)_{i=1}^k\right] \to_{\beta\eta}^* x(B_i)_{i=1}^r(A_i)_{i=1}^k$$

where M is an affine term and $x \notin \mathrm{FV}(M^0)$. Then $M^0 \to_{\beta\eta}^* X(B_i)_{i=1}^r$.

We can now give the completeness theorem.

Theorem 3.6 (Completeness). If $\sigma \trianglelefteq^1 \tau$, there is a pair of terms $\hat{T}^{\sigma\to\tau}, \hat{T}'^{\tau\to\sigma}$ over the signature from Definition 4 such that $\vdash \hat{T} \bullet \hat{T}' : \sigma \trianglelefteq^1 \tau$ is derivable in the term system and the pair $\mathsf{lambdify}(\hat{T}), \mathsf{lambdify}(\hat{T}')$ is a witness that σ is a retract of τ.

Proof. The proof is by induction with respect to the size of $\sigma \trianglelefteq^1 \tau$. Let $C^{\sigma\to\tau}, D^{\tau\to\sigma}$ be a pair of terms that is a witness for $\sigma \trianglelefteq^1 \tau$.

Size of $\sigma \trianglelefteq^1 \tau$ is 2.
In this case, $\sigma = \tau = a$ and the term I_a can serve for both C and D as the pair of two identical terms, and $\mathsf{lambdify}(I_a) = \lambda x^a.x$ is indeed a witness of the retractability here.

Size of $\sigma \trianglelefteq^1 \tau$ is greater than 2.
Let $\sigma = \sigma_1 \to \cdots \to \sigma_l \to a$ and $\tau = \tau_1 \to \cdots \to \tau_n \to a$. We can assume that $C^{\sigma\to\tau}, D^{\tau\to\sigma}$ are such that $x \notin \mathrm{FV}(C, D)$ and $D(Cx^\sigma) =_{\beta\eta} x$ (otherwise we just rename all free occurrences of x in C and D to a fresh variable). Our goal is to construct a derivation in the term system and two terms $\hat{T}, \hat{T}'$ over the signature from Definition 4 such that $\mathsf{lambdify}(\hat{T}), \mathsf{lambdify}(\hat{T}')$ is a witness of $\sigma \trianglelefteq^1 \tau$. Without loss of generality, we may assume that C, D are in long normal form. As C, D witness the retractability of σ and τ, we have $D(Cx) =_{\beta\eta} x$. Thus Lemma 3.3(1) gives us the shape of D, and we have

$$\begin{aligned} C &= \lambda f^\sigma \lambda(y_i^{\tau_i})_{i=1}^n.\overline{C}, \\ D &= \lambda g^\tau \lambda(z_i^{\sigma_i})_{i=1}^l.\overline{D} = \lambda g^\tau \lambda(z_i^{\sigma_i})_{i=1}^l.g(S_i^{\tau_i})_{i=1}^n. \end{aligned} \tag{2}$$

The variable g has exactly one occurrence in $\overline{D}$ as the term is affine and the reduction $\lambda(z_i^{\sigma_i})_{i=1}^l.\overline{D}[g := Cx]$ must result in x, which is absent from $\overline{D}$.

As $\overline{D}$ is affine, we can define a partial function $t : \{1,\ldots,l\} \rightharpoonup \{1,\ldots,n\}$ such that $t(i) = j$ when $z_i^{\sigma_i}$ occurs in S_j.

If the partial function t is empty, σ has no arguments. Indeed, if σ had an argument, then $D(Cx) =_{\beta\eta} \lambda(z_i^{\sigma_i})_{i=1}^k.xC_1\ldots C_k$ for $k > 0$, but none of z_i for $i = 1,\ldots,k$ could occur in $C_1,\ldots,C_k$ so it would not be possible to obtain x as the $\beta\eta$ normal form. As σ has no arguments, we can construct the derivation using rule (N) with the axiom in the premise and define $\hat{T} = N_{\sigma,\tau,\Sigma}I_a$ and $\hat{T}' = N'_{\sigma,\tau,\Sigma}I_a$ where Σ is the sequence of all the arguments of τ.

We can now assume that t is a non-empty partial function. We can rearrange the arguments in τ so that we have $t(1) = 1$. Let τ' be the result of the operation. Suppose we obtained $\vdash U \bullet U' : \sigma \trianglelefteq^1 \tau'$. We can now use rule (P) with appropriate permutations π_1, π_2 to obtain a derivation for $\vdash P_{\pi_2}^{\pi_1}U \bullet P_{\pi_1}^{\pi_2}U' : \sigma \trianglelefteq^1 \tau$ and the required $\hat{T} = P_{\pi_2}^{\pi_1}U$ and $\hat{T}' = P_{\pi_1}^{\pi_2}U'$.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

With the above reasoning in mind, we may assume without loss of generality that for the original τ we have $t(1) = 1$. Let $C^{\#} = \lambda f^{\sigma} \lambda (y_i^{\tau_i})_{i=1}^n . \overline{C}[y_1 := \#]$ where $\#$ is a fresh variable. We say that the index 1 is *active* if the head symbol of $\mathrm{NF}(D(C^{\#}x))$ is $\#$. Otherwise the index is *passive*. We construct the derivation and terms by cases depending on the activity of the index 1.

Case when the index 1 *is passive.*
In this case we prove below that

(i) $\sigma_1 \trianglelefteq^1 \tau_1$
(ii) $\sigma' \trianglelefteq^1 \tau'$

where $\sigma' = \sigma_2 \to \cdots \to \sigma_l \to a$ and $\tau' = \tau_2 \to \cdots \to \tau_n \to a$. By the induction hypothesis, we then get that $\vdash \hat{T}_1 \bullet \hat{T}'_1 : \sigma_1 \trianglelefteq^1 \tau_1$ for some $\hat{T}_1, \hat{T}'_1$ and $\vdash \hat{T}_2 \bullet \hat{T}'_2 : \sigma' \trianglelefteq^1 \tau'$ for some $\hat{T}_2, \hat{T}'_2$ are derivable. This enables us to use rule (H) and the terms $\hat{T} = H_{\sigma_1,\sigma',\tau_1,\tau'} \hat{T}_2 \hat{T}'_1$ with $\hat{T}' = H_{\tau_1,\tau',\sigma_1,\sigma'} \hat{T}'_2 \hat{T}_1$ to obtain the required derivation and terms. Finally, we conclude that $\mathsf{lambdify}(\hat{T})$ and $\mathsf{lambdify}(\hat{T}')$ witness the inequality $\sigma \trianglelefteq^1 \tau$ by Proposition 3.2.

We now give the proofs of (i) and (ii) above.

(i) *Proof of* $\sigma_1 \trianglelefteq^1 \tau_1$.
Consider the leftmost-outermost reductions of $D(C^{\#}x)$ and $D(Cx)$:

$$D(C^{\#}x) \to_{\beta\eta}^* U \;=\; \lambda (z_i)_{i=1}^l . (\lambda (y_i^{\tau_i})_{i=1}^n . \overline{T}[y_1 := \#])(S_i^{\tau_i})_{i=1}^n$$
$$D(Cx) \to_{\beta\eta}^* V \;=\; \lambda (z_i)_{i=1}^l . (\lambda (y_i^{\tau_i})_{i=1}^n . \overline{T})(S_i^{\tau_i})_{i=1}^n$$

where U, V are the first terms to have this shape. We have the following fact connecting U and V above:

For each U' *such that* $U \to_{\beta\eta}^* U' = \lambda (z_i)_{i=1}^l . U''$ *where the reduction uses the leftmost-outermost redexes only, there exists a* V' *such that* $V \to_{\beta\eta}^* V' = \lambda (z_i)_{i=1}^l . V''$ *and* $V'' = U''[\# := S^{\tau_1}]$. (∗)

This fact is easy to show by induction on the length of the reduction

$$D(C^{\#}x) \to_{\beta\eta}^* U' .$$

As $\#$ cannot occur as the head symbol in $\mathrm{NF}(D(C^{\#}x))$, we can now use use (∗) to perform reductions

$$D(C^{\#}x) \to_{\beta\eta}^* \lambda (z_i)_{i=1}^l . \overline{T}[y_1 := \#][y_r := S_r^{\tau_r}]_{r=2}^n \to_{\beta\eta}^* \lambda (z_i)_{i=1}^l . x (N_i^{\sigma_i})_{i=1}^l$$
$$D(Cx) \to_{\beta\eta}^* \lambda (z_i)_{i=1}^l . \overline{T}[y_r := S_r^{\tau_r}]_{r=1}^n \to_{\beta\eta}^* \lambda (z_i)_{i=1}^l . x (M_i^{\sigma_i})_{i=1}^l$$

such that $x (M_i^{\sigma_i})_{i=1}^l = x (N_i^{\sigma_i})_{i=1}^l [\# := S^{\tau_1}]$. As $t(1) = 1$, we have $\# \in \mathrm{FV}(N_1^{\sigma_1})$ and then $M_1^{\sigma_1} = N_1^{\sigma_1}[\# := S^{\tau_1}]$.
Furthermore, the final reduction of $D(Cx) \to_{\beta\eta}^* x$ is possible only when $M_1^{\sigma_1} \to_{\beta\eta}^* z_1^{\sigma_1}$, so

$$(\lambda y_1^{\tau_1} . N_1^{\sigma_1}[\# := y_1])((\lambda z_1^{\sigma_1} . S_1^{\tau_1}) z_1^{\sigma_1}) \to_{\beta\eta}^* z_1^{\sigma_1} .$$

This implies that $\sigma_1 \trianglelefteq^1 \tau_1$ with a coder $\lambda z_1^{\sigma_1} . S_1^{\tau_1}$ and a decoder

$$\lambda z_1^{\tau_1} . N_1^{\sigma_1}[\# := z_1] .$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

(ii) *Proof of* $\sigma' \trianglelefteq^1 \tau'$.

Using the notation in (2), we define terms

$$\begin{aligned} C' &= \lambda f'^{\sigma'} \lambda (y_i^{\tau_i})_{i=2}^n . \overline{C}[f := \lambda (z_i^{\sigma_i})_{i=1}^l . f'(z_i^{\sigma_i})_{i=2}^l][y_1 := S_1^{\tau_1}] \\ D' &= \lambda g^{\tau'} \lambda (z_i^{\sigma_i})_{i=2}^l . \overline{D'} = \lambda g^{\tau'} \lambda (z_i^{\sigma_i})_{i=2}^l . g(S_i^{\tau_i})_{i=2}^n . \end{aligned}$$

Consider the reduction

$$\begin{aligned} & D'(C' x'^{\sigma'}) \\ & \quad \to_{\beta\eta}^* \lambda (z_i^{\sigma_i})_{i=2}^l . (\lambda (y_i^{\tau_i})_{i=2}^n . \overline{C}[f := \lambda (z_i^{\sigma_i})_{i=1}^l . x'(z_i^{\sigma_i})_{i=2}^l][y_1 := S_1^{\tau_1}])(S_i^{\tau_i})_{i=2}^n \\ & \quad \to_{\beta\eta}^* \lambda (z_i^{\sigma_i})_{i=2}^l . \overline{C}[y_i := S_i^{\tau_i}]_{i=1}^n [f := \lambda (z_i^{\sigma_i})_{i=1}^l . x'(z_i^{\sigma_i})_{i=2}^l] \\ & \quad = \lambda (z_i^{\sigma_i})_{i=2}^l . \overline{C}[y_i := S_i^{\tau_i}]_{i=1}^n [f := x^{\sigma}][x := \lambda (z_i^{\sigma_i})_{i=1}^l . x'(z_i^{\sigma_i})_{i=2}^l] \\ & \quad = A[x := \lambda (z_i^{\sigma_i})_{i=1}^l . x'(z_i^{\sigma_i})_{i=2}^l] \end{aligned}$$

where $A =_{\beta\eta} D(Cx^{\sigma})$. As $A \to_{\beta\eta}^* x$, we immediately obtain $D'(C'x') \to_{\beta\eta}^* x'$.

Case when the index 1 *is active.*

We can assume here that $t^{-1}(1) = \{1, \ldots, p\}$, since if this is not the case, we can rearrange the arguments of σ by permuting the appropriate variables under λ in D and arguments of f in C. Then the conclusion obtained for the case with rearranged types can be adapted by using rule (P) to permute the arguments of σ back to the original sequence.

Let $\sigma = \sigma_1 \to \cdots \to \sigma_p \to \sigma'$ and $\tau = (\delta_1 \to \cdots \to \delta_q \to a) \to \tau'$. Under the assumption that $t^{-1}(1) = \{1, \ldots, p\}$, we show below that $\sigma_1 \to \cdots \to \sigma_p \to a \trianglelefteq^1 \delta_r$ for some $r \in \{1, \ldots, q\}$ and $\sigma' \trianglelefteq^1 \tau'$. By the induction hypothesis, we then obtain terms $\hat{T}_1, \hat{T}'_1$ such that $\vdash \hat{T}_1 \bullet \hat{T}'_1 : \sigma_1 \to \cdots \to \sigma_p \to a \trianglelefteq^1 \delta_r$ holds and terms $\hat{T}_2, \hat{T}'_2$ such that $\vdash \hat{T}_2 \bullet \hat{T}'_2 : \sigma' \trianglelefteq^1 \tau'$ holds. This enables us to use rule (D) and the terms $\hat{T} = D_{\sigma,\tau} \hat{T}_1 \hat{T}_2$ and $\hat{T}' = D'_{\sigma,\tau} \hat{T}'_1 \hat{T}'_2$ to obtain the required derivation and terms. Finally, we conclude that $\mathsf{lambdify}(\hat{T})$ and $\mathsf{lambdify}(\hat{T}')$ witness the inequality $\sigma \trianglelefteq^1 \tau$ by Proposition 3.2.

Proof of the derivability of $\sigma_1 \to \cdots \to \sigma_p \to a \trianglelefteq^1 \delta_r$ *for some* $r \in \{1, \ldots, q\}$.

Recall that $\tau_1 = \delta_1 \to \cdots \to \delta_q \to a$ and $t^{-1}(1) = \{1, \ldots, p\}$. Consider the equalities

$$\begin{aligned} D(Cx) &\to_{\beta\eta}^* \lambda (z_i^{\sigma_i})_{i=1}^l . (\lambda (y_i^{\tau_i})_{i=1}^n . \overline{C})(S_i^{\tau_i})_{i=1}^n \\ &= \lambda (z_i)_{i=1}^l . (\lambda (y_i^{\tau_i})_{i=1}^n . \overline{C}) \#(S_i^{\tau_i})_{i=2}^n [\# := S_1^{\tau_1}] \\ &=_{\beta\eta} \lambda (\hat{z}_i)_{i=1}^l . (\lambda (z_i)_{i=1}^l . (\lambda (y_i^{\tau_i})_{i=1}^n . \overline{C}) \#(S_i^{\tau_i})_{i=2}^n [\# := S_1^{\tau_1}])(\hat{z}_i)_{i=1}^l \\ &=_{\beta\eta} \lambda (\hat{z}_i)_{i=1}^l . (\lambda (z_i)_{i=1}^l . (\lambda (y_i^{\tau_i})_{i=1}^n . \overline{C}) \#(S_i^{\tau_i})_{i=2}^n) \\ &\qquad\qquad [\# := (\lambda (z_i)_{i=1}^l . S_1^{\tau_1})(\hat{z}_i)_{i=1}^l](\hat{z}_i)_{i=1}^l \\ &=_{\beta\eta} \lambda (\hat{z}_i)_{i=1}^l . D(C^{\#} x)[\# := (\lambda (z_i)_{i=1}^l . S_1^{\tau_1})(\hat{z}_i)_{i=1}^l](\hat{z}_i)_{i=1}^l \\ &= M . \end{aligned}$$

As the index 1 is active, we have

$$M \to_{\beta\eta}^* \lambda (\hat{z}_i)_{i=1}^l . (\lambda (z_i)_{i=1}^l . \#(M_i^{\delta_i})_{i=1}^q)[\# := (\lambda (z_i)_{i=1}^l . S_1^{\tau_1})(\hat{z}_i)_{i=1}^l](\hat{z}_i)_{i=1}^l . \tag{3}$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

There is only one occurrence of x in the term $\#(M_i^{\delta_i})_{i=1}^q$. Thus, this occurrence must be in one of terms M_i for $i = 1, \ldots, q$. Let x occur within $M_r^{\delta_r}$ and let $M_r^{\delta_r} = \mathrm{C}[x(W_i^{\sigma_i})_{i=1}^l]$. We claim that the inequality

$$\sigma_1 \to \cdots \to \sigma_p \to a \trianglelefteq^1 \delta_r \tag{4}$$

holds. Note that the variables $z_1, \ldots, z_p$ cannot occur in $M_r^{\delta_r}$ as they already occur in $S_1^{\tau_1}$ and the terms are affine. In order to show (4), we define

$$\hat{\sigma} = \sigma_1 \to \cdots \to \sigma_p \to a \quad \text{and} \quad \hat{C} = \lambda X^{\hat{\sigma}}.\mathrm{C}[X(W_i^{\sigma_i})_{i=1}^p]$$

where X is a fresh variable. Furthermore, we define

$$\hat{D} = \lambda g^{\delta_r} \lambda(z_i^{\sigma_i})_{i=1}^p.S_1^{\tau_1}(M_i^{\delta_i})_{i=1}^{r-1} g^{\delta_r}(M_i)_{i=r+1}^q\,.$$

We have

$$M' = \hat{D}(\hat{C}x') \to_{\beta\eta}^* \lambda(z_i^{\sigma_i})_{i=1}^p.S_1^{\tau_1}(M_i)_{i=1}^{r-1}(\mathrm{C}[X(W_i^{\sigma_i})_{i=1}^p][X := x'])(M_i)_{i=r+1}^q\,.$$

The final term reduces to $\lambda(z_i)_{i=k}^l.x'(z_i)_{i=1}^p$. Indeed, we have $M =_{\beta\eta} \lambda(z_i)_{i=1}^l.x(z_i)_{i=1}^l$ and M' with X replaced by $\lambda(\hat{z}_i^{\sigma_i})_{i=1}^p.x(\hat{z}_i^{\sigma_i})_{i=1}^p(W_i^{\sigma_i})_{i=p+1}^l$ is $\beta\eta$ equivalent to the final term in (3). We can now apply Lemma 3.5 and obtain the required conclusion. In this way we get that the inequality (4) holds with $\hat{C}, \hat{D}$ witnessing the inequality.

The proof that $\sigma' \trianglelefteq^1 \tau'$ is omitted since it is very similar to the proof given in part (ii) for the *Case when index 1 is passive*. Further details of the proof are left for the interested reader. □

Example 8. Now that we have the completeness theorem, we can apply it to show that in some cases there are no affine retractions. Consider types σ and τ from Example 4. We show here that there is no derivation in the term system that relates these two types. This makes the relation $\sigma \trianglelefteq^1 \tau$ impossible as it would contradict the conclusion of Theorem 3.6.

Now, we analyse the possible applications of the rules from the term system:

(Ax) This rule is impossible as σ and τ are not type atoms.

(H) The application of this rule results in two premises that relate pairs of types: $a \trianglelefteq^1 (a \to a \to a) \to (a \to a \to a) \to a$ and $a \to a \to a \trianglelefteq^1 a$. However, the only rule that can be applied to the second pair is (P), which reduces the problem of finding a derivation to the one for the same pair $a \to a \to a \trianglelefteq^1 a$. Thus there is no derivation for $\sigma \trianglelefteq^1 \tau$ starting with (H).

(N) The only possible application of the rule reduces the problem to the one of finding a derivation for $a \to a \to a \to a \trianglelefteq^1 a$. However, the only rule that can be applied to the pair is (P), which, in a similar way to the case for rule (H), does not reduce the problem.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

(D) This rule can be applied in two ways:

— The first allows us to reduce the problem to that of finding derivations for $a \to a \to a \trianglelefteq^1 a \to a \to a$ and $a \to a \trianglelefteq^1 a$. However, in this case the only possible rule that can be applied to the second pair is (P), which does not reduce the problem.

— The second allows us to reduce the problem to that of finding derivations for $a \to a \to a \to a \trianglelefteq^1 a \to a \to a$ and $a \trianglelefteq^1 a$. In this case, we can apply (H) several times to the first pair, but this eventually leads to a problem in finding a derivation for $a \to a \trianglelefteq^1 a$, which is exactly the same as in the previous case.

(P) The application of rule (P) just transforms the problem of finding a derivation to the same problem, so we cannot apply the rule.

As a result of this analysis we get that a derivation for $\sigma \trianglelefteq^1 \tau$ cannot start with any of the rules in the term system, which implies, as required, that there is no derivation in this system.

3.5. *Permutations of a derivation*

We have already established the main properties of the system in Figure 6. We can now try to present a certain normal form of derivations in the system. To this end, we provide some rules that allow us to make permutations inside derivations. These permutations will be exploited later in the NP-hardness proof and in the section where the polynomial algorithms are studied. The permutation possibilities are summarised in Propositions 3.7–3.13.

In order to make the formulation of the propositions easier, we introduce a few conventions for referring to nodes in a derivation. For each rule in Figure 6 the sequents above the rule line are called *premises* of the rule and the sequent below the line is called the *conclusion*. In both rules (H) and (D) in Figure 6, the first sequent above the rule line is called the *left premise* of the rule and the second is called the *right premise*. For simplicity, we will sometimes refer to the single premise sequent of rules (N) and (P) as either the *left premise* or the *right premise*. We treat derivations as trees, the nodes of which are labelled with sequents and rule names, for example, (H), (N). We say that a derivation ends with a particular rule when the root of the derivation tree is labelled with the rule name. We also use additional superscripts in the rule names, for example, $(\mathrm{H})^1$, $(\mathrm{H})^2$, to disambiguate the references to different uses of the same rule.

The proofs of the following propositions are by routine case analyses and are left for the interested reader. Note that we have simplified the notation here and have left fragments of the sequents implicit.

Proposition 3.7 (Collapsing of (P) rules). Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$. If D has the form

$$\dfrac{\dfrac{\mathsf{D}'}{\vdash \cdots}\,(\mathrm{P})}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}\,(\mathrm{P})$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

where D' is a derivation, then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ of the form

$$(\mathrm{P})\ \frac{\mathsf{D}'}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}$$

Proposition 3.8 (Collapsing of (N) rules). Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$. If D has the form

$$(\mathrm{N})\ \frac{(\mathrm{N})\ \dfrac{\mathsf{D}'}{\vdash \cdots}}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}$$

where D' is a derivation, then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ of the form

$$(\mathrm{N})\ \frac{\mathsf{D}'}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}$$

Proposition 3.9 (Permutation of (P) with (N)). Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$. If D has the form

$$(\mathrm{P})\ \frac{(\mathrm{N})\ \dfrac{\mathsf{D}'}{\vdash \cdots}}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}$$

where D' is a derivation, then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ of the form

$$(\mathrm{N})\ \frac{(\mathrm{P})\ \dfrac{\mathsf{D}'}{\vdash \cdots}}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}$$

Proposition 3.10 (Contraction of (H) or (D) with (N)). Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$. If D has the form

$$(\mathrm{N})\ \frac{(\mathrm{X})\ \dfrac{\mathsf{D}_L \qquad \mathsf{D}_R}{\vdash \cdots}}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}$$

where (X) is either rule (H) or rule (D) and $\mathsf{D}_L, \mathsf{D}_R$ are derivations, then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ of the form

$$(\mathrm{N})\ \frac{\mathsf{D}_R}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}$$

or of the form D_R.

Proposition 3.11 (Permutation of (H) or (D) with (P)). Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$. If D has the form

$$(\mathrm{X})\ \frac{\mathsf{D}_L \qquad (\mathrm{P})\ \dfrac{\mathsf{D}_R}{\vdash \cdots}}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

where (X) is either rule (H) or rule (D) and $\mathsf{D}_L, \mathsf{D}_R$ are derivations, then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ of the form

$$\text{(P)}\ \dfrac{\text{(X)}\ \dfrac{\mathsf{D}_L \qquad \mathsf{D}_R}{\vdash \cdots}}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}$$

Proposition 3.12 (Permutation of (H) or (D) with (H) or (D)). Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$. If D has the form

$$\text{(X)}^1\ \dfrac{\mathsf{D}_L \qquad \text{(X)}^2\ \dfrac{\mathsf{D}_{RL} \qquad \mathsf{D}_{RR}}{\vdash \cdots}}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}$$

where $(X)^1$ and $(X)^2$ are either (H) or (D) (but not necessarily the same), and $\mathsf{D}_L, \mathsf{D}_{RL}$ and D_{RR} are derivations, then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ of the form

$$\text{(P)}\ \dfrac{\text{(X)}^2\ \dfrac{\mathsf{D}_{RL} \qquad \text{(X)}^1\ \dfrac{\mathsf{D}_L \qquad \mathsf{D}_{RR}}{\vdash \cdots}}{\vdash \cdots}}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}$$

Note that rule (D) may be replaced equivalently by rule (N) when the former has a sequent $a \trianglelefteq^1 a$ with $a \in \mathscr{B}$ as the left premise.

As a corollary of this series of propositions, we obtain a certain normal form of derivations.

Definition 7 (Well-formed derivations). We say that a derivation D is *almost well formed* when it

— ends with (Ax);
— ends with (N) and its premise is almost well formed;
— ends with (H) and its right premise is almost well formed;
— ends with (D) that cannot be replaced equivalently by an (N) rule, and its right premise is almost well formed.

We say that a derivation D is *well formed* when it ends with rule (P) preceded by an almost well-formed derivation.

Note that the derivation D from Example 7 is not well formed as it involves the use of rule (P) in D_R, but the derivation D_R is well formed as it involves the use of (P) at the end only.

We need one more kind of permutation of derivations. The formulation requires the notion of an almost well-formed derivation.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Proposition 3.13 (Permutation of (N) with (P)). Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$. If D has the form

$$\dfrac{\dfrac{\mathsf{D}'}{\vdash \cdots}\,(\mathrm{P})}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}\,(\mathrm{N})$$

where D' is an almost well-formed derivation, then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ of the form

$$\dfrac{\mathsf{D}'''}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau.}\,(\mathrm{P})$$

for some almost well-formed derivation D''', the size of which is not greater than the size of D' plus k where k is the number of arguments discarded by all the (N) rules in the derivation D.

Proof. The proof is by induction on the lexicographically ordered pairs $\langle n, m\rangle$ where n is the size of the derivation D' and m is the number of arguments discarded by the rules (N) in D'.

Case $\langle 1, 1\rangle$:

This case is impossible as the size of the derivation is 1, which means that D' is an axiom.

Case $\langle n+1, 1\rangle$:

Let the premise of (N) include $\sigma \trianglelefteq^1 \tau_1 \to \tau$. We have two cases here:

— The permutation π_2 of rule (P) (in the notation of Figure 6) is such that $\pi_2(1) = 1$:
In this case we can immediately transform the derivation to

$$\dfrac{\dfrac{\mathsf{D}'}{\vdash \cdots}\,(\mathrm{N})}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}\,(\mathrm{P})$$

which completes the proof in this case.

— $\vec{\pi}_2(1) \neq 1$:
In this case we do the case analysis with respect to the first rule (X) of D'.
The situation when D' ends with (Ax) is impossible since then $\sigma = \tau = a$ for some atom a and the final rule (N) of D could not be applied.
When D' ends with (X) that is either (H) or (D), the derivation D has the form

$$\dfrac{\dfrac{\dfrac{\mathsf{D}_L \qquad \mathsf{D}_R}{\vdash \cdots}\,(\mathrm{X})}{\vdash \cdots}\,(\mathrm{P})}{\vdash C \bullet D : \sigma \trianglelefteq^1 \tau}\,(\mathrm{N})$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

A simple case analysis shows that this derivation can be replaced by

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{D}_L}{\vdash \cdots}\,(\mathrm{X}) \qquad \dfrac{\dfrac{\mathsf{D}_R}{\vdash \cdots}\,(\mathrm{P})}{\vdash \cdots}\,(\mathrm{N})}{\vdash \cdots}\,(\mathrm{P})}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau} \tag{5}$$

where the (N) rule discards one argument as the final (N) in D. Here we can apply the induction hypothesis to the right premise of the (X) rule in (5) and obtain a derivation D'_R, the size of which is no greater than the size of D_R with (N) plus k, where $k = k_1 + 1$ with 1 standing for the one argument discarded by the final (N) rule and k_1 being the number of arguments discarded by all (N) rules in D_R. In this way we obtain a derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{D}_L}{\vdash \cdots}\,(\mathrm{X}) \qquad \dfrac{\mathsf{D}'_R}{\vdash \cdots}\,(\mathrm{P})}{\vdash \cdots}}{\vdash C'' \bullet D'' : \sigma \trianglelefteq^1 \tau}\,(\mathrm{P})$$

Next we can use Proposition 3.12 to permute the upper (P) rule below (X) and then Proposition 3.7 to collapse the (P) rules. The size of the resulting derivation is less than the size of D plus k, which fulfils the requirements of the current proposition. When D' ends with (N), we can use Proposition 3.9 to move the upper (N) rule below (P) and then collapse the (N) rules using Proposition 3.8. The resulting derivation is smaller, so we can apply the induction hypothesis and obtain the final result.

Case $\langle n+1, m+1\rangle$:

In this case we can split the final (N) rule into two rules – the upper one, which discards only one argument, and the lower one, which discards the remaining m arguments. We can then apply the induction hypothesis for $\langle n+1, 1\rangle$. The size of the resulting derivation D^1 above the rule (P) is no greater than the size of D' plus the number $m_0 + 1$, where m_0 is the number of the types discarded by the (N) rules in D'. We can now apply the induction hypothesis to the remaining derivation

$$\dfrac{\dfrac{\mathsf{D}^1}{\vdash \cdots}\,(\mathrm{P})}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}\,(\mathrm{N})$$

where the number of arguments discarded by (N) is m. This will result in a derivation

$$\dfrac{\mathsf{D}^2}{\vdash C' \bullet D' : \sigma \trianglelefteq^1 \tau}\,(\mathrm{P})$$

where the size of D^2 is no greater than $m + m_0 + 1$, which completes the proof. □

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Corollary 3.14 (Normalisation to well-formed derivations). For each derivation D for a retraction $\sigma \trianglelefteq^1 \tau$, there is a derivation D' of $\sigma \trianglelefteq^1 \tau$ that is well formed.

Proof. We use induction over the derivation D, taking cases for the final rule.

(Ax):

In this case the derivation is almost well formed. We can add here a rule (P) with the identical permutation of arguments. In this way we obtain the required well-formed derivation.

(D) or (H):

By the induction hypothesis, we may assume that the right premise of the rule is in well-formed form and ends with rule (P) preceded by an almost well-formed derivation D'. Then we can use Proposition 3.11 to make rule (P) the final rule of the derivation, which is preceded by rule (D) or (H), respectively, for which a derivation D' is the derivation of the right premise. The derivation obtained in this way is well formed.

(N):

By the induction hypothesis, we may assume that the premise of the rule is in well-formed form and ends with rule (P) preceded by an almost well-formed derivation D'. We can then use Proposition 3.13 to make rule (P) the final rule of the derivation, which is preceded by an almost well-formed derivation D'. The resulting derivation is a well-formed derivation of $\sigma \trianglelefteq^1 \tau$.

(P):

By the induction hypothesis, we may assume that the premise of the rule is in well-formed form and ends with rule (P). Then the application of Proposition 3.7 gives a well-formed derivation immediately. □

4. Polynomial cases

In this section we introduce polynomial algorithms that decide affine retractability for types of order 1, 2 and 3. We present them as separate procedures instead of a combined one. This is because we want to analyse the running time of the algorithms separately.

The algorithm for order 1 is obvious as we can only use the rule (Ax) in this case.

Definition 8 (Algorithm for order 1). Let $\sigma = a$ and $\tau = b$ be the input types, where a, b are atomic. If $a = b$, the affine retractability holds, otherwise it does not.

This algorithm runs in constant time when the number of atoms is limited. When there is no limit to the number, the type atoms must be encoded in some way using the input alphabet (for example, in unary or binary representation). Comparing two strings that have a size of $O(n)$, where n is the size of the initial input, requires $O(n)$ time (on a two tape Turing machine).

The algorithm for the types in $\mathrm{T}^2_{\to}$ (order 2) requires an additional observation.

Proposition 4.1 (Rule (D) in low orders). If σ, τ have order strictly less than 4, then for each derivation of $\sigma \trianglelefteq^1 \tau$ there exists one that does not involve rule (D).

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Proof. As τ has order at most 3, the right premise of rule (D) must be $\sigma_1 \trianglelefteq^1 \tau_1$, where the order of τ_1 is at most 1. Thus the order of σ_1 must be 1 as well. However, in such an application of (D) the right premise can also be obtained by rule (N). □

Proposition 4.2 (Order and number of arguments). Let $\sigma \trianglelefteq^1 \tau$ be derivable.

1 $\mathrm{ord}(\sigma) \leqslant \mathrm{ord}(\tau)$.
2 If the order of σ and τ is less than 3, then $\sigma = \sigma_1 \to \cdots \to \sigma_n \to a$ and $\tau = \tau_1 \to \cdots \to \tau_m \to a$ with $n \leqslant m$.

Proof. The proof is a routine check, which is left for the reader. □

Definition 9 (Weight of types). For types of order 2 we introduce a weight function $\# : \mathrm{T}^2_{\to} \to \mathbb{N}^{\mathscr{B}}$ defined by

$$\#(\sigma)(a) = |\{i \mid \sigma = \sigma_1 \to \cdots \to \sigma_n \to b, \text{ and } \sigma_i = a\}|.$$

We use the partial order $\leqslant$ on elements of $\mathbb{N}^{\mathscr{B}}$ defined by $f \leqslant g$ if and only if $f(a) \leqslant g(a)$ for each $a \in \mathscr{B}$.

Example 9. The type $\rho = a \to b \to a \to a \to b \to a$ has

$$\#(a \to b \to a \to a \to b \to a)(a) = 3$$
$$\#(a \to b \to a \to a \to b \to a)(b) = 2.$$

Proposition 4.3 (The weight and $\trianglelefteq^1$). If $\sigma, \tau \in \mathrm{T}^2_{\to}$ and $head(\sigma) = head(\tau)$, then $\sigma \trianglelefteq^1 \tau$ if and only if $\#(\sigma) \leqslant \#(\tau)$.

Proof. The proof is by induction on the number n of arguments in the type σ.

($n = 1$) Let $\sigma = b \to a$ where $b \in \mathscr{B}$.

($\Rightarrow$) If $\sigma \trianglelefteq^1 \tau$ has a derivation, we can assume that the derivation is well formed. The derivation ends with rule (P) preceded by rule (H) or (N) (as the types have order 2). In the former case, the left premise of rule (H) must have the form $b \trianglelefteq^1 b$, since otherwise the derivation cannot be constructed. This means that $\#(\sigma)(b) = 1 \leqslant \#(\tau)(b)$ for b. For all other $c \in \mathscr{B}$, we have $\#(\sigma)(c) = 0$, so $\#(\sigma) \leqslant \#(\tau)$. When rule (P) is preceded by rule (N), the induction hypothesis applied to the premise $\sigma' \trianglelefteq^1 \tau'$ of (N) gives us $\#(\sigma') \leqslant \#(\tau')$, but we also have, by the definition of rule (N), $\#(\sigma) = \#(\sigma')$ and $\#(\tau') \leqslant \#(\tau)$, which implies our claim here.

($\Leftarrow$) If $\#(\sigma) \leqslant \#(\tau)$, there is at least one argument of τ that is equal to b. We construct a derivation for $\sigma \trianglelefteq^1 \tau$ by first permuting τ so that an argument b occurs in the first place, then we perform rule (H) with (Ax) as the left premise. Rule (N) is then applied to the right premise so that τ loses all its arguments. The beginning of the whole derivation is then (Ax) for $a \trianglelefteq^1 a$.

($n > 1$) Let $\sigma = \sigma_1 \to \cdots \to \sigma_n \to a$.

($\Rightarrow$) If $\sigma \trianglelefteq^1 \tau$ is derivable, we may assume that the derivation is well formed. The derivation ends with rule (P) preceded by rule (H). The left premise of the rule is $b \trianglelefteq^1 b$ for a certain $b \in \mathscr{B}$. The right premise of the rule

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

is $\sigma' \trianglelefteq^1 \tau'$ where σ' has one b less than σ and τ' has one b less than τ. This means that $\#(\sigma')(c) = \#(\sigma)(c)$ and $\#(\tau')(c) = \#(\tau)(c)$ for $c \neq b$, whereas $\#(\sigma')(b)+1 = \#(\sigma)(b)$ and $\#(\tau')(b)+1 = \#(\tau)(b)$. By the induction hypothesis, we have $\#(\sigma') \leqslant \#(\tau')$. This, together with the observation concerning the difference between $\#$ on types with primes and without primes, implies that $\#(\sigma) \leqslant \#(\tau)$.

($\Leftarrow$)If $\#(\sigma) \leqslant \#(\tau)$, assume that $\sigma_1 = b$. This means that $1 \leqslant \#(\sigma)(b) \leqslant \#(\tau)(b)$ and, furthermore, that τ has an argument equal to b. We can construct a derivation ending with rule (P) that permutes σ and τ so that both begin with b. Next we use rule (H) with the left premise $b \trianglelefteq^1 b$. The right premise is $\sigma' \trianglelefteq^1 \tau'$ such that

$$\#(\sigma')(c) = \begin{cases} \#(\sigma)(c) \text{ for } c \neq b \\ \#(\sigma)(b) - 1 \text{ for } c = b \end{cases} \qquad \#(\tau')(c) = \begin{cases} \#(\tau)(c) \text{ for } c \neq b \\ \#(\tau)(b) - 1 \text{ for } c = b. \end{cases}$$

This means that $\#(\sigma') \leqslant \#(\tau')$. By the induction hypothesis, we then have that $\sigma' \trianglelefteq^1 \tau'$ has a derivation, which completes the proof. □

Definition 10 (Algorithm for order 2). Let σ and τ be the input types. If $head(\sigma) \neq head(\tau)$, the affine retractability does not hold. Otherwise, if $\#(\sigma) \leqslant \#(\tau)$, then $\sigma \trianglelefteq^1 \tau$.

Note that the above algorithm runs in linear time with respect to the size of the types.

This algorithm for types σ, τ in Example 1 gives the answer true as $\#(\sigma)(a) = 1 \leqslant 1 = \#(\tau)(a)$ and $\#(\sigma)(b) = 0 \leqslant 1 = \#(\tau)(b)$. When we compare the same τ with ρ from Example 9, we have $\#(\tau)(a) \leqslant 3 = \#(\rho)(a)$ and $\#(\tau)(b) \leqslant 2 = \#(\rho)(b)$, so the answer is again true.

In order to present the algorithm for order 3, we need an additional notion from graph theory.

Definition 11 (Maximal matching). Let $G = (V_1 \cup V_2, E)$ be a bipartite graph. We say that a set of edges E' is a *matching* in G when each element of $V_1 \cup V_2$ is adjacent to at most one edge from E'. A matching E' is a *maximal matching* in G when there is no matching E'' in G such that $|E''| > |E'|$.

The correctness of the algorithm defined below for order 3 is based on Proposition 4.1 and the correctness of the algorithm for order 2.

Definition 12 (Algorithm for order 3). Let σ and τ be the input types. When the types are of order 2, we use the algorithm from Definition 10. When $head(\sigma) \neq head(\tau)$ we know that the retractability does not hold. When $head(\sigma) = head(\tau)$ we construct a bipartite graph $G = (V_1 \cup V_2, E)$ in which

$$V_1 = \{\sigma_i \mid 1 \leqslant i \leqslant k \text{ and } \sigma = \sigma_1 \to \cdots \to \sigma_k \to a\}$$
$$V_2 = \{\tau_i \mid 1 \leqslant i \leqslant l \text{ and } \tau = \tau_1 \to \cdots \to \tau_l \to a\}$$
$$E = \{(\sigma_i, \tau_j) \mid \sigma_i \trianglelefteq^1 \tau_i, \sigma_i \in V_1, \tau_j \in V_2\}.$$

Now we look for a maximal matching in G. If there is one of the size of V_1, then $\sigma \trianglelefteq^1 \tau$ holds, otherwise it does not hold.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

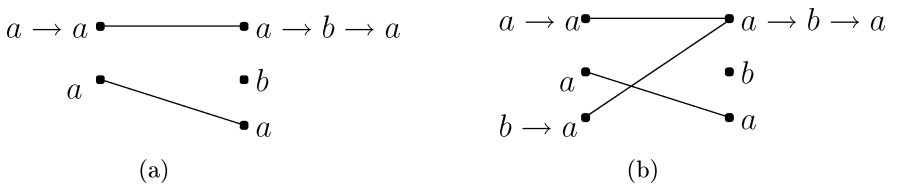


Fig. 8. Examples of bipartite graphs resulting from the algorithm in Definition 12: (a) when the inequality $\trianglelefteq^1$ holds and (b) when it does not hold.

Note that the construction of the graph takes $O(n^2)$ where n is the size of the input and the size of G is $O(n^2)$. The running time for a maximal matching procedure is $O(m^3)$ where m is the size of an input graph (see, for example, Cormen *et al.* (1990, Chapter 27)). This makes the overall running time $O(n^6)$.

Example 10. Consider the types σ, τ from Example 2. The bipartite graph constructed for them in the algorithm in Definition 12 is shown in Figure 8(a). If we replace σ by $\sigma' = (a \to a) \to a \to (b \to a) \to a$, the graph changes shape to the one shown in Figure 8(b). Note that in the former case the maximal matching does indeed have the required size 2, while in the latter case the size remains the same, but the required size is equal to 3, so the algorithm states that the inequality $\trianglelefteq^1$ does not hold, which is exactly the case.

5. NP-completeness

The proof of NP-completeness is split into the usual two parts: the first shows that the problem is in NP and the second that the problem is NP-hard. The proof of NP-hardness we provide here is parametrised so that we are able to conduct a further discussion of complexity in different sub-cases depending on the number of type atoms used.

5.1. *Non-deterministic polynomial algorithm*

Theorem 5.1 (Affine retractions are in NP). There is a non-deterministic polynomial algorithm that solves the problem of affine retractions.

Proof. The problem of affine retractions immediately reduces to the problem of finding an affine solution to the following higher-order matching equation $X(Yx) = x$, where X, Y are unknowns of the types $\tau \to \sigma$ and $\sigma \to \tau$, respectively, and x is a constant of the type σ. The problem of solving such equations has been proved NP-complete in Wierzbicki and Dougherty (2002).

A more direct argument could rely on the observation that in the system in Figure 5, for each rule, the different inequalities in the assumptions of the rule can be constructed from disjoint parts of the inequalities in the result and no part of the result is duplicated in the assumptions. Thus each derivation using the system has only polynomially many uses of these rules. This means that it is enough to guess such a polynomial tree labelled with the rules and then check if the tree is a proper derivation. □

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

5.2. *NP-hardness*

In this section we present a generic NP-hardness proof, which is then instantiated for three languages of types: in the first we deal with an unbounded number of type atoms; in the second the number of atoms is bounded by a fixed number $k > 1$; and in the third there is only one type atom. In order to reuse a part of the NP-hardness construction, we have to define a few notions that describe requirements on the language of types that enables the NP-hardness construction.

We assume that the set of basic types $\mathscr{B}$ contains an atom a. We deal with families of sets of types of the form $\mathscr{A} = \{A_i\}_{i\in\mathbb{N}}$ where $|A_i| = i$. Here we call such families *graded families* for short. We define $\sigma' = \sigma \to a$ for $\sigma \in \bigcup_{i\in\mathbb{N}} A_i$. The idea behind the requirements below is to provide a coding of the variables that does not interfere with the top-level NP-hardness encoding.

Definition 13 (Properly isolated family). We say that a graded family $\{A_i\}_{i\in\mathbb{N}}$ is *properly isolated* if for each i and for all $\sigma_\alpha, \sigma_{\alpha_1}, \ldots, \sigma_{\alpha_n}, \sigma_{\beta_1}, \ldots, \sigma_{\beta_m} \in A_i$, we have:

1 If $\sigma_{\alpha_1} \preceq^1 \sigma_{\beta_1}$, then $\sigma_{\alpha_1} = \sigma_{\beta_1}$.
2 If $\sigma_\alpha \to a \preceq^1 (\sigma'_{\alpha_1} \to \cdots \to \sigma'_{\alpha_n} \to a) \to a$, then for some j we have $\sigma_\alpha \preceq^1 \sigma_{\alpha_j}$.
3 $\sigma_\alpha \not\preceq^1 \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a$.
4 If $\sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \preceq^1 \sigma_{\beta_1} \to \cdots \to \sigma_{\beta_m} \to a$, then $\{\alpha_1, \ldots, \alpha_n\} \subseteq \{\beta_1, \ldots, \beta_m\}$ where $\subseteq$ is the multiset inclusion.
5 If $\sigma_\alpha \to a \preceq^1 ((\sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a) \to (\sigma_{\beta_1} \to \cdots \to \sigma_{\beta_m} \to a) \to a) \to a$, then $\sigma_\alpha \in \{\sigma_{\alpha_1}, \ldots, \sigma_{\alpha_n}, \sigma_{\beta_1}, \ldots, \sigma_{\beta_m}\}$.

We defer giving examples of such families to Section 5.3, where we conclude our NP-completeness results.

The proof of NP-hardness relies on a reduction of the 3-SAT problem to the problem of affine retractions. First we present a translation of 3-SAT problem instances into instances of the affine retractions problem. Then we show that the instance of the former has a solution if and only if there is a pair of terms that is a witness of retractability for the translation.

We fix a set X of denumerably many propositional variables. Let ϕ be an instance of 3-SAT. The formula has the form

$$\phi = \bigwedge_{i=1}^{n} \phi_i \tag{6}$$

where $\phi_i = y_{i,1} \vee y_{i,2} \vee y_{i,3}$ with $y_{i,j} = x$ or $y_{i,j} = \neg x$ for some $x \in X$. Let $\mathrm{PV}(\phi) = \{x_1, \ldots, x_m\}$ be the set of all propositional variables occurring in ϕ and let

$$Y_0 = \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}.$$

The set Y_0 is called the set of *literals* in ϕ. For each literal $y = \neg x$ we define $\neg y$ to be x. As usual, a valuation $v : X \to \{0, 1\}$ allows us to define a value $v(\phi)$ of the formula ϕ. This is done by induction on the structure of ϕ using the usual semantics of the connectives $\wedge, \vee$ and $\neg$.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

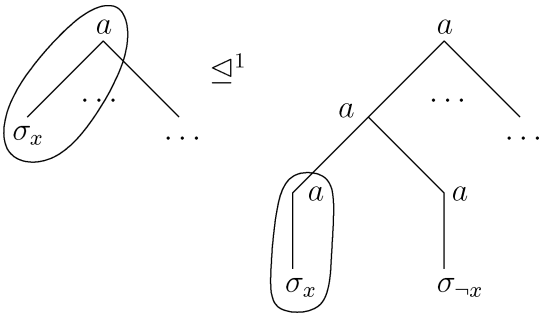


Fig. 9. Application of rule (D) to remove a variable

Suppose we have a properly isolated family $\{A_i\}_{i\in\mathbb{N}}$. Given the above-mentioned notation for subformulae of ϕ, we fix the notation for elements of A_{2m+n} as

$$A_{2m+n} = \{\sigma_{x_1}, \ldots, \sigma_{x_m}, \sigma_{\neg x_1}, \ldots, \sigma_{\neg x_m}, \sigma_{\phi_1}, \ldots, \sigma_{\phi_n}\}.$$

Let

$$B_y = \{\phi_i \mid \phi_i \text{ is a subformula of } \phi \text{ with } \phi_i = y_1 \vee y_2 \vee y_3, \\ y_1, y_2, y_3 \in Y_0, \text{ and } y \text{ is one of } y_1, y_2, y_3\}.$$

For each literal $y \in Y_0$, we define $\overline{\tau}_{B_y} = \sigma_{\neg y} \to \sigma_{\phi_{i_1}} \to \cdots \to \sigma_{\phi_{i_k}} \to a$ where $B_y = \{\phi_{i_1}, \ldots, \phi_{i_k}\}$. This allows us to define $\tau_{B_x} = \overline{\tau}_{B_x} \to \overline{\tau}_{B_{\neg x}} \to a$. Moreover, for $x \in \mathrm{PV}(\phi)$, we define a type $\tau_x = \sigma'_x \to \sigma'_{\neg x} \to a$ (recall that $\sigma'_x = \sigma_x \to a$).

This allows us to translate ϕ into an instance of the affine retractions problem σ_ϕ, τ_ϕ:

$$\begin{aligned} \sigma_\phi &= \sigma_{x_1} \to \cdots \to \sigma_{x_m} \to \sigma_{\neg x_1} \to \cdots \to \sigma_{\neg x_m} \to \sigma_{\phi_1} \to \cdots \to \sigma_{\phi_n} \to a \\ \tau_\phi &= \tau_{x_1} \to \cdots \to \tau_{x_m} \to \tau_{B_{x_1}} \to \cdots \to \tau_{B_{x_m}} \to a\,. \end{aligned} \tag{7}$$

Example 11. In order to simplify the presentation, we consider here a 2-SAT formula: $\phi = (x \vee y) \wedge (x \vee \neg y)$. Suppose that we have an isolated family such that $\mathscr{A}_6 = \{a_x, a_y, a_{\neg x}, a_{\neg y}, a_{x\vee y}, a_{x\vee\neg y}\}$. We can now transform the formulae using the prescription above so that

$$\begin{aligned} \sigma_\phi &= a_x \to a_y \to a_{\neg x} \to a_{\neg y} \to a_{x\vee y} \to a_{x\vee\neg y} \to a, \\ \tau_\phi &= \tau_x \to \tau_y \to \tau_{B_x} \to \tau_{B_y} \to a \\ &= (a'_x \to a'_{\neg x} \to a) \to (a'_y \to a'_{\neg y} \to a) \to \\ &\quad (\overline{\tau}_{B_x} \to \overline{\tau}_{B_{\neg x}} \to a) \to (\overline{\tau}_{B_y} \to \overline{\tau}_{B_{\neg y}} \to a) \to a \\ &= ((a_x \to a) \to (a_{\neg x} \to a) \to a) \to ((a_y \to a) \to (a_{\neg y} \to a) \to a) \to \\ &\quad ((a_{\neg x} \to a_{x\vee y} \to a_{x\vee\neg y} \to a) \to (a_x \to a) \to a) \to \\ &\quad ((a_{\neg y} \to a_{x\vee y} \to a) \to (a_y \to a_{x\vee\neg y} \to a) \to a) \to a\,. \end{aligned}$$

We can try to construct a retraction so that it encodes the type $\sigma_x = a_x$ into the first argument τ_x of τ_ϕ. This is shown on Figure 9. We can now see the mechanism of blocking that was pointed out in Example 3. The choice for a_x blocks the possibility of encoding the type $\sigma_{\neg x} = a_{\neg x}$ in the first argument. This can be exploited as a way of defining a valuation of the propositional variables – when a_x is encoded into τ_x, the valuation assigns

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

1 to x. Of course, all the other arguments of σ_ϕ must be encoded in some argument of τ_ϕ. We use the type τ_{B_x} to encode $a_{\neg x}$ together with all the disjunctive subformulae of ϕ that evaluate to 1 under a valuation that assigns 1 to x. (Note that we can also encode a_x into τ_{B_x} together with all the disjunctive subformulae of ϕ that evaluate to 1 under a valuation that assigns 0 to x, but this cannot be done at the same time.) This mechanism allows us to encode all the arguments of σ provided we have a valuation that makes all the conjuncts in ϕ evaluate to 1.

Lemma 5.2 (A valuation induces a derivation). Let $v : X \to \{0,1\}$ be a valuation of propositional variables. If $v(\phi) = 1$, there is a derivation for $\sigma_\phi \trianglelefteq^1 \tau_\phi$.

Proof. In order to prove the lemma, we have to generalise it so that it holds for a wider range of τ_ϕ. We say that types $\sigma_\phi, \hat{\tau}_\phi$ are an acceptable coding of ϕ if σ_ϕ is as before and $\hat{\tau}_\phi$ has the form $\hat{\tau}_\phi = \tau_{x_1} \to \cdots \to \tau_{x_m} \to \hat{\tau}_{B_{x_1}} \to \cdots \to \hat{\tau}_{B_{x_m}} \to a$ where $\hat{\tau}_{B_y} = \tilde{\tau}_{B_y} \to \tilde{\tau}_{B_{\neg y}} \to a$ and $\tilde{\tau}_{B_y} = \Sigma \to a$ with Σ being a multiset $\{\sigma_{\neg y}, \sigma_{\phi_{i_1}}, \ldots, \sigma_{\phi_{i_k}}, \rho_1, \ldots, \rho_l\}$ for $y \in Y_0$, $\rho_1, \ldots, \rho_l$ being arbitrary types, and $B_y = \{\phi_{i_1}, \ldots, \phi_{i_k}\}$.

We produce a derivation for $\sigma_\phi \trianglelefteq^1 \hat{\tau}_\phi$ where $\sigma_\phi, \hat{\tau}_\phi$ are an acceptable coding of ϕ by induction on the number of variables in ϕ. First we permute $\sigma_\phi \trianglelefteq^1 \hat{\tau}_\phi$ using (P) and reduce the task of finding the derivation to the task of finding one for

$$\begin{array}{llll} \sigma^1_\phi = \sigma_{x_1} \to \sigma^2_\phi & & \hat{\tau}^1_\phi = \tau_{x_1} \to \hat{\tau}^2_\phi & \text{when } v(x_1) = 1 \\ \sigma^1_\phi = \sigma_{\neg x_1} \to \sigma^2_\phi & & \hat{\tau}^1_\phi = \tau_{x_1} \to \hat{\tau}^2_\phi & \text{when } v(x_1) = 0\,. \end{array} \tag{8}$$

We apply rule (D) as follows (we omit terms for clarity):

$$\text{(D)}\ \frac{\vdash \sigma_y \trianglelefteq^1 \sigma_y \qquad \vdash \sigma^2_{\phi_2} \trianglelefteq^1 \hat{\tau}^2_\phi}{\vdash \sigma_y \to \sigma^2_\phi \trianglelefteq^1 ((\sigma_{x_1} \to a) \to (\sigma_{\neg x_1} \to a) \to a) \to \hat{\tau}^2_\phi}$$

where $y = x_1$ or $\neg x_1$ depending on the value of $v(x_1)$. The derivation for $\vdash \sigma_y \trianglelefteq^1 \sigma_y$ can be completed by an iterative use of rule (H).

In order to construct a derivation for the right premise of rule (D), we permute $\sigma^2_\phi \trianglelefteq^1 \hat{\tau}^2_\phi$ so that we reduce the problem of finding the derivation to the problem of finding one for

$$\sigma_{\neg y} \to \sigma_{\phi_{i_1}} \to \cdots \to \sigma_{\phi_{i_r}} \to \sigma^3_\phi \trianglelefteq^1 \hat{\tau}_{B_{x_1}} \to \hat{\tau}^3_\phi \tag{9}$$

where y is either x_1 (when $v(x_1) = 0$) or $\neg x_1$ (when $v(x_1) = 1$), and $\{\phi_{i_1}, \ldots, \phi_{i_r}\} = B_{\neg y}$. We again apply rule (D) as suggested in Figure 10 – the types surrounded by the dashed line become the left premise and $\vdash \sigma^3_\phi \trianglelefteq^1 \hat{\tau}^3_\phi$ becomes the right premise. The left premise of the rule can be derived by repeated application of rule (H) to get rid of σ_α, preceded by an application of (N) to get rid of ρ_β for $\beta = 1, \ldots, l$.

The types $\sigma^3_\phi, \hat{\tau}^3_\phi$ are now an acceptable coding of a formula ϕ', which is constructed from ϕ by erasing all the subformulae from $B_y \cup B_{\neg y}$. Note that the types are an acceptable coding even though they still contain subtypes that encode the elements of $B_y \cup B_{\neg y}$. Thus we obtain by the induction hypothesis a derivation for $\sigma^3_\phi \trianglelefteq^1 \hat{\tau}^3_\phi$. This completes the proof. □

Lemma 5.3 (A derivation induces a valuation). If $\sigma_\phi \trianglelefteq^1 \tau_\phi$ has a derivation, there exists a valuation $v : \mathrm{PV}(\phi) \to \{0,1\}$ such that $v(\phi) = 1$.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

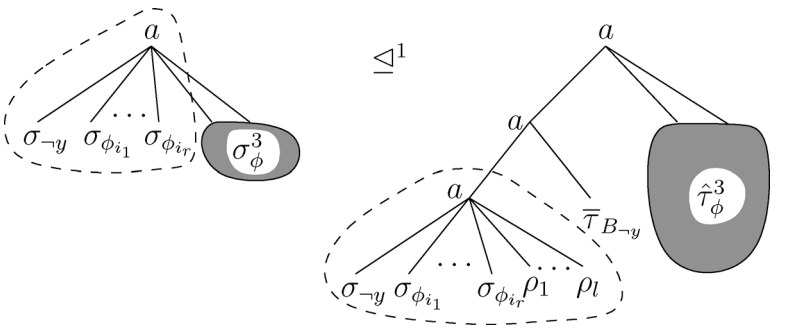


Fig. 10. Application of rule (D) to decompose inequality (9)

Proof. In order to prove the lemma, we have to generalise it to cover a wider class of terms σ_ϕ and τ_ϕ. We say that for a given formula ϕ and a partial function $v : \mathrm{PV}(\phi) \rightharpoonup \{0,1\}$, types σ_ϕ, τ_ϕ are an acceptable coding for ϕ, v if

(1) we have

$$\begin{aligned}\sigma_\phi &= \sigma_{x_1} \to \cdots \to \sigma_{x_m} \to \sigma_{\neg x_1} \to \cdots \to \sigma_{\neg x_m} \to \sigma_{\phi_1} \to \cdots \to \sigma_{\phi_n} \to a\\ \tau_\phi &= \tau_{x_1} \to \cdots \to \tau_{x_m} \to \tau_{B_{x_1}} \to \cdots \to \tau_{B_{x_m}} \to a\end{aligned}$$

where $\{x_1, \ldots, x_m\} \subseteq \mathrm{PV}(\phi) \backslash \mathrm{Dom}(v)$; and

(2) for each conjunct ψ of the formula ϕ such that $\psi \notin \{\phi_1, \ldots, \phi_n\}$ we have $v(\psi) = 1$.

We prove the following generalisation of our lemma:

> *Let σ_ϕ, τ_ϕ be an acceptable coding for a formula ϕ and a partial function $v : \mathrm{PV}(\phi) \rightharpoonup \{0,1\}$. If $\sigma_\phi \trianglelefteq^1 \tau_\phi$ has a derivation, there exists a valuation $v' : \mathrm{PV}(\phi) \to \{0,1\}$ that extends v and for which $v'(\phi) = 1$.*

The proof is by induction on the number of variables in $\mathrm{PV}(\phi) \backslash \mathrm{Dom}(v)$:

$|\mathrm{PV}(\phi) \backslash \mathrm{Dom}(v)| = 0$:

As σ_ϕ, τ_ϕ is an acceptable coding for a formula ϕ, we have $\sigma = a$, so for all conjuncts ψ of ϕ, the condition $v(\psi) = 1$ holds. This implies that $v(\phi) = 1$, which completes the proof in this case.

$|\mathrm{PV}(\phi) \backslash \mathrm{Dom}(v)| > 0$:

If there is a derivation D for $\sigma_\phi \trianglelefteq^1 \tau_\phi$, then, by Corollary 3.14, we may assume that it is a well-formed derivation. By Proposition 3.10, we may further assume that the derivation ends with rule (P) preceded by a certain number of (H) and (D) rules applied to the subsequent right premises with a single (N) rule at the beginning preceded by the axiom. Moreover, we assume that no rule (D) can be replaced by an equivalent rule (N). A derivation of this form is called a *very well-formed* derivation. We can transform the derivation D by repeated application of Proposition 3.12 to

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

obtain a very well-formed derivation D' such that the first (P) rule is preceded by one of:

1 rule $(\mathrm{D})^{11}$ in which the left premise is $\sigma_{x_1} \trianglelefteq^1 \sigma_\alpha$ for some propositional literal $\alpha \in \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}$ and σ_α occurring in $\tau_{\tilde{\alpha}}$ where $\tilde{\alpha}$ is a propositional variable;

2 rule $(\mathrm{D})^{12}$ in which the left premise is

$$\sigma_{x_1} \to \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \sigma_{\beta_1} \to \cdots \to \sigma_{\beta_m} \to a$$

for certain $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}$, where the right-hand side of the inequality comes from some τ_{B_y};

3 rule $(\mathrm{N})^1$; or

4 rule $(\mathrm{Ax})^1$.

(Note that the application of rule (H) is impossible by Condition (3) on the properly isolated family.)

We now consider each of these cases:

1 We have $\sigma_\alpha = \sigma_{x_1}$. This is guaranteed by Condition (2) on the properly isolated family and by the fact that only τ_{x_1} contains an occurrence of σ_{x_1}. Define D_R to be the derivation for the right premise of rule $(\mathrm{D})^{11}$. We extend v to $\tilde{v}$ so that $\tilde{v}(x_1) = 1$. By repeated application of Proposition 3.12, we can transform D_R into a very well-formed derivation D'_R that ends with rule (P) preceded by:

(i) rule $(\mathrm{D})^{21}$ in which the left premise is $\sigma_{\neg x_1} \trianglelefteq^1 \sigma_\beta$ with σ_β occurring in some $\tau_{\tilde{\beta}}$ where $\tilde{\beta}$ is a propositional variable; or

(ii) rule $(\mathrm{D})^{22}$ in which the left premise is

$$\sigma_{\neg x_1} \to \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \sigma_{\beta_1} \to \cdots \to \sigma_{\beta_m} \to a \tag{10}$$

for certain $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}$, where the right-hand side of the inequality comes from some τ_{B_y}; or

(iii) rule $(\mathrm{N})^2$.

(Again, (H) is impossible by Condition (3).)

We now carry out the proof for each of these cases:

(i) Note that this is only possible if $\tilde{\beta} = x_1$. This is guaranteed by Condition (2) on the properly isolated family. This is impossible, however, as τ_{x_1} is introduced only in the left premise of $(\mathrm{D})^{11}$ so it does not occur here, so this case is impossible.

(ii) Note that Condition (4) of Definition 13 implies that $\neg x_1 \in \{\beta_1, \ldots, \beta_m\}$. This is possible only by an application of rule $(\mathrm{D})^{22}$ so that $B_y = B_{x_1}$. This implies that the inequality (10) is in fact $\sigma_{\neg x_1} \to \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \sigma_{\neg x_1} \to \sigma_{\phi_{i_1}} \to \cdots \to \sigma_{\phi_{i_r}} \to a$ where $\{\phi_{i_1}, \ldots, \phi_{i_r}\} = B_{x_1}$. Condition (4) on the properly isolated family implies that $\{\alpha_1, \ldots, \alpha_n\} \subseteq \{\phi_{i_1}, \ldots, \phi_{i_r}\}$ (where the inclusion is taken as for multisets). This means that the right premise of $(\mathrm{D})^{22}$, possibly

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

after an application of rule (P), has the form

$$\sigma_{x_2} \to \cdots \to \sigma_{x_m} \to \sigma_{\neg x_2} \to \cdots \to \sigma_{\neg x_m} \to \sigma_{\phi_{j_1}} \to \cdots \to \sigma_{\phi_{j_l}} \to$$
$$\trianglelefteq^1$$
$$\tau_{x_2} \to \cdots \to \tau_{x_m} \to \tau_{B_{x_2}} \to \cdots \to \tau_{B_{x_m}} \to a\,.$$

The variables $x_2, \ldots, x_m$ belong to $\mathrm{PV}(\phi)\backslash\mathrm{Dom}(\tilde{v})$, while $\tilde{v}(x_1) = 1$ together with the definition of $\tau_{B_{x_1}}$ guarantees that $\tilde{v}(\alpha_i) = 1$ for $\alpha_1 \in \{\alpha_1, \ldots, \alpha_n\}$, so this pair of types is an acceptable coding of ϕ. As this premise has its derivation, the induction hypothesis implies that $\tilde{v}$ can be extended to a valuation v' such that $v'(\phi) = 1$.

(iii) In this case we exploit the fact that the derivation D' is very well formed. This implies that the premise we consider has the form $a \trianglelefteq^1 \tau_1 \to \cdots \to \tau_r \to a$. This is impossible though, as each σ_x on the left-hand side of the original inequality is accompanied by $\sigma_{\neg x}$.

2 The proof for this case is similar to the proof for case (1), but we have to define $\tilde{v}(x_1) = 0$, and the only non-trivial situation is the one corresponding to the use of rule $(\mathrm{D})^{12}$ preceded by the use of rule $(\mathrm{D})^{21}$. This is done by an analysis similar to the one in the remaining parts of case (1).

3 In this case we exploit the fact that the derivation is well formed. This implies that the premise we consider has the form $a \trianglelefteq^1 \tau_1 \to \cdots \to \tau_r \to a$. This is impossible, however, as each σ_x on the left-hand side of the original inequality is accompanied by $\sigma_{\neg x}$.

4 In this case we arrive at an impossible situation as both σ_ϕ and τ_ϕ must be equal to a. This is impossible as ϕ contains at least one variable. □

5.3. *Properly isolated families*

Now we are in a position to define particular families of types that will give us the NP-hardness proofs for type systems restricted in different ways (that is, for a type system with no restriction on the number of atoms, for a type system with a fixed number of type atoms, and for a system with a single atom).

Definition 14 (Families of types).

— A family of types $\mathscr{A}^\infty = \{A_i^\infty\}_{i\in\mathbb{N}}$ is defined so that $A_n^\infty = \{a_1, \ldots, a_n\}$ where a_i for $i = 1, \ldots, n$ are distinct type atoms and each $a_i \neq a$ where a is the atom used in Section 5.2.

— A family of types $\mathscr{A}^2 = \{A_i^2\}_{i\in\mathbb{N}}$ is defined so that $A_n^2 = \{\sigma_1, \ldots, \sigma_n\}$ where $\sigma_i = \underbrace{a \to \cdots \to a}_{n-i \text{ times}} \to \underbrace{b \to \cdots \to b}_{i+1 \text{ times}} \to b$ for $i = 1, \ldots, n$ and $b \neq a$ from Section 5.2.

— A family of types $\mathscr{A}^1 = \{A_i^1\}_{i\in\mathbb{N}}$ is defined so that $A_n^1 = \{\sigma_1, \ldots, \sigma_n\}$ where $\sigma_i = (\tilde{\sigma}_{2n-i+2} \to \tilde{\sigma}_{i+1} \to a) \to a$ for $i = 1, \ldots, n$ with $\tilde{\sigma}_i = \underbrace{a \to \cdots \to a}_{i \text{ times}} \to a$.

Note that the types in $\mathscr{A}^\infty$ have order 1, the types in $\mathscr{A}^2$ have order 2 while the types in $\mathscr{A}^1$ have order 4.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Lemma 5.4. isolation

(A) The family $\mathscr{A}^\infty$ is a properly isolated family of types built from infinitely many atoms.
(B) The family $\mathscr{A}^2$ is a properly isolated family with k atomic types where $k \geqslant 2$.
(C) The family $\mathscr{A}^1$ is a properly isolated family of types built from a single atomic type.

Proof.

(A) Recall that $\mathscr{A}^\infty = \{A_i^\infty\}_{i\in\mathbb{N}}$ is such that $A_n^\infty = \{a_1, \ldots, a_n\}$ where a_i for $i = 1, \ldots, n$ are distinct type atoms and each $a_i \neq a$ where a is the atom used in Section 5.2. We will now prove each of the necessary properties listed in Definition 13.

1 This property holds since for atomic types we may only use the (Ax) rule, so σ, τ must be equal for $\sigma \trianglelefteq^1 \tau$ to hold.

2 Suppose $\sigma_\alpha \to a \trianglelefteq^1 (\sigma'_{\alpha_1} \to \cdots \to \sigma'_{\alpha_n} \to a) \to a$. The use of rule (P), (N) or (Ax) is trivially impossible here. Rule (H) cannot be used either as $head(\sigma_\alpha) \neq a$. The only possible use of (D) implies that $\sigma_\alpha \trianglelefteq^1 \sigma_{\alpha_j}$ or $\sigma_\alpha \trianglelefteq^1 \sigma_{\neg\alpha_j}$. The latter case is impossible as $\sigma_\alpha, \sigma_{\neg\alpha_j}$ are different atoms.

3 In $\sigma_\alpha \trianglelefteq^1 \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a$, the head of the left-hand side is different from the head of the right-hand side, so the inequality cannot hold.

4 Suppose we have a derivation for

$$\sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \sigma_{\beta_1} \to \cdots \to \sigma_{\beta_m} \to a\,.$$

We can reorder the derivation to obtain a well-formed derivation. As the types in the inequality have order 2, rule (D) cannot be used. This means that the only rule that can be applied before the (P) rule is rule (H) (we can omit the (N) rules using Proposition 3.10). The left premise of such a rule may only be derived by the axiom (Ax), which means that for appropriate i, j we have $\alpha_i = \beta_j$. As all the subtypes $\{\sigma_{\alpha_1}, \ldots, \sigma_{\alpha_n}\}$ must be used and each must be assigned to a different σ_{β_j}, we obtain the expected inclusion.

5 The proof in this case is essentially the same as in case (2).

(B) Recall that $\mathscr{A}^2 = \{A_i^2\}_{i\in\mathbb{N}}$ is such that $A_n^2 = \{\sigma_1, \ldots, \sigma_n\}$ where σ_i for $i = 1, \ldots, n$ is defined as

$$\sigma_i = \underbrace{a \to \cdots \to a}_{n-i \text{ times}} \to \underbrace{b \to \cdots \to b}_{i+1 \text{ times}} \to b$$

where $b \neq a$ and b does not occur explicitly in the constructions from Section 5.2. We will now prove each of the necessary properties listed in Definition 13.

1 Suppose that $\sigma_{\alpha_1} \trianglelefteq^1 \sigma_{\beta_1}$ has a derivation. As the types have order 2, we may use Proposition 4.3 and obtain $\#(\sigma_{\alpha_1}) \leqslant \#(\sigma_{\beta_1})$. The definition of $\mathscr{A}^2$ implies that $n - \alpha_1 \leqslant n - \beta_1$ and $\alpha_1 + 1 \leqslant \beta_1 + 1$. This immediately gives $\alpha_1 = \beta_1$.

2 One can see immediately that the inequality $\sigma_\alpha \to a \trianglelefteq^1 (\sigma'_{\alpha_1} \to \cdots \to \sigma'_{\alpha_n} \to a) \to a$ cannot have a derivation that ends with (N), (P) or (Ax). Rule (H) cannot be the last one as $head(\sigma_\alpha) = b \neq a$. Thus the only remaining possibility is for the final rule to be (D). As a result, we obtain, *a priori*, $\sigma_\alpha \trianglelefteq^1 \sigma_{\alpha_i}$ for some $i = 1, \ldots, n$.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Proposition 4.3 implies that $\#(\sigma_\alpha) \leqslant \#(\sigma_{\alpha_i})$. By the definition of the family $\mathscr{A}^2$, this is possible only when $\alpha = \alpha_i$.

3 The inequality $\sigma_\alpha \trianglelefteq^1 \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a$ cannot have a derivation as $head(\sigma_\alpha) = b$, which is different from the head of the right-hand side.

4 Suppose that $\sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \sigma_{\beta_1} \to \cdots \to \sigma_{\beta_m} \to a$ has a derivation. As the types have order 3, the derivation cannot use rule (D) – see Proposition 4.1. This means that rule (H) must be used to get rid of subsequent $\sigma_{\alpha_1}, \ldots, \sigma_{\alpha_n}$ on the left-hand side. This can be obtained only by using different $\sigma_{\beta_{i_1}}, \ldots, \sigma_{\beta_{i_n}}$, respectively. This means that $\{\alpha_1, \ldots, \alpha_n\} = \{\beta_{i_1}, \ldots, \beta_{i_n}\}$, and, consequently, that $\{\alpha_1, \ldots, \alpha_n\} \subseteq \{\beta_1, \ldots, \beta_m\}$.

5 The proof in this case is essentially the same as in case (2).

(C) Recall that $\mathscr{A}^1 = \{A^1_i\}_{i \in \mathbb{N}}$ is such that $A^1_n = \{\sigma_1, \ldots, \sigma_n\}$ where σ_i for $i = 1, \ldots, n$ is defined as $\sigma_i = (\tilde{\sigma}_{2n-i+2} \to \tilde{\sigma}_{i+1} \to a) \to a$ where

$$\tilde{\sigma}_i = \underbrace{a \to \cdots \to a}_{i \text{ times}} \to a\,.$$

We will now prove each of the necessary properties listed in Definition 13.

1 Suppose that $\sigma_{\alpha_1} \trianglelefteq^1 \sigma_{\beta_1}$ has a derivation. We can assume by Corollary 3.14 that this derivation is well formed. As the σ_i have only one argument, we may assume the premise of the final (P) rule is the same as the result. We can now analyse the remaining almost well-formed part D of the derivation. There is no rule (D) at the end of D as it would result in us comparing σ_{α_1} with a type of order 2, which is impossible by Proposition 4.2. As the elements of $\mathscr{A}^1_i$ for $i \in \mathbb{N}$ have one argument, the final rule of D is (H), and in the left premise we obtain

$$\tilde{\sigma}_{2n-\alpha_1+2} \to \tilde{\sigma}_{\alpha_1+1} \to a \trianglelefteq^1 \tilde{\sigma}_{2n-\beta_1+2} \to \tilde{\sigma}_{\beta_1+1} \to a\,. \tag{11}$$

By Corollary 3.14, we may again assume the derivation for (11) is well formed. By Proposition 4.1, we cannot use rule (D) before (P). As the left-hand side has arguments, we may assume the rule before (P) is (H). Let us fix the notation of the result of the (P) rule to have the left premise of the subsequent (H) rule equal to $\tilde{\sigma}_{2n-\alpha_1+2} \trianglelefteq^1 \tilde{\sigma}_k$. As $i \leqslant n$, the inequality $2n - i + 2 > i + 1$ holds, so we have $\tilde{\sigma}_{2n-\alpha_1+2} \trianglelefteq^1 \tilde{\sigma}_{2n-\beta_1+2}$. Because these types are of order 2, we have, by Proposition 4.3, that $2n - \alpha_1 + 2 \leqslant 2n - \beta_1 + 2$ and thus $\alpha_1 \geqslant \beta_1$ (∗).
As we have two arguments in (11), we must also use rule (H) to derive the inequality $\tilde{\sigma}_{\alpha_1+1} \trianglelefteq^1 \tilde{\sigma}_{\beta_1+1}$. Proposition 4.3 gives us that $\alpha_1 + 1 \leqslant \beta_1 + 1$, so $\alpha_1 \leqslant \beta_1$. This, together with the inequality (∗), implies $\alpha_1 = \beta_1$.

2 Suppose that $\sigma_\alpha \to a \trianglelefteq^1 (\sigma'_{\alpha_1} \to \cdots \to \sigma'_{\alpha_n} \to a) \to a$ has a derivation. This derivation cannot end with rules (P) or (Ax). We have two cases based on the remaining possible final rule

— If the final rule is (H), we obtain in the left premise

$$(\tilde{\sigma}_{2n-\alpha+2} \to \tilde{\sigma}_{\alpha+1} \to a) \to a \trianglelefteq^1 \sigma'_{\alpha_1} \to \cdots \to \sigma'_{\alpha_n} \to a\,.$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Well-formed derivations for this inequality end with rule (P) preceded by either:

(a) Rule (H):
In this case we arrive at the inequality

$$\tilde{\sigma}_{2n-\alpha+2} \to \tilde{\sigma}_{\alpha+1} \to a \trianglelefteq^1 ((\tilde{\sigma}_{2n-\alpha_i+2} \to \tilde{\sigma}_{\alpha_i+1} \to a) \to a) \to a\,.$$

This could be derived again by either:

(i) Rule (H):
In this case we have to derive as the right premise

$$\tilde{\sigma}_k \to a \trianglelefteq^1 a\,,$$

which is impossible.

(ii) Rule (D).
In this case we must have both

$$2n-\alpha+2 \leqslant 2n-\alpha_i+2 \quad \text{and} \quad \alpha+1 \leqslant \alpha_i+1$$

or

$$2n-\alpha+2 \leqslant \alpha_i+1 \quad \text{and} \quad \alpha+1 \leqslant 2n-\alpha_i+2\,,$$

which is possible only when $\alpha = \alpha_i$.

(b) Rule (D):
In this case we get

$$(\tilde{\sigma}_{2n-\alpha+2} \to \tilde{\sigma}_{\alpha+1} \to a) \to a \trianglelefteq^1 \sigma_{\alpha_i} = (\tilde{\sigma}_{2n-\alpha_i+2} \to \tilde{\sigma}_{\alpha_i+1} \to a) \to a\,.$$

It follows immediately that $\alpha = \alpha_i$ in this case.

— If (P) is preceded by rule (D), we obtain in the left premise $\sigma_\alpha \trianglelefteq^1 \sigma_{\alpha_i}$ for some i. Hence, by property (1), we have $\alpha = \alpha_i$.

3 Suppose $\sigma_\alpha \trianglelefteq^1 \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a$ has a derivation. We may assume that the derivation is well formed. We have two cases depending on the rule that occurs before the final (P) rule.

— (P) is preceded by rule (H):
In this case we get the inequality

$$\tilde{\sigma}_{2n-\alpha+2} \to \tilde{\sigma}_{\alpha+1} \to a \trianglelefteq^1 (\tilde{\sigma}_{2n-\alpha_i+2} \to \tilde{\sigma}_{\alpha_i+1} \to a) \to a$$

in the left premise of this (H). The derivation of the inequality cannot end with rule (D) as this would result in comparing a type of order 3 with a type of order 2 or 1. However, the derivation cannot end with rule (H) either, since there are 2 arguments on the left-hand side and only one on the right-hand side, which gives a contradiction.

— (P) is preceded by rule (D):
In this case we get

$$((\tilde{\sigma}_{2n-\alpha+2} \to \tilde{\sigma}_{\alpha+1} \to a) \to a) \to a \trianglelefteq^1 \sigma_{\alpha_i}$$

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

in the left premise, which is impossible as the order of the right-hand side is less than the order of the left-hand side.

4 Suppose $\sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \sigma_{\beta_1} \to \cdots \to \sigma_{\beta_m} \to a$ has a derivation. We may assume that the derivation is well formed. We have two cases depending on the rule that occurs before the final (P) rule:

— (P) is preceded by rule (D):
In this case we get an inequality between a type of order 5 and a type of order 3 in the left premise of this (D), or this (D) rule can be replaced by the (N) rule. This means that the derivation in this case is impossible.

— (P) is preceded by rule (H):
In this case we get the inequality $\sigma_{\alpha_i} \trianglelefteq^1 \sigma_{\beta_j}$ in the left premise of this (H), which, by property (1) proved above, means that $\alpha_i = \beta_j$. Subsequent applications of rule (H) to other $\alpha_{i'}$ shows a correspondence between all α_i and some β_j. Thus we obtain $\{\alpha_1, \ldots, \alpha_n\} \subseteq \{\beta_1, \ldots, \beta_m\}$.

5 The reasoning in this case is essentially the same as for case (2). □

Theorem 5.5. NP-completeness The problem of affine retractions is NP-complete:

1 for the class of types of order 4 having infinitely many type atoms;
2 for the class of types of order 5 having k type atoms where $k \geqslant 2$;
3 for the class of types of order 7 having a single type atom.

Proof. It follows from Lemmas 5.2 and 5.3 that the existence of a properly isolated family implies NP-completeness. From Lemma 5.4 we know that a properly isolated family does indeed exist in each case. As the NP-hardness construction uses types with an order greater by 3 than the order of the types in the properly isolated family, the types in the family $\mathscr{A}^\infty$ imply order 4, the types in the family $\mathscr{A}^2$ imply order 5, and the types in the family $\mathscr{A}^1$ imply order 7. □

6. Conclusions

The results in this section close the algorithmic gap for type languages with infinitely many type atoms. Formally, this is the kind of approach with which current applied programming languages operate. Moreover, most of the currently used higher-order functions have order 3.

One may want to restrict types so that they have at most k type atoms for some reasonably high fixed number, such as 10. The problem of affine retractability is already NP-complete for types with 2 atoms having order 5. The author also conjectures that this problem is NP-hard for types of order 4.

Acknowledgements

I would like to thank Paweł Urzyczyn for pointing out the problem and for discussions on it. I would also like to thank Damian Niwiński and the anonymous referees for help in the preparation of this paper.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

References

Barendregt, H. P. (1992) Lambda calculi with types. In: Gabbay, S. A. D. and Maibaum, T. (eds.) *Handbook of Logic in Computer Science* **2**, Oxford Science Publications 117–309.

Bono, V. and Tiuryn, J. (2002) Products and polymorphic subtypes. *Fundamenta Informaticae* **51** (1) 13–41.

Bruce, K. and Longo, G. (1985) Provable isomorphisms and domain equations in models of typed languages. In: *ACM Symposium on Theory of Computing (STOC'85)*.

Considine, J. (2000) Deciding isomorphisms of simple types in polynomial time. Technical report, Boston University, Boston, MA, USA.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990) *Introduction to Algorithms*, MIT Press.

de'Liguoro, U., Piperno, A. and Statman, R. (1992) Retracts in Simply Typed $\lambda\beta\eta$-Calculus. In: *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science* 461–469.

Dezani-Ciancaglini, M. (1976) Characterization of normal forms possessing inverse in the lambda-beta-mu-calculus. *Theoretical Computer Science* **2** (3) 323–337.

Di Cosmo, R. (1995) *Isomorphisms of types: from lambda-calculus to information retrieval and language design*, Birkhauser.

Dowek, G. (1993) A unification algorithm for second-order linear terms. Manuscript available at `http://coq.inria.fr/dowek`.

Fiore, M. P., Cosmo, R. D. and Balat, V. (2002) Remarks on isomorphisms in typed lambda calculi with empty and sum types. In: *LICS '02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society 147.

Levy, J., Schmidt-Schauß, M. and Villaret, M. (2006) Bounded second-order unification is NP-complete. In: Term Rewriting and Applications (RTA-17). *Springer-Verlag Lecture Notes in Computer Science* **4098** 400–414.

Lincoln, P. and Mitchell, J. C. (1992) Algorithmic aspects of type inference with subtypes. In: *POPL '92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM Press 293–304.

McAdam, B. J. (2002) *Repairing Type Errors in Functional Programs*, Ph.D. thesis, University of Edinburgh. (Available at `http://hdl.handle.net/1842/368.`)

Mitchell, J. C. (1988) Polymorphic type inference and containment. *Information and Computation* **76** (2-3) 211–249.

Padovani, V. (2001) Retracts in simple types. In: Proceedings of Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001. *Springer-Verlag Lecture Notes in Computer Science* **2044** 376–384.

Palsberg, J. and Zhao, T. (2001) Efficient and flexible matching of recursive types. *Information and Computation* **171** (2) 364–387.

Regnier, L. and Urzyczyn, P. (2001) Retractions of types with many atoms. Technical report, Institute of Informatics, Warsaw University. (Available at `http://arxiv.org/abs/cs.LO/0212005`.)

Rittri, M. (1991) Using types as search keys in function libraries. *Journal of Functional Programming* **1** (1) 71–89.

Schmidt-Schauß, M. (2004) Decidability of bounded second order unification. *Information and Computation* **188** (2) 143–178.

Schubert, A. (2004) On the building of affine retractions. In: Marcinkowski, J. and Tarlecki, A. (eds.) Proceedings of Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL. *Springer-Verlag Lecture Notes in Computer Science* **3210** 205–219.

Spławski, Z. and Urzyczyn, P. (1999) Type fixpoints: Iteration vs. recursion. In: *Proceedings of 4th International Conference on Functional Programming*, ACM Press 102–113.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press

Stirling, C. (2006) A game-theoretic approach to deciding higher-order matching. In: Bugliesi, M., Preneel, B., Sassone, V. and Wegener, I. (eds.) Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Proceedings, Part II. *Springer-Verlag Lecture Notes in Computer Science* **4052** 348–359.

Vorobyov, S. (1997) The "Hardest" Natural Decidable Theory. In: *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science* 294–305.

Wierzbicki, T. and Dougherty, D. (2002) A Decidable Variant of Higher Order Matching. In: Tison, S. (ed.) Proc. 13th Conf. on Rewriting Techniques and Applications, RTA'02. *Springer-Verlag Lecture Notes in Computer Science* **2378** 340–351.

Zibin, Y., Gil, J. Y. and Considine, J. (2003) Efficient algorithms for isomorphisms of simple types. In: *POPL '03: Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press 160–171.

https://doi.org/10.1017/S096012950800683X Published online by Cambridge University Press