On the Building of Affine Retractions

Aleksy Schubert[*]

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

Abstract. 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 paper presents a system which for given σ, τ derives affine retractability i.e. the above relation with additional restriction that in C and D every bound variable occurs at most once. A derivation in the system constructs these terms. What is more, the complexity of building affine retractions is studied. The problem of affine retractability is NP-complete even for the class of types over single type atom and having limited functional order. A polynomial algorithm for types of orders less than 3 is also 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 exists 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 1980s, see e.g. [BL85]. A complete and effective characterisation of the relation is given in [Cos95]. Moreover, the relation of type isomorphism has already been successfully used in tools supporting searching in software libraries e.g. [Cos95, Rit91].

The notion of retraction is a generalisation of isomorphism. In this case only $D \circ C =_{\beta\eta} \lambda x.x$ is required in place of two abovementioned equalities. 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.

The knowledge concerning type retractions is undeveloped even for very simple formalisms, like the simply typed lambda calculus. There is a complete and effective criterion of retractability in the calculus, but with respect to the β-reduction [SU99]. There are also several sufficient conditions for $\beta\eta$-reduction based retractions in [dPS92]. The last paper includes a complete characterisation of *affine retractions* in the lambda calculus with a single type atom

[*] This work was partly supported by KBN grant no 7 T11C 028 20, part of this work was conducted while the author was a winner of the fellowship of The Foundation for Polish Science (FNP).

© Springer-Verlag Berlin Heidelberg 2004

where the word *affine* means that in terms C, D each bound variable is used at most once. A paper by Padovani [Pad01] contains an algorithm that decides full $\beta\eta$ retractability in the simply typed lambda calculus with types that have a single type atom. In a paper by Regnier and Urzyczyn [LR01], the authors give characterisations of $\beta\eta$ retractability in the lambda calculus with many type atoms, but these characterisations are either not effective or not complete. However, they give an effective and complete characterisation of affine retractability.

Type retractions (but in a richer type system) have already been used to prove that recursive polymorphic types cannot be encoded in the polymorphic lambda calculus $\lambda 2$ [SU99]. Type retractions may, in addition, turn out to be useful as a method to find more general operations in software libraries — similarly to the foregoing applications of type isomorphisms. By 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 to generate 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.

There are two reasons that justify the restriction to affine retractions. First, the type retraction problem can be defined in terms of the higher-order matching problem. This can be done using the following equation $\lambda x.X(Yx) = \lambda x.x$ where X, Y are unknown variables. The higher-order matching problem is known to be at least non-elementarily hard [Vor97], if not undecidable. Moreover, the construction by Padovani in [Pad01], which employs a special — known to be non-elementary — case of the higher-order matching problem suggests that the problem of finding retractions is highly intractable. Second, dealing with affine retractions is supported by the assumption that every transformation approved by a human must be fairly simple so that it can be understood. This kind of situation can occur in the afore-mentioned search in software libraries.

This paper contains a proof that the relation of affine retractability is polynomial for types of order at most 3. This provides a tight bound on the tractability of the problem as the problem is NP-complete already for types of order 4. The latter result holds for the class of types with unbounded number of atoms. If the number of type atoms is bounded by a number $k > 1$ the problem is NP-complete for types of order 5. If there is only one type atom the problem is NP-complete for types of order 7.

2 Basic Definitions

We assume that the reader is familiar with the simply typed lambda calculus. Thus we only sketch the basic definitions in order rather to settle the notation

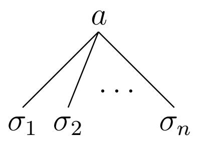

Fig. 1. A schematic picture of $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to a$

than thoroughly introduce the notions. A more gentle introduction can be found e.g. in [Bar92].

The simply typed lambda calculus $\lambda_\to$ (in the Church fashion) has terms of the form MN or $\lambda x : \sigma.M$. The λ operator binds the variable x in the body of M so we deal with the notions of free and bound variables together with the α equivalence relation. We identify α-quivalent expressions. The types in $\lambda_\to$ are built using basic types from a nonempty set $\mathcal{B}$. The types are constructed by means of the $\to$ operator e.g. $\sigma \to \tau$. The types are assigned to terms by means of a type inference system. Rules of such a system can be found in [Bar92]. We sometimes use a notation $\Delta \to a$, where Δ is a set or a sequence of types $\{\delta_1, \dots, \delta_n\}$, to denote $\delta_1 \to \cdots \to \delta_n \to a$. The types are categorised by an order. The order is defined as $\mathrm{ord}(a) = 1$ for $a \in \mathcal{B}$ and $\mathrm{ord}(\sigma \to \tau) = \max(\mathrm{ord}(\sigma) + 1, \mathrm{ord}(\tau))$. The set of types having the order k is denoted $\mathrm{T}^k_\to$. If $\sigma = \sigma_1 \to \cdots \to \sigma_n \to a$ or $\sigma = a$ then $head(\sigma) = a$. All the types $\sigma_1, \dots, \sigma_n$ are called *arguments* of σ.

We also adopt a method of representing types on pictures which is more convenient in dealing with type retractions. The idea of the representation is presented on Fig. 1.

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 with the η-reduction reduction generated by $(\lambda x : \sigma.Mx) \to_\eta M$ where x does not occur freely in M. The combination of the two relations is denoted by $\to_{\beta\eta}$. We use also the reflexive transitive closure $\to^*_{\beta\eta}$ of $\to_{\beta\eta}$ as well as the least equivalence $=_{\beta\eta}$ containing $\to_{\beta\eta}$.

Definition 1. `(retracts)`
We say that a type σ is a *retract* of a type τ, and we write $\sigma \trianglelefteq_{\beta\eta} \tau$, iff there exists a pair of terms $C : \sigma \to \tau$ and $D : \tau \to \sigma$ such that $D \circ C =_{\beta\eta} \mathbf{I}_\sigma = \lambda x : \sigma.x$. We say that a type σ is an *affine retract* of a type τ, and we write $\sigma \trianglelefteq^1 \tau$ iff C, D are affine terms (i.e. terms with at most one occurrence of each bound variable). We usually omit subscript $\beta\eta$ and write $\trianglelefteq$ or $\trianglelefteq^1$.

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, D such that $C : \sigma \to \tau$ and $D : \tau \to \sigma$ such that $D \circ C =_{\beta\eta} \mathbf{I}_\sigma$?

(Ax) $$a \trianglelefteq^1 a$$

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

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

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

Fig. 2. The syntax directed system

2.1 A System for Inference of Retraction Terms

We consider a system to infer inequalities $\trianglelefteq^1$. The system was proposed by Regnier and Urzyczyn (RU) and is presented at Fig. 2. The most important property of the system is the lack of any cut-like rule.

The system RU is a good starting point in design of our term system for deducing affine retractions. We modify the system RU since its original form has several notational conveniences which are inadequate in the context of generation of retracting terms. The term system presented on Fig. 3 infers sequents $\vdash C \bullet D : \sigma \trianglelefteq^1 \tau$ such that C, D correspond to terms certifying the retraction $\sigma \trianglelefteq^1 \tau$.

(Ax) $$\vdash \mathbf{I}_a \bullet \mathbf{I}_a : a \trianglelefteq^1 a$$

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

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

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

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

Fig. 3. The term system for linear retractions

The rule (D) in the system RU and in the term system are a little bit difficult to grasp. Figure 4 may help in understanding of them.

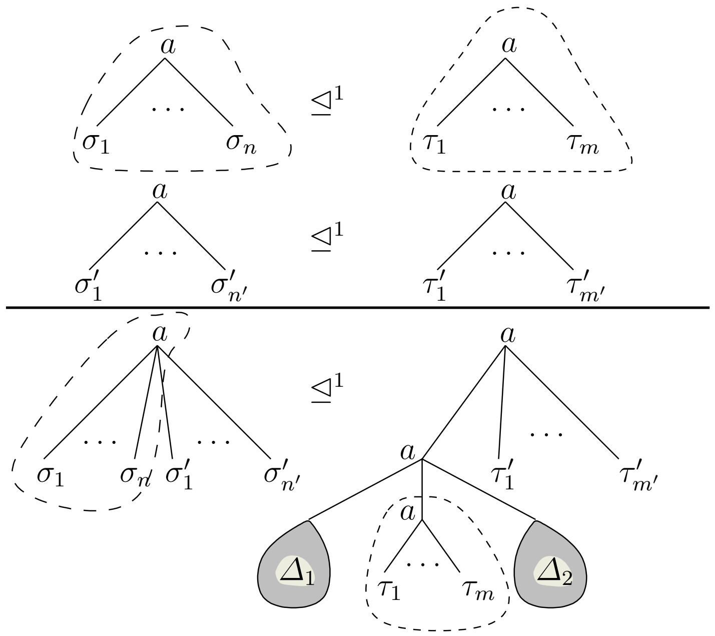

Fig. 4. The rule (D) from RU with emphasised deep inequality

Definition 3. `(terms from the term system)`
We define a transformation $\texttt{lambdify}(\cdot)$ for the retraction combinators used in Fig. 3

- $\texttt{lambdify}(\mathbf{I}_\sigma) = \lambda x : \sigma . x$,
- $\texttt{lambdify}(H_{\sigma,\tau,\sigma',\tau'}) =$
  $$\lambda T_1 : \tau \to \tau' . \lambda T_2 : \sigma' \to \sigma . \lambda f : \sigma \to \tau . \lambda x : \sigma' . T_1(f(T_2 x))$$
- $\texttt{lambdify}(N_{\sigma,\tau,\Sigma}) = \lambda T : \sigma \to \tau . \lambda f : \sigma . \lambda x_1 : \sigma_1 \ldots x_n : \sigma_n . T f$
  where $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$
- $\texttt{lambdify}(N'_{\sigma,\tau,\Sigma}) = \lambda T : \tau \to \sigma . \lambda f : \Sigma \to \tau . T(f z_1 \cdots z_n)$
  where $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and $z_i : \sigma_i$ for $i = 1, \ldots, n$.
- $\texttt{lambdify}(D) =$
  $$\lambda T_1 : (\Sigma \to a) \to \tau' . \lambda T_2 : \sigma \to \tau .$$
  $$\lambda f : \Sigma \to \sigma . \lambda y : \Delta_1 \to \tau' \to \Delta_2 \to a .$$
  $$\lambda z_1 : \tau_1 \ldots z_p : \tau_p .\ y c_1^{\delta_1} \cdots c_{q_1}^{\delta_{q_1}} \overline{D} c_{q_1+1}^{\delta_{q_1+1}} \cdots c_{q_2}^{\delta_{q_2}}$$
- $\texttt{lambdify}(D') =$
  $$\lambda T'_1 : \tau' \to \Sigma \to a . \lambda T'_2 : \tau \to \sigma .$$
  $$\lambda g : (\Delta_1 \to \tau' \to \Delta_2 \to a) \to \tau . \lambda x_1 : \rho_1, \ldots, x_r : \rho_r .\ T'_2 \overline{D'}$$

where

$$\begin{aligned}
\tau\ \ &= \tau_1 \to \cdots \to \tau_p \to a,\\
\Delta_1 &= \{\delta_1, \ldots, \delta_{q_1}\}, \quad \Delta_2 = \{\delta_{q_1+1}, \ldots, \delta_{q_2}\}, \quad \Sigma = \{\rho_1, \ldots, \rho_r\}\\
\overline{D}\ \ &= T_1 \lambda y_1 : \rho_1 \ldots y_r : \rho_r . T_2 (f y_1 \cdots y_r) z_1 \cdots z_p,\\
\overline{D'} &= g \lambda z_1 : \delta_1 \ldots z_{q_1} : \delta_{q_1} . \lambda y : \tau' . \lambda z_{q_1+1} : \delta_1 \ldots z_{q_2} : \delta_{q_2} . T_1' y x_1 \cdots x_r
\end{aligned}$$

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

– `lambdify`$(P^{\pi_1}_{\pi_2}) =$

$$\begin{aligned}
&\lambda f : \sigma \to \tau.\\
&\quad \lambda x : \sigma_{\pi_1(1)} \to \cdots \to \sigma_{\pi_1(n)} \to a.\\
&\quad\quad \lambda y_{\pi_2(1)} : \tau_{\pi_2(1)} \ldots y_{\pi_2(m)} : \tau_{\pi_2(m)}.\\
&\quad\quad\quad f(\lambda v_1 : \sigma_1 \ldots v_n : \sigma_n . x v_{\pi_1(1)} \cdots v_{\pi_1(n)}) y_1 \cdots y_m
\end{aligned}$$

where π_1, π_2 are permutations of the sets $\{1, \ldots, n\}$ and $\{1, \ldots, m\}$ respectively whereas $\sigma = \sigma_1 \to \cdots \to \sigma_n \to a$ and $\tau = \tau_1 \to \cdots \to \tau_m \to a$.

We omit type parametrisation of D, D' and $P^{\pi_1}_{\pi_2}$ as suitable notation is overwhelming, but we implicitly assume that these constants are annotated as follows

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

Proposition 4. `(soundness)`
Let

$$\frac{\vdash T_1 \bullet T_1' : \sigma_1 \trianglelefteq^1 \tau_1, \cdots, \vdash T_n \bullet T_n' : \sigma_n \trianglelefteq^1 \tau_n}{\vdash T \bullet T' : \sigma \trianglelefteq^1 \tau}$$

be a rule of the system on Fig. 3. If for all premises $\vdash T_i \bullet T_i' : \sigma_i \trianglelefteq^1 \tau_i$ *the pair* `lambdify`(T_i), `lambdify`(T_i') *certifies that* σ_i *is a retract of* τ_i *then the pair* `lambdify`(T), `lambdify`(T') *certifies that* σ *is a retract of* τ.

Proof. The proof is by cases according to the rules of the system from Fig. 3. □

Theorem 5. `(completeness)`
If there is a pair of terms T, T' *which certifies that* σ *is a retract of* τ *then there is a pair of terms* $\hat{T}, \hat{T}'$ *over the signature defined in Definition 3 such that* `lambdify`$(\hat{T})$, `lambdify`$(\hat{T}')$ *certifies that* σ *is a retract of* τ.

Proof. The proof is by induction with respect to the size of $\sigma \trianglelefteq^1 \tau$. □

2.2 Permutations of a Derivation

The term system allows us to perform several changes in particular form of a derivation. The possibilities are summarised in the following Prop. 6.

In order to make the formulation of the proposition easier we introduce a few conventions for referring to nodes in a derivation. For each rule on Fig. 3 the sequents above the rule line are called *premises* of the rule while the sequent below the line is called a *conclusion.* In the rule (H) the first sequent above the rule line is called a *left premise* of the rule whereas the second one is called a *right premise.* In the rule (D) the first sequent above the rule line is called the left premise and the second one is called the right premise.

Proposition 6. `(permutations inside derivations)` *Let D be a derivation for a retraction $\sigma \trianglelefteq^1 \tau$.*

1. *If D starts with two subsequent applications of the rule (P) followed by a derivation D' then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ which starts with a single application of the rule (P) followed by the derivation D'.*
2. *If D starts with two subsequent applications of the rule (N) followed by a derivation D' then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ which starts with a single application of the rule (N) followed by the derivation D'.*
3. *Let D start with an application of the rule (N) followed by a derivation D'. If D' starts with an application of the rule (P) with a derivation D'' applied to its premise then there is a derivation $\hat{D}$ for $\sigma \trianglelefteq^1 \tau$ which starts with an application of the rule (P) followed by an application of the rule (N) with D'' applied to its premise.*
4. *Let D start with an application of the rule (N) followed by a derivation D'. If D' starts with an application of one of the rules (H) or (D) with a derivation D_L applied to its left premise and D_R applied to its right premise then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ which starts with an application of the rule (H) or (D) respectively with D_L applied to its left premise and D'_R applied to its right premise where D'_R starts with the rule (N) followed by the derivation D_R.*
5. *Let D start with an application of the rule (H) or (D) with derivations D_L and D_R applied to the left and to the right premise of the rule respectively. If D_R starts with an application of the rule (P) to the right premise of the rule which is further followed by a derivation D'_R then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ which starts with an application of the rule (P) followed by an application of (H) or (D) respectively followed by D_L applied to the left premise and D'_R to the right premise.*
6. *Let D start with an application of the rule $(H)^1$ or $(D)^1$ with derivations D_L and D_R applied to the left and to the right premise of the rule. If D_R starts with an application of the rule $(H)^2$ or $(D)^2$ with derivations D_{RL} and D_{RR} applied to the left and to the right premise of the rule then there is a derivation D'' for $\sigma \trianglelefteq^1 \tau$ which starts with an application of the rule (P) followed by an application of the rule $(H)^2$ or $(D)^2$ respectively with the derivation D_{RL} applied to its left premise and derivation D'_R applied to its right premise where D'_R starts with an application of the rule $(H)^1$ or $(D)^1$ respectively followed by the derivation D_L applied to its left premise and D_{RR} to its right premise.*

Proof. We leave a routine case analysis for the reader. □

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

3 NP-Completeness

Theorem 7. `(affine retractions are in NP)`
There is a nondeterministic 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 [WD02].

A more direct argument could rely on the observation that in the system on Fig. 2 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 according to 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. □

3.1 NP-Hardness

We present here NP-hardness proofs for three languages of types. In the first one we deal with unbounded number of type atoms. In the second one the number of atoms is bounded by a number $k > 1$. In the third one there is only one type atom. In order to reuse a part of the NP-hardness construction we have to define a few notions which describe requirements on the language of types which enables the NP-hardness construction.

We deal with families of sets of types of the form $\mathcal{A} = \{A_i\}_{i\in\mathbb{N}}$ where $|A_i| = i$. Such families are called here for short *graded families*. We define $\sigma' = \sigma \to a$ for $\sigma \in \bigcup_{i\in\mathbb{N}} A_i$.

Definition 8. `(properly isolated family)`
We say that a graded family $\{A_i\}_{i\in\mathbb{N}}$ is *properly isolated* if for each i and for each $\sigma_\alpha, \sigma_{\alpha_1}, \ldots, \sigma_{\alpha_n}, \sigma_{\beta_1}, \ldots, \sigma_{\beta_m} \in A_i$ we have

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

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 certifying 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

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

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 $\mathrm{FV}(\phi) = \{x_1, \dots, x_m\}$ be the set of all variables occurring in ϕ and let $Y_0 = \{x_1, \dots, x_m, \neg x_1, \dots, \neg x_m\}$. 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 according to the usual semantics of the connectives $\wedge, \vee$, and $\neg$.

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

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

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

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 $B_y = \{\phi_{i_1}, \dots, \phi_{i_k}\}$. This allows us to define $\tau_{B_x} = \overline{\tau}_{B_x} \to \overline{\tau}_{B_{\neg x}} \to a$. Moreover, we define for $x \in \mathrm{FV}(\phi)$ a type $\tau_x = \sigma'_x \to \sigma'_{\neg x} \to a$.

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

$$\begin{aligned} \sigma_\phi = \sigma_{x_1} \to \cdots \to \sigma_{x_m} \to \qquad & \tau_\phi = \tau_{x_1} \to \cdots \to \tau_{x_m} \to \\ \sigma_{\neg x_1} \to \cdots \to \sigma_{\neg x_m} \to \qquad & \tau_{B_{x_1}} \to \cdots \to \tau_{B_{x_m}} \to a \\ \sigma_{\phi_1} \to \cdots \to \sigma_{\phi_n} \to a \qquad & \end{aligned} \tag{2}$$

Lemma 9. `(a valuation induces a derivation)`
Let $v : X \to \{0, 1\}$ be a valuation of propositional variables. If $v(\phi) = 1$ then there is a derivation for $\sigma_\phi \trianglelefteq^1 \tau_\phi$.

Proof. In order to prove the lemma we have to generalize the lemma so that it holds for wider range of τ_ϕ. We say that types σ_ϕ, τ'_ϕ are an acceptable coding of ϕ if σ_ϕ is as before and τ'_ϕ has the form $\tau'_\phi = \tau_{x_1} \to \cdots \to \tau_{x_m} \to \tau'_{B_{x_1}} \to \cdots \to \tau'_{B_{x_m}} \to a$ where $\tau'_{B_y} = \overline{\tau}'_{B_x} \to \overline{\tau}'_{B_{\neg x}} \to a$ and $\overline{\tau}'_{B_y} = \sigma_{\neg y} \to \sigma_{\phi_{i_1}} \to \cdots \to \sigma_{\phi_{i_k}} \to \rho_1 \to \cdots \rho_l \to a$ for $y \in Y_0$ with $\rho_1, \dots, \rho_l$ being arbitrary types and $B_y = \{\phi_{i_1}, \dots, \phi_{i_k}\}$. Moreover, we say that a coding is acceptable if in $\overline{\tau}'_{B_y}$ subtypes ρ_i are permuted with the types σ_α.

We produce a derivation for $\sigma_\phi \trianglelefteq^1 \tau'_\phi$ where σ_ϕ, τ'_ϕ are an acceptable coding of ϕ by induction on the number of variables in ϕ. Note that σ_ϕ, τ_ϕ are an acceptable coding for ϕ. First, we may permute $\sigma_\phi \trianglelefteq^1 \tau'_\phi$ using (P) so that we obtain

$$\begin{aligned} \sigma^1_\phi = \sigma_{x_1} \to \sigma^2_\phi, \quad & \tau'^1_\phi = \tau_{x_1} \to \tau'^2_\phi \text{ in case } v(x_1) = 1 \\ \sigma^1_\phi = \sigma_{\neg x_1} \to \sigma^2_\phi, \quad & \tau'^1_\phi = \tau_{x_1} \to \tau'^2_\phi \text{ in case } v(x_1) = 0 \end{aligned} \tag{3}$$

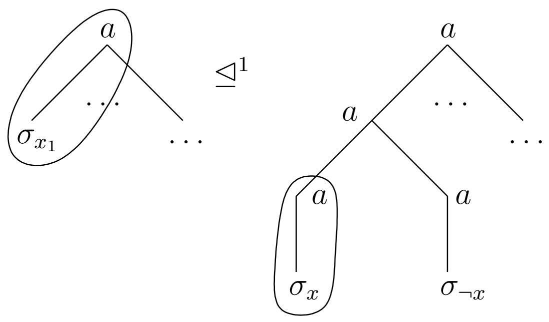

Fig. 5. The application of the rule (D) in order to remove the first variable in (3)

We apply the rule $(\mathrm{D})^1$ and obtain the left premise $\sigma'_{x_1} \trianglelefteq^1 \sigma'_{x_1}$ (or $\sigma'_{\neg x_1} \trianglelefteq^1 \sigma'_{\neg x_1}$) — see Fig. 5. This derivation can be done by the rule (H).

In order to construct a derivation for the right premise of $(\mathrm{D})^1$ we permute $\sigma^2_\phi \trianglelefteq^1 \tau'^2_\phi$ so that we obtain

$$\sigma_{\neg y} \to \sigma_{\phi_{i_1}} \to \cdots \to \sigma_{\phi_{i_r}} \to \sigma^3_\phi \trianglelefteq^1 \tau'_{B_{x_1}} \to \tau'^3_\phi \tag{4}$$

where y is either x_1 or $\neg x_1$ and $\{\phi_{i_1}, \ldots, \phi_{i_r}\} = B_y$. We apply the rule $(\mathrm{D})^2$ as on Fig. 6. The left premise of the rule can be derived by repeated application of the rule (H) to get rid of σ_α followed by an application of (N) to get rid of ρ_α.

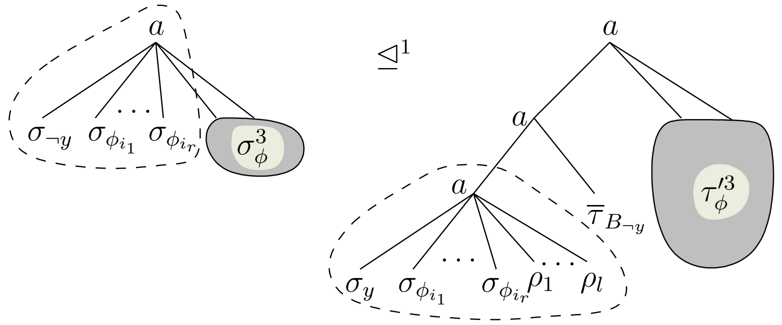

Fig. 6. The application of the rule (D) in order to decompose inequality (4)

The types $\sigma^3_\phi, \tau'^3_\phi$ are an acceptable coding of a formula ϕ' which is constructed from ϕ by erasing all the subformulae from B_y. Thus we obtain by induction a derivation for $\sigma^3_\phi \trianglelefteq^1 \tau'^3_\phi$. This derivation forms the lacking derivation for the right premise of the rule $(\mathrm{D})^2$. □

Lemma 10. (a derivation induces a valuation)
If $\sigma_\phi \trianglelefteq^1 \tau_\phi$ has a derivation then there exists a valuation $v : X \to \{0,1\}$ such that $v(\phi) = 1$.

Proof. In order to prove the lemma we have to generalize 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 $v_0 : X \rightharpoondown \{0,1\}$ types σ_ϕ, τ_ϕ are an acceptable coding for ϕ, v if

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

where $\{x_1, \ldots, x_m\} \subseteq \mathrm{FV}(\phi)\backslash\mathrm{Dom}(v)$. 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_0. If $\sigma_\phi \trianglelefteq^1 \tau_\phi$ has a derivation then there exists a valuation $v : X \to \{0,1\}$ which extends v_0 and for which $v(\phi) = 1$.

The proof is by induction on the number of variables in $\mathrm{FV}(\phi)\backslash\mathrm{Dom}(v_0)$. If there is a derivation for $\sigma_\phi \trianglelefteq^1 \tau_\phi$ then by Prop. 6 we may assume that it starts with the rule (P) followed by a certain number of the rules (H) and (D) applied to the subsequent right premises with a single (N) rule at the end followed by the axiom. Moreover, we assume that no rule (D) can be replaced by an equivalent rule (N). Derivations having this shape are called *well-formed derivations.*

For such derivations the first (P) rule may be followed by

1. the rule $(\mathrm{H})^1$ in which the left premise is $\sigma_{x_1} \trianglelefteq^1 \tau_\alpha$ for some α, or
2. the rule $(\mathrm{D})^1$ in which the left premise is

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

   for certain $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m$, or
3. the rule $(\mathrm{N})^1$.

In case (1) we have two options either (a) $\tau_\alpha = \tau_{x_1}$, or (b) $\tau_\alpha = \tau_{B_{x_1}}$. This is guaranteed by the condition (5) on the properly isolated family.

In case (1.a) we have $\tau_\alpha = \tau_{x_1}$. We extend v_0 to v_0' so that $v_0'(x_1) = 1$. In this situation we may assume that the right premise of the rule $(\mathrm{H})^1$ has a derivation which starts with the rule (P) followed by

i. the rule $(\mathrm{H})^2$ in which the left premise is $\sigma_{\neg x_1} \trianglelefteq^1 \tau_{\alpha'}$ or
ii. the rule $(\mathrm{D})^2$ in which the left premise is

$$\sigma_{\neg x_1} \to \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \tau_{\beta_1} \to \cdots \to \tau_{\beta_m} \to a \ \text{ or} \qquad (5)$$

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

For the situation (1.a.i) we observe that this is possible only if $\alpha' = x_j$ or $\alpha' = B_{x_j}$. The condition (2) on the properly isolated family implies that in the former case $\alpha' = x_1$. This is impossible, though, as τ_{x_1} is already used in the left premise of $(\mathrm{H})^1$. The condition (5) on the properly isolated family implies that in the latter case $B_{x_j} = B_{x_1}$. As we got rid of both σ_{x_1} and $\sigma_{\neg x_1}$ the right premise of $(\mathrm{H})^2$ is a formula which is an acceptable coding of ϕ and v_0' so by induction hypothesis v_0' extends into a valuation v such that $v(\phi) = 1$.

For the proof in the case (1.a.ii), we observe that the part (4) of Definition 8 implies that $\neg x_1 \in \{\beta_1, \ldots, \beta_m\}$. This is possible by an application the rule $(\mathrm{D})^2$ either to τ_{x_1} or to $\tau_{B_{x_1}}$. The former option is not possible since τ_{x_1} is already used in $(\mathrm{H})^1$. The latter option implies that the inequality (5) 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 $\{\phi_{i_1}, \ldots, \phi_{i_r}\} = B_{x_1}$. The 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 $(\mathrm{D})^2$ — possibly after an application of the rule (P) — has the form

$$\begin{array}{lll} \sigma_{x_2} \to \cdots \to \sigma_{x_m} \to & \trianglelefteq^1 & \tau_{x_2} \to \cdots \to \tau_{x_m} \to \\ \sigma_{\neg x_2} \to \cdots \to \sigma_{\neg x_m} \to & & \tau_{B_{x_2}} \to \cdots \to \tau_{B_{x_m}} \to a \\ \sigma_{\phi_{j_1}} \to \cdots \to \sigma_{\phi_{j_l}} \to a & & \end{array}$$

The variables $x_2, \ldots, x_m$ belong to $\mathrm{FV}(\phi)\backslash\mathrm{Dom}(v_0')$ so this pair of types is an acceptable coding of ϕ. As this premise has its derivation, the induction hypothesis implies that v_0' can be extended to a valuation v such that $v(\phi) = 1$.

For the proof in the case (1.a.iii) 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 though, as each σ_x on the left hand side of the original inequality is accompanied by $\sigma_{\neg x}$.

In case (1.b) we have $\tau_\alpha = \tau_{B_{x_1}}$. We extend v_0 to v_0' so that $v_0'(x_1) = 0$. In this situation the right premise of the rule $(\mathrm{H})^1$ has a derivation which starts with the rule (P) followed by

i. the rule $(\mathrm{H})^2$ in which the left premise is $\sigma_{\neg x_1} \trianglelefteq^1 \tau_{\alpha'}$ or
ii. the rule $(\mathrm{D})^2$ in which the left premise is

$$\sigma_{\neg x_1} \to \sigma_{\alpha_1} \to \cdots \to \sigma_{\alpha_n} \to a \trianglelefteq^1 \tau_{\beta_1} \to \cdots \to \tau_{\beta_m} \to a \quad \text{or}$$

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

The proof is similar to the one for the case (1.a), but we exchange the role of τ_{x_1} and $\tau_{B_{x_1}}$.

In case (2) the proof runs similarly as in the case (1.a.ii), but we have to put $v_0'(x_1) = 0$ and before stepping into the induction hypothesis we have to analyse the way $\sigma_{\neg x_1}$ is handled. This is done by an analysis similar to the one in the remaining parts of the case (1).

In case (3) 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 though, as each σ_x on the left hand side of the original inequality is accompanied by $\sigma_{\neg x}$. □

3.2 Properly Isolated Families

Definition 11. `(families of types)`
We now define three particular families of types.

– A family of types $\mathcal{A}^\infty = \{A_i^\infty\}_{i\in\mathbb{N}}$ 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 Sect. 3.1.
– A family of types $\mathcal{A}^2 = \{A_i^2\}_{i\in\mathbb{N}}$ 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 Sect. 3.1.
– A family of types $\mathcal{A}^1 = \{A_i^1\}_{i\in\mathbb{N}}$ is such that $A_n^1 = \{\sigma_1, \ldots, \sigma_n\}$ 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 $\tilde{\sigma}_i = \underbrace{a \to \cdots \to a}_{i \text{ times}} \to a$.

Lemma 12. `(isolation)`
(A) The family $\mathcal{A}^\infty$ is a properly isolated family for the set of types built of infinitely many atoms. (B) The family $\mathcal{A}^2$ is a properly isolated family for the set of types built of k atomic types where $k \geq 2$. (C) The family $\mathcal{A}^1$ is a properly isolated family for the set of types built of a single atomic type.

Proof. The proof is by a routine case analysis. □

Theorem 13. `(NP-completeness)`
The problem of affine retractions is NP-complete

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

Proof. It follows from Lemma 9 and 10 that the existence of a properly isolated family implies NP-completeness. From Lemma 12 we know that a properly isolated family does indeed exist in each case. □

4 Polynomial Case

Here 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 single one. This is because we want to analyse separately the running time of the algorithms.

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

Definition 14. `(algorithm for the order 1)`
Let $\sigma = a$ and $\tau = b$ be the input types where a, b are atomic. If $a = b$ then the affine retractability holds else it does not hold.

The algorithms for orders 2 and 3 are based on the following observations.

Proposition 15. `(the 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 which does not involve the rule (D).

Proof. As the right-hand side has the order at most 3 the rule (D) may give only a term of the order 1 on the right-hand side. Thus we may have the left-hand side of order 1. However, in such an application of (D) the right premise can also be obtained by the rule (N). □

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

Definition 16. `(weight of types)`
For types of order 2 we introduce a weight function $\# : \mathrm{T}^2_{\rightarrow} \rightarrow \mathbb{N}^{\mathcal{B}}$ defined as

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

We use the order $\leqslant$ on elements of $\mathbb{N}^{\mathcal{B}}$ defined as — $f \leqslant g$ iff for each $a \in \mathcal{B}$ we have $f(a) \leq g(a)$.

Proposition 17. `(the weight and` $\trianglelefteq^1$`)`
If $\sigma, \tau \in \mathrm{T}^2_{\rightarrow}$ and $head(\sigma) = head(\tau)$ then $\sigma \trianglelefteq^1 \tau$ iff $\#(\sigma) \leqslant \#(\tau)$.

Proof. Induction on the size of σ. □

Definition 18. `(algorithm for the order 2)`
Let σ and τ be the input types. If $head(\sigma) \neq head(\tau)$ then the affine retractability does not hold. Otherwise, if $\#(\sigma) \leqslant \#(\tau)$ then $\sigma \trianglelefteq^1 \tau$.

Note that the above algorithm runs in time linear to the size of the input.

The correctness of the below defined algorithm for the order 3 is based on Prop. 15 and the correctness of the algorithm for the order 2.

Definition 19. `(algorithm for the order 3)`
Let σ and τ be the input types. If $head(\sigma) \neq head(\tau)$ then the affine retractability does not hold. In case $head(\sigma) = head(\tau)$ we construct a bipartite graph $G = (V_1 \cup V_2, E)$ in which

$$\begin{aligned} V_1 &= \{\sigma_i \mid 1 \leq i \leq n \text{ and } \sigma = \sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow a\} \\ V_2 &= \{\tau_i \mid 1 \leq i \leq m \text{ and } \tau = \tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow a\} \\ E &= \{(\sigma_i, \tau_j) \mid \sigma_i \trianglelefteq^1 \tau_i, \sigma_i \in V_1, \tau_j \in V_2\} \end{aligned}$$

Now, we find a perfect matching in G. If there is one then $\sigma \trianglelefteq^1 \tau$ holds otherwise 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 perfect matching procedure is $O(m^3)$ where m is the size of an input graph (see e.g. Chap. 27 in [CLR90]). This makes the overall running time $O(n^6)$.

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

Although, one may want to restrict types so that they have at most k type atoms for some reasonably high fixed number e.g. 10. Unfortunately, the problem of affine retractability is NP-complete already for types with 2 atoms and having the order 5. The author concjectures, though, that this problem is polynomial for types of the order 4.

The algorithms presented here construct derivations for type retractions. The system on Fig. 3 can be used to compute retraction terms based on these derivations.

Acknowledgements. The author would like to thank Paweł Urzyczyn for pointing out the problem and for discussions on it. Thanks also go to Damian Niwiński for help in preparation of the paper.

References

[Bar92] Henk P. Barendregt, *Lambda calculi with types*, Handbook of Logic in Computer Science (S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, eds.), vol. 2, Oxford Science Publications, 1992, pp. 117–309.

[BL85] Kim Bruce and Giuseppe Longo, *Provable isomorphisms and domain equations in models of typed languages*, ACM Symposium on Theory of Computing (STOC'85), May 1985.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to algorithms*, ch. 27, MIT Press, 1990.

[Cos95] Roberto Di Cosmo, *Isomorphisms of types: from lambda-calculus to information retrieval and language design*, Birkhauser, 1995.

[dPS92] Ugo de'Liguoro, Adolfo Piperno, and Richard Statman, *Retracts in Simply Typed $\lambda\beta\eta$-Calculus*, Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science (Santa Cruz, CA), 1992, pp. 461–469.

[LR01] Paweł Urzyczyn Laurent Regnier, *Retractions of types with many atoms*, Tech. report, Institute of Informatics, Warsaw University, 2001.

[Pad01] Vincent Padovani, *Retracts in simple types*, Proc. of TLCA'2001, LNCS, no. 2044, Springer-Verlag, 2001, pp. 376–384.

[Rit91] Mikael Rittri, *Using types as search keys in function libraries*, Journal of Functional Programming **1** (1991), no. 1, 71–89.

[SU99] Zdzisław Spławski and Paweł Urzyczyn, *Type fixpoints: Iteration vs. recursion*, Proceedings of 4th International Conference on Functional Programming, ACM, 1999, pp. 102–113.

[Vor97] Sergei Vorobyov, *The "Hardest" Natural Decidable Theory*, Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science, 1997, pp. 294–305.

[WD02] Tomasz Wierzbicki and Dan Dougherty, *A Decidable Variant of Higher Order Matching*, Proc. 13th Conf. on Rewriting Techniques and Applications, RTA'02 (Sophie Tison, ed.), LNCS, no. 2378, Springer-Verlag, 2002, pp. 340–351.