Third-Order Matching in  
\( \lambda \rightarrow \text{-Curry} \) is Undecidable

Sergei Vorobyov

MP1-I-97-2-006 \hspace{1cm} May 1997
Author’s Address

Max-Planck Institut für Informatik, Im Stadtwald, D-66123, Saarbrücken, Germany

Publication Notes

May 28, 1997

Acknowledgements

Harald Ganzinger suggested important improvements to the statement of the problem, motivations, and presentation. Roger Hindley suggested the simple theories of type assignment, explained their classification and interrelation.
Abstract

Given closed untyped $\lambda$-terms $\lambda x_1 \ldots x_k.s$ and $t$, which can be assigned some types $\sigma_1 \rightarrow \ldots \rightarrow \sigma_k \rightarrow \tau$ and $\tau$ respectively in the Curry-style systems of type assignment (essentially due to R. Hindley) $\lambda \rightarrow$-Curry [Bar92], $\lambda \rightarrow$ [Mit96], $\lambda \rightarrow$ [Hin97], it is undecidable whether there exist closed terms $s_1, \ldots, s_k$ of types $\sigma_1, \ldots, \sigma_k$ such that $s[s_1/x_1, \ldots, s_k/x_k] = \beta n t$, even if the orders of $\sigma_i$’s do not exceed 3.

This undecidability result should be contrasted to the decidability of the third-order matching in the Church-style simply typed lambda calculus with a single constant base type [Dow94].

The proof is by reduction from the recursively inseparable sets of unsatisfiable and finitely satisfiable sentences of the first-order theory of binary relation [Tra53, Vau60].

Keywords

Typed lambda calculus, Higher-order matching problem, Decidability, Recursive inseparability, First-order theory of binary relation, Full type hierarchy
Contents

1 Introduction 2
2 Simple Theory of Type Assignment 4
3 Recursive Inseparability 6
4 Transformation Sequence 7
5 Language \(L_1\). Relativization: Translation of \(L_0\) into \(L_1\) 7
6 Reduction to \(\exists^*\forall^*\)-Sentences, Language \(L_2\) 9
7 Transformation to the Pure Prenex \(\exists^*\forall^*\)-Form 11
8 Getting Rid of Boolean Connectives and Predicates 11
9 Transformation to a Higher-Order Matching Problem 12
10 Faithfulness of the Reduction 13
1 Introduction

Unification, i.e., solving equations $e(\overline{x}) \approx e'(\overline{x})$ in different calculi is at the heart of the theorem-proving procedures. Matching (or half-unification) is a particular case of unification $e(\overline{x}) \approx c$, where only the left-hand side $e(\overline{x})$ contains instantiable variables. Matching is important for parameter passing, rewriting, and, more generally, everywhere where a transformation applicability is expressed by a pattern.

In the first-order languages both unification and matching are easily decidable [PW78]. Unification in the third-order languages was proved undecidable by G. Huet [Hue73]. Later W. Goldfarb proved undecidability of unification in the second-order languages [Gol81, Far91]. But Goldfarb’s result, unlike Huet’s, applies to second-order languages with constants, which turn out to be third-order after the canonical transformation to the constant-free form [Sta81] (pp. 330–331). Huet and Lang [HL78] proved decidability of the second-order pattern matching (with constants), which should be contrasted to Goldfarb’s undecidability result. Today decidability of matching is known for languages (with constants) of orders up to 4 [Dow94, Pad95] and unknown beyond. All the results above hold for the languages with explicit typing, which corresponds to the Church-style simply typed lambda calculus with a single constant type $o$ and no variable types.

The higher-order matching problem in the simply typed lambda calculus $\Lambda$ due to G. Huet [Hue76] is formally stated as follows: given an equation $s(x_1, \ldots, x_n) \approx_{\beta\eta} t$, where $s, t$ are $\lambda$-terms of fixed simple types, and $t$ does not contain free variables, do there exist simply typed terms $s_1, \ldots, s_n$ of appropriate types such that $s[s_1/x_1, \ldots, s_n/x_n] = t$ modulo $\beta\eta$-equality? Here the simply typed lambda calculus $\Lambda$ refers to the Church-style system, where terms are annotated with types generated from a single constant type $o$ by a single type constructor $\rightarrow^1$.

Despite all efforts it still remains open whether the above problem is decidable or not. Several approximation results are as follows. Third-order [Dow94] and fourth-order matchings [Pad95] are decidable. Higher-order matching in extensions of $\Lambda$ with dependent types and type constructors is undecidable [Dow91]. It is also undecidable in the polymorphic second-order system $\lambda2$ and in Gödel’s system $T$ [Dow93]. On the other hand, third-order matching may be decidable or undecidable in $\Lambda$ extended with type constructors [Spr95]. R. Statman showed [Sta82] that the decidability of the Plotkin-

\footnote{This should be contrasted to the simply typed lambda calculus as defined, e.g., in [Bar92] (Sections 5.1, 3.2) or in [TS96] (Section 1.2), which do have variable types. All systems in Barendregt’s $\lambda$-cube have variable types.}
Statman’s definability conjecture would have implied the decidability of the higher-order matching, but R. Loader has proved [Loa93] the undecidability of the lambda definability. D. Wolfram [Wol93] has suggested a higher-order matching algorithm and conjectured its finite termination. The lower complexity bound for the higher-order matching problem in $\Lambda$ is the tower $2^{2^{c \cdot n / \log(n)}}$ of height $c \cdot n / \log(n)$, where $n$ is the length of equation and $c > 0$ [Vor97].

Note that the higher-order rewriting community, without waiting until the ultimate solution for the higher-order matching problem in $\Lambda$, switched to the so-called higher-order patterns [Mil91, Nip91, Nip93] of very restricted linear form. The abovementioned lower bound justifies such a decision.

In this paper we address the higher-order matching problem in the simple theories of type assignment (essentially due to R. Hindley), the so-called systems $\lambda \rightarrow \text{Curry}$ [Bar92], $\lambda_\rightarrow [\text{Mit96}, TA_\lambda [\text{Hin97}])$, closely related to the simply typed lambda calculus $\Lambda$ and typing the same set of terms (modulo type erasure). The importance of the problem stems from the fact that the simple theories of type assignment form a core of the ML type inference system $\lambda \rightarrow, \Pi, \text{let}$ [Mit96] (Section 11.3), which extends $\lambda_\rightarrow$ by adding type abstraction, type application, and polymorphic let declaration. The main result of our paper implies that given two simple ML terms (without type abstractions, applications, and polymorphic let’s), it is undecidable whether one is an instance of the other obtained by substituting some terms for its free variables. Our undecidability result has direct consequences on the ML-style pattern matching, or more generally, on all higher-order pattern matching languages based on implicit typing. It should be noted, however, that for practical reasons and to avoid complications the designers of ML defined ML-patterns to be linear, i.e., with each variable occurring at most once. Now this wise but somehow arbitrary solution gets a theoretical justification.

As a technical tool we use the reduction from the recursive inseparability of the sets of unsatisfiable and finitely satisfiable sentences in the first-order language of a binary relation (Trakhtenbrot-Vaught). This makes the proof especially easy, because a sentence satisfied in a finite binary relation can be realized (witnessed) by a third-order $\lambda$-definable functionals of order three over projection functions (which is not the case for infinitely satisfiable sentences). Usually undecidability proofs for higher-order matching are based on Goldfarb’s result, or on direct reduction from Hilbert’s tenth problem [Dow91, Dow93].

We start by defining the calculi and stating the problem. Then we briefly explain the recursive inseparability techniques and proceed to the reduction from sentences of binary relation to instances of the higher-order matching problem.
2 Simple Theory of Type Assignment

The simple theory of type assignment \(^2\) refers to a class of Curry-style systems of type assignment to untyped \(\lambda\)-terms, including the systems \(\lambda\rightarrow\text{Curry} \ [\text{Bar92}]\) (pp. 148–150), \(\lambda_7\) with implicit typing [Mit96] (p. 782), \(TA_\lambda \ [\text{Hin97}]\). The main characteristic features of these systems are: the presence of type variables and the absence of built-in types in terms\(^3\).

**Definition 1 (Types)** Simple types are generated by \(T_o \equiv o \mid (T_o \rightarrow T_o)\), where \(o\) is a single type constant and \(\rightarrow\) is a single functional type constructor. Types are generated by \(T = o \mid \alpha \mid (T \rightarrow T)\), where \(\alpha\) is a single variable. Types are denoted by \(\sigma, \tau\). The order of a simple type is defined by \(\text{ord}(o) = 1\) and \(\text{ord}(\sigma \rightarrow \tau) = \max\{1 + \text{ord}(\sigma), \text{ord}(\tau)\}\).

**Remark 2** Our definition and results extend to finitely or infinitely many type constants and/or variables. We confine ourselves to the simplest simple theory of type assignment. H. Barendregt [Bar92] does not use type constants in \(\lambda\rightarrow\text{Curry}\). We also do not need the type constant \(o\) if the orders of types are irrelevant. \(\Box\)

**Example 3** Let \(O_0 = o\) and \(O_{n+1} = (o \rightarrow O_n)\). Every second-order simple type equals \(O_n\) for some \(n \geq 1\).

**Convention 4** We adopt the usual conventions on omitting parentheses in writing types with \(\rightarrow\) associating to the right. For \(k \in \omega\), \(\sigma^k \rightarrow \tau\) abbreviates \(\underbrace{\sigma \rightarrow \ldots \rightarrow \sigma}_{k} \rightarrow \tau\).

**Definition 5 (Terms)** Let \(\mathcal{V} = x, y, z, u, v, w, \ldots\) (probably, with indices) be an infinite set of variables. Untyped \(\lambda\)-terms are generated by

\[ T = \mathcal{V} \mid T T \mid \lambda x.T \]

Define the set of free variables in a term by \(\text{FV}(x) = \{x\}\), \(\text{FV}(MN) = \text{FV}(M) \cup \text{FV}(N)\), \(\text{FV}(\lambda x . M) = \text{FV}(M) \setminus \{x\}\). A term \(t\) is closed if \(\text{FV}(t) = \emptyset\).

**Remark 6** We do not introduce, although we could, term constants. \(\Box\)

\(^2\)The term suggested by J. Roger Hindley, private communication, May 6, 1997.
\(^3\)It is worth noting that all systems in \(\lambda\)-cube [Bar92] do have variable types.
**Definition 7 (Type Assignment Systems \(\lambda\rightarrow\text{-Curry}, \lambda_t^\rightarrow, \text{TA}_\lambda\)**) Let \(\Gamma\) denote a basis or a context, a finite set of type assumptions \(x_i : \sigma_i\) for different variables \(x_i\). The type assignment rules are as follows [Bar92]:

\[
\begin{align*}
\Gamma \vdash x : \sigma & \quad \text{(if } x : \sigma \in \Gamma \text{)} \\
\Gamma, x : \sigma \vdash M : \tau \\
\Gamma \vdash \lambda x. M : \sigma \rightarrow \tau \\
\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma \\
\Gamma \vdash (MN) : \tau
\end{align*}
\]

A term \(t\) has (or can be assigned) type \(\tau\) in context \(\Gamma\) iff \(\Gamma \vdash t : \tau\). A term \(t\) is typeable in a context \(\Gamma\) iff there exists a type \(\sigma\) such that \(\Gamma \vdash t : \sigma\). A term \(t\) is typeable iff it is typeable in the empty context. We write \(\vdash_{\lambda\rightarrow\text{-Curry}}\) whenever we need to stress typeability in \(\lambda\rightarrow\text{-Curry}\).

This type assignment system is a core of the ML type inference system \(\lambda\rightarrow\text{n.let}\) [Mit96] (Section 11.3), which extends \(\lambda_t^\rightarrow\) by adding type abstraction, type application, and polymorphic \texttt{let} declaration.

**Definition 8 (Higher-Order Matching Problem in \(\lambda\rightarrow\text{-Curry})**
Given two \(\lambda\)-terms \(s\) and \(t\) with \(\text{FV}(s) = \{x_1, \ldots, x_k\}\) and \(\text{FV}(t) = \emptyset\), such that \(\lambda x_1\ldots x_k.s\) and \(t\) can be assigned some types \(\sigma_1 \rightarrow \cdots \rightarrow \sigma_k \rightarrow \tau\) and \(\tau\) respectively, do there exist closed terms \(s_1, \ldots, s_k\) with types \(\sigma_i\) \((i = 1, \ldots, n)\) such that

\[s[s_1/x_1, \ldots, s_k/x_k] =_{\beta_0} t\]

In the positive case the problem is solvable.

The problem is said to be of order \(n\) if \(\max_{i=1}^k(\sigma_k) \leq n\). \(\Box\)

The main result of this paper is the following

**Theorem 9** The higher-order matching in simple type assignment systems \(\lambda\rightarrow\text{-Curry}, \lambda_t^\rightarrow, \text{TA}_\lambda\) is undecidable, even if terms sought for are of types of order \(\leq 3\).

**Remark 10** We prove, in fact, a stronger result: the problem is undecidable even if a variable-free term \(t\) on the right-hand side is always of the form \(\lambda x_1\ldots x_p.(\lambda uv.u)\) — it simply ignores its arguments and unconditionally yields \texttt{true}.

\(\Box\)
3 Recursive Inseparability

A useful method of proving undecidability is by reduction from a pair of recursively inseparable sets. Two disjoint sets $A$, $B$ are called recursively inseparable iff there are no recursive sets $C$ containing $A$ and disjoint with $B$, i.e., $A \subseteq C$, $B \cap C = \emptyset$. Obviously, each of the recursively inseparable sets $A$, $B$ is non-recursive.

**Proposition 11** Let $A$, $B$ be recursively inseparable and $f$ be a total recursive function. Then $f(A)$, $f(B)$ are also recursively inseparable.

**Proof.** Suppose $f(A)$, $f(B)$ are separable by a recursive set $C$. Then $A$, $B$ are separable by the recursive set $f^{-1}(C)$, the co-image of $C$. Indeed, $f^{-1}(C)$ is recursive: to test whether $x \in f^{-1}(C)$, compute $f(x)$ and check if $f(x) \in C$.

Moreover, $f^{-1}(C)$ separates $A$ and $B$. In fact, $f(A) \subseteq C$ implies $A \subseteq f^{-1}(f(A)) \subseteq f^{-1}(C)$. If we suppose that $f^{-1}(C) \cap B \neq \emptyset$ then $C \cap f(B) \neq \emptyset$, and we get a contradiction. \qed

We will apply the above method as follows. Let $\mathcal{M}$ be the set of all solvable higher-order matching problems (i.e., problems that have solutions). We will take a pair of well-known recursively inseparable sets $U$ and $FS$ and describe a total recursive function $f$ such that $f(U) \subseteq \overline{\mathcal{M}}$ and $f(FS) \subseteq \mathcal{M}$. By Proposition 11, $\mathcal{M}$ cannot be recursive; otherwise $f(U)$ and $f(FS)$ would be recursively separable.

The well-known pair of recursively inseparable sets $U$ and $FS$ is due to B. Trakhtenbrot and R. Vaught. Consider the first-order language $L_0$ without equality containing just one binary predicate symbol $P$. Any model for $L_0$ is called a binary relation. A binary relation is finite iff its extension (domain) is finite. Let $U$ be the set of all sentences of $L_0$ false ($Unsatisfiable$) in all binary relations, and $FS$ be the set of all sentences of $L_0$ satisfied by some finite binary relation ($Finitely Satisfiable$).

**Theorem 12 (Trakhtenbrot-Vaught)** The sets $U$ and $FS$ are recursively inseparable. \qed

Trakhtenbrot proved it for several unary and binary predicate symbols [Tra53], Vaught reduced it to just one binary predicate symbol, [Vau60], Section 4.
4 Transformation Sequence

Given a prenex sentence $\Delta_0$ of the language $L_0$ (containing just one binary predicate symbol $P$ and no equality) we will describe a recursive function $f$ yielding an instance of the higher-order matching problem $f(\Delta_0)$ with the properties:

\begin{align*}
\Delta_0 \in U &\Rightarrow f(\Delta_0) \not\in \mathcal{M} & (1) \\
\Delta_0 \in FS &\Rightarrow f(\Delta_0) \in \mathcal{M} & (2)
\end{align*}

where $U$ is the set of sentences of $L_0$ false in all binary relations (unsatisfiable), $FS$ is the set of sentences of $L_0$ true in some finite binary relation (finitely satisfiable), and $\mathcal{M}$ is the set of all higher-order matching problems that have solutions.

By Theorem 12 and Proposition 11, the existence of such a recursive function $f$ implies that the set $\mathcal{M}$ is non-recursive, i.e., the higher-order matching problem is undecidable. It is conceptually easier to divide the description of $f$ in the sequence of transformation steps, including: 1) translation to a higher-order language and relativization, 2) transformation to the relativized and then pure $\exists^*\forall^*$-form (skolemization), 3) elimination of predicates and boolean connectives for the boolean functions, 4) transformation to the pure existential form, and 5) to a higher-order matching problem. All the transformation steps will be done effectively, as described in the following sections.

The central idea behind our proof is that every sentence about a binary relation satisfied by a finite $n$-element model of binary relation has a $\lambda$-definable realization by means of third-order functionals over projection functions $\lambda x_1 \ldots x_n.x_i$ ($i = 1, \ldots, n$), whereas an unsatisfiable sentence does not have any such realization. A higher-order matching problem we get by transformation is solvable iff such a realization exists.

5 Language $L_1$. Relativization: Translation of $L_0$ into $L_1$

Let $L_1$ be the language without equality containing variables of type $\alpha$, the usual boolean connectives, quantifiers, and two distinguished predicate symbols, unary $R$ and binary $P$, both taking arguments of type $\alpha$. Define the translation $\overline{\alpha}$ of formulas of the language $L_0$ into formulas of the language $L_1$ as follows:
\((P(x, y))\alpha \equiv_{df} P(x^\alpha, y^\alpha),\)
\((\Phi \land \Psi)\alpha \equiv_{df} \Phi^\alpha \land \Psi^\alpha,
(similarly for the other boolean connectives),
\((\forall x.\Phi)\alpha \equiv_{df} \forall x^\alpha.[R(x^\alpha) \Rightarrow \Phi^\alpha],\)
\((\exists x.\Phi)\alpha \equiv_{df} \exists x^\alpha.[R(x^\alpha) \land \Phi^\alpha].\)

**Remark 13** This translation is the usual relativization with respect to the
unary predicate \(R\), plus labeling every variable occurrence with type \(\alpha\). Keeping
variables explicitly typed will give us additional information on the orders
and structure of types for which the problem is undecidable. \(\square\)

**Convention 14** In the sequel it will be convenient to abbreviate \(\forall x^\alpha.[R(x^\alpha) \Rightarrow \Phi]\) by \(\forall^R x.\Phi\) and \(\exists x^\alpha.[R(x^\alpha) \land \Phi]\) by \(\exists^R x.\Phi\).

**Definition 15 (Interpretation of \(L_1\))** The language \(L_1\) is interpreted in
the full type hierarchy over the base domain \(\omega\) of natural numbers. Define
\[T_\omega \equiv_{df} \bigcup_{\tau \in T_\alpha} H_\tau,\]
where \(H_\alpha = \omega\) and \(H_{\omega \rightarrow \tau} = \{\text{the set of all functions from } H_\tau \text{ to } H_\tau\}\).

Let \(\alpha\) be an arbitrary type from \(T_\alpha\). Fix interpretations of the predicate
symbols \(R\), \(P\) as arbitrary subsets \(R \subseteq H_\alpha\) and \(P \subseteq H_\alpha \times H_\alpha\). The validity
relation \(\langle T_\omega, R, P \rangle \models \Phi\) for a sentence \(\Phi\) of \(L_1\) is defined in the usual
manner.

We will now relate validity of sentences of \(L_0\) and \(L_1\).

**Lemma 16** Let \(\Delta_0\) be a sentence of \(L_0\) and \(\Delta_1\) in \(L_1\) be its \(\alpha\)-translation.

1. If for some type \(\alpha \in T_\alpha\) and some interpretations \(R \subseteq H_\alpha\), \(P \subseteq H_\alpha \times H_\alpha\) of predicate symbols \(R\), \(P\) one has \(\langle T_\omega, R, P \rangle \models \Delta_1\), then \(\Delta_0\) is true in some model of binary relation, or, equivalently, \(\Delta_0 \not\in U\).

2. If the sentence \(\Delta_0\) is true in some finite model \(\langle \{a_1, \ldots, a_n\}; B \rangle\) (where \(B \subseteq \{a_1, \ldots, a_n\}^2\) of binary relation, then \(\langle T_\omega, R, P \rangle \models \Delta_1\) for the
type \(\alpha \equiv o^n \rightarrow o\) and for the finite interpretations \(R \subseteq H_\alpha\), \(P \subseteq H_\alpha \times H_\alpha\) of predicate symbols \(R\), \(P\) defined by \(R = \{g_1, \ldots, g_n\}\)
(where \(g_i \equiv \lambda x_1^\alpha \ldots \lambda x_i^\alpha \cdot x_i\) of type \(\alpha \equiv o^n \rightarrow o\) for \(i = 1, \ldots, n\) and
\(P = \{\langle g_i, g_i \rangle \mid \langle a_i, a_j \rangle \in B\}\).

**Proof.** 1) A model \(\langle T_\omega, R, P \rangle \models \Delta_1\) can be straightforwardly transformed
into a model of binary relation satisfying \(\Delta_0\). 2) Obvious. \(\square\)
6 Reduction to $\exists^*\forall^*$-Sentences, Language $L_2$

Consider the sentences of $L_1$ resulting from the $^0$-translation of the prenex sentences of $L_0$. Let us call these sentences quasi-prenex, since according to our convention they have form $Q^R_{i_1} \ldots Q^R_{n_1} \Phi$, where $Q_i \in \{\forall, \exists\}$ and $\Phi$ is quantifier-free.

The language $L_2$ is like $L_1$, but allows variables of types of the form $\alpha^k \rightarrow \alpha$, and quantification over these variables. In this section we show that every such quasi-prenex sentence is equivalent to an $\exists^*\forall^*$-quasi-prenex sentence of the language $L_2$ of the form $\exists w^\sigma_1 \ldots \exists w^\sigma_p \forall^R u^\alpha_1 \ldots \forall^R u^\alpha_q \Phi$, where $\Psi$ is a quantifier-free formula of $L_2$ and the types $\sigma_i$ are of the form $\alpha^k \rightarrow \alpha$. The transformation we use is the usual “skolemization”. Notice that the full type hierarchy $T_\omega$ contains all skolem functions.

**Lemma 17** Every quasi-prenex sentence of $L_1$

$$\Delta_1 \equiv Q^R_{i_1} \ldots Q^R_{n_1} \Phi,$$

where $Q_i \in \{\forall, \exists\}$ and $\Phi$ quantifier-free, can be transformed into an $\exists^*\forall^*$-quasi-prenex sentence of $L_2$ of the form

$$\Delta_2 \equiv \exists w^\sigma_1 \ldots \exists w^\sigma_p \forall^R y^\alpha_1 \ldots \forall^R y^\alpha_q \Psi,$$

such that $\Psi$ is a quantifier-free formula of $L_2$, the types $\sigma_i \equiv \alpha^r_i \rightarrow \alpha$ (for $1 \leq i \leq p$ and $r_i \in \omega$), and the following properties hold:

1. If for some type $\alpha \in T_\omega$ and some interpretations $R \subseteq H_\alpha$ for $R$ and $P \subseteq H_\alpha \times H_\alpha$ for $P$ one has $\langle T_\omega, R, P \rangle \models \Delta_2$, then for these interpretations one also has $\langle T_\omega, R, P \rangle \models \Delta_1$.

2. Let $\langle T_\omega, R, P \rangle \models \Delta_1$ for some type $\alpha$ of the form $o^n \rightarrow o$ ($n \in \omega$) and finite interpretations $R \subseteq H_\alpha$, $P \subseteq R \times R$ of predicate symbols $R$, $P$ such that (cf., Lemma 16.2) $R = \{p_1, \ldots, p_n\}$ (where $p_i = \lambda x^o_1 \ldots \lambda x^n_{o_i} x_i$, $i = 1, \ldots, n$) and $P \subseteq R \times R$.

Then $\langle T_\omega, R, P \rangle \models \Delta_2$, and, moreover, the witness functions for the existentially quantified variables $w^\sigma_1 \ldots w^\sigma_p$ in (4) may be chosen from the finite sets of functions $(P_n)^k \rightarrow P_n$, where $P_n = \{\lambda x^o_1 \ldots \lambda x^n_o x_i \mid i = 1, \ldots, n\}$.

**Proof.** If the formula (3) is already in the form (4), nothing has to be done. Otherwise repeatedly select the rightmost maximal $\forall^*\exists^*$-blocks (with nonempty $\forall^*$- and $\exists^*$-parts) and transform them into $\exists^*\forall^*$-blocks as described.
below. This repeated transformation always terminates, since the number of such \( \forall^n \exists^m \)-blocks reduces at each step.

Consider the rightmost maximal \( \forall^n \exists^m \)-block (with nonempty \( \forall \) and \( \exists \) parts):

\[
\begin{align*}
\ldots \exists^R y_q^n \forall^R x_1^\alpha \ldots \forall^R x_m^\alpha \exists y_1^{\tau_1} \ldots \exists y_n^{\tau_n} \left[ \forall^R z_1^\alpha \ldots \forall^R z_p^\alpha \Phi_i \right].
\end{align*}
\]

(5)

The following invariant properties hold at every transformation step. They are immediately checked by induction on the number of transformations.

- Part (A) is either empty or ends with an existential quantifier.

- Part (D) may contain only universal quantifiers.

- All quantifiers in parts (A), (B), (D) are \( R \)-relativized, and all quantified variables in (A), (B), (D) are of type \( \alpha \).

- All quantified variables in part (C) are of types \( \sigma_i \) of the form \( \alpha^{r_i} \to \alpha \) for \( r_i \in \omega \).

Let \( \overline{w} \) be the vector of all variables universally quantified in parts (A) and (B) (all these variables are of type \( \alpha \)). The transformation step consists in replacement of the \( \forall^n \exists^m \)-quantifier block (B)(C) in (5) with the quantifier block \( \exists^n \forall^m \) (C')(B'), and modifying the part (D) into (D') as shown below:

\[
\begin{align*}
\begin{array}{cccc}
\ldots \exists^R y_q^n \forall^R x_1^\alpha \ldots \forall^R x_m^\alpha \exists f_1 \ldots \exists f_n \forall^R x_1^{\tau_1} \ldots \forall^R x_m^{\tau_m} [\forall^R z_1^\alpha \ldots \forall^R z_p^\alpha \Phi[f_i(\overline{w})/y_i^{\tau_i}]],
\end{array}
\end{align*}
\]

(6)

where \( \tau_i = \alpha^{r_i+m} \to \sigma_i \) for \( 1 \leq i \leq n \) and some \( m \geq 0 \).

This is an equivalence transformation in the full type hierarchy \( T_\omega \), which contains all possible (skolem) functions: if for every \( x \) there exists a \( y \) such that \( \phi(x, y) \), then there exists a function \( h \) such that \( \forall x. \phi(x, h(x)) \). [In principle, the equivalence claim here requires the Axiom of Choice. However, in Lemma 17.1 we do not need equivalence, just a weakening \( \exists \forall \Rightarrow \exists \). For the claim of Lemma 17.2 the Axiom of Choice is unneeded.]

The claims of the lemma are obtained by induction on the number of transformations described above. \( \square \)
7 Transformation to the Pure Prenex $\exists^*\forall^*$-Form

After the previous transformation the sentence $\Delta_2$ in (4) may contain the relativized quantifiers. Transform it to the pure prenex sentence $\Delta_3$, see (7) below, by applying the usual logical equivalences for $Q \in \{\exists, \forall\}$ to push quantifiers outwards:

$$\exists x (R(x) \land Qy \Phi(x, y)) \iff \exists x Qy (R(x) \land \Phi(x, y)),$$
$$\forall x (R(x) \Rightarrow Qy \Phi(x, y)) \iff \forall x Qy (R(x) \Rightarrow \Phi(x, y)).$$

8 Getting Rid of Boolean Connectives and Predicates

The sentence $\Delta_3$, equivalent to $\Delta_2$ in (4), we obtained so far is of the form

$$\Delta_3 \equiv \exists y \forall x \bar{\Phi}, \quad (7)$$

with $\Phi$ quantifier-free built of variables of types $\alpha^k \to \alpha$, boolean connectives, and predicate symbols $R$ (unary), $P$ (binary) taking arguments of type $\alpha$.

Transform $\Delta_3$ into $\Delta_4$ by getting rid of boolean connectives and predicate symbols $R$, $P$ as follows. Let $\text{bool}$ be the type $\alpha \to \alpha \to \alpha$ of booleans with $t \equiv \lambda x^0.0.x$ (true) and $f \equiv \lambda x^0.0.y$ (false), both of type $\text{bool}$. Let (by slightly abusing notation) $R$, $P$ be variables of types $\alpha \to \text{bool}$ and $\alpha \to \alpha \to \text{bool}$ respectively, and

$$\Delta_4 \equiv \exists R^{\to \text{bool}} \exists P^{\to \alpha \to \text{bool}} \exists y \forall x (\bar{\Phi}^* = t), \quad (8)$$

where the * transformation is defined as follows:

1. $R(t)^* = R(t)$ and $P(s, t)^* = P(s, t)$ (with predicate symbols $R$, $P$ on the left and function variables $R$, $P$ of types $\alpha \to \text{bool}$, $\alpha \to \alpha \to \text{bool}$ on the right),

2. $(-\Phi)^* = \text{Cond} (\Phi^*)$ if, $t$,

3. $(\Phi \land \Psi)^* = \text{Cond} (\Phi^*) (\text{Cond} (\Psi^*)$ if $f), t$,

4. $(\Phi \lor \Psi)^* = \text{Cond} (\Phi^*) t (\text{Cond} (\Psi^*)$ if $f), t$,

where $\text{Cond} = \lambda x^\text{bool} \lambda y^\text{bool} \lambda z^\text{bool}.(\lambda a^\alpha \lambda b^\beta. x(yab)(zab))$

(one can easily check that $\text{Cond} t x y = x$ and $\text{Cond} f x y = y$).
Lemma 18 1. If for some type \( \alpha \in T_\omega \) one has \( T_\omega \models \Delta_4 \), then for some interpretations \( R \subseteq H_\alpha \) and \( P \subseteq H_\alpha \times H_\alpha \) for predicate symbols \( R \) and \( P \) one has \( \langle T_\omega, R, P \rangle \models \Delta_3 \).

2. Let \( \langle T_\omega, R, P \rangle \models \Delta_3 \) for some type \( \alpha \) of the form \( o^n \to o \) \((n \in \omega)\) and finite interpretations \( R \subseteq H_\alpha \) and \( P \subseteq H_\alpha \times H_\alpha \) of predicate symbols \( R, P \) such that (cf., Lemmas 16.2 and 17.2) \( R = \{ p_1, \ldots, p_n \} \) (where \( p_i = \lambda x_1^0 \cdots \lambda x_n^0.x_i, \ i = 1, \ldots, n \) and \( p \subseteq R \times R \).

Then \( T_\omega \models \Delta_4 \), and, moreover, the witness functions for the existentially quantified variables in (8) may be chosen from the finite sets of functions \((P_n)^k \to P_n\) or \((P_n)^k \to \text{bool}\), where \( P_n = \{ \lambda x_1^0 \cdots \lambda x_n^0.x_i | h = 1, \ldots, n \} \).

Proof. 1) Let for some type \( \alpha \in T_\omega \) one has \( T_\omega \models \Delta_4 \). Then for some functions \( R \in H_\alpha \to \text{bool} \) \( P \in H_\alpha \to \text{bool} \) one has \( T_\omega \models \exists \Phi \forall x (\Phi = t) \).

Define the interpretations \( R \subseteq H_\alpha \) and \( P \subseteq H_\alpha \times H_\alpha \) for the predicate symbols \( R \) and \( P \) as \( R = \{ u | Ru = t \} \) and \( P = \{ \langle u, v \rangle | Puv = t \} \). Then induction on the definition of the * translation shows that for all values of free parameters of \( \Phi^* \) (except \( R, P \)) one has: \( \Phi \) is true iff \( \Phi^* = t \).

2) Straightforward.

9 Transformation to a Higher-Order Matching Problem

Transform the sentence \( \Delta_4 \) in (8) we obtained so far into a higher-order matching problem by: 1) omitting the leading existential quantifiers, 2) eliminating the trailing universal quantifiers by rewriting \( \forall \bar{x}(\Phi^* = t) \) into \( \lambda \bar{x}.\Phi^* = \lambda \bar{x}t \), and 3) erasing all type information from the last equation, thus getting

\[
\Delta_5 \equiv_{df} \lambda \bar{x}.\Phi'^* = \lambda \bar{x}(\lambda u.\tilde{u}), \quad (9)
\]

where \( \Phi'^* \) is obtained from \( \Phi^* \) by erasing all types. We have the following

Lemma 19 Let the matching problem \( \Delta_5 \) in (9) be obtained from the sentence \( \Delta_4 \) in (8) by the transformation described above.

1. If the matching problem \( \Delta_5 \) has a solution, then \( T_\omega \models \Delta_4 \) for some type \( \alpha \in T_\omega \).
2. Let $T_\omega \models \Delta_4$ for some type $\alpha$ of the form $\alpha^n \to \alpha$ ($n \in \omega$) and, moreover, the witness functions for the existentially quantified variables in (8) may be chosen from the finite sets of functions (cf., Lemmas 16.2, 17.2, and 18.2) $(P_n)^k \to P_n$ or $(P_n)^k \to \text{bool}$, where $P_n = \{\lambda x_1^o \cdots \lambda x_n^o . x_i \mid i = 1, \ldots, n\}$. Then the matching problem $\Delta_5$ has a solution of order $\leq 3$.

**Proof.** 1) Obvious, because for all closed terms $\bar{t}$ one has $T_\omega \models \Psi[\bar{t}/\bar{x}] \Rightarrow \exists x \Psi$ for all $\Psi$ (logical rule), and $T_\omega \models \lambda x.s = \lambda x.t$ is equivalent to $T_\omega \models \forall x(s = t)$.

2) Every function in the finite sets of functions $(P_n)^k \to P_n$ or $(P_n)^k \to \text{bool}$, where $P_n = \{\lambda x_1^o \cdots \lambda x_n^o . x_i \mid i = 1, \ldots, n\}$ is $\lambda$-definable by a closed term. In fact, every $\lambda x_1^o \cdots \lambda x_n^o . x_i$ is $\lambda$-definable, and the equality between two elements of $P_n$ is $\lambda$-definable ([HKM97], Section 2.4) by

$\EQ_n \equiv \exists q \lambda p^{\alpha^o \to q} • q^{\alpha^o \to \alpha} • p(\nu \nu \cdots\nu)(\nu\nu\nu \cdots \nu) \cdots (\nu \cdots \nu \cdots \nu)$.

$(\EQ_n(\lambda x_1 \cdots x_n . x_i)(\lambda x_1 \cdots x_n . x_j))$ normalizes to $t$ if $i = j$ and to $f$ otherwise.

Using $\EQ$, every function in the finite sets of functions $(P_n)^k \to P_n$ or $(P_n)^k \to \text{bool}$, where $P_n = \{\lambda x_1^o \cdots \lambda x_n^o . x_i \mid i = 1, \ldots, n\}$ is easily lambda-definable by a $\text{Cond}$-expression encoding a finite table. The order of any term defining a function in $(P_n)^k \to P_n$ or $(P_n)^k \to \text{bool}$ is $\leq 3$. \hfill \Box

10 Faithfulness of the Reduction

Let $f(\Delta_0) = \Delta_5$, where the sequence of transformations $\Delta_0 \leadsto \Delta_1 \leadsto \Delta_2 \leadsto \Delta_3 \leadsto \Delta_4 \leadsto \Delta_5$ is described in the previous sections. It is obvious from the description of all transformations that the function $f$ is recursive. We now turn to the proof that $f$ satisfies the properties (1) and (2).

**Case $\Delta_0 \in U$.** Instead of (1) we prove equivalent $f(\Delta_0) \in M \Rightarrow \Delta_0 \not\in U$.

Let the higher-order matching problem $\Delta_5 = f(\Delta_0)$ have a solution.

1. Lemma 19.1 implies that $T_\omega \models \Delta_4$ for some type $\alpha \in T_\omega$.

2. Lemma 18.1 implies that $(T_\omega, R, P) \models \Delta_{2,3}$ ($\Delta_2$ and $\Delta_3$ are equivalent) for this type $\alpha \in T_\omega$ and some interpretations $R \subseteq H_\alpha$, $P \subseteq H_\alpha \times H_\alpha$ for $R$, $P$.  

13
3. Lemma 17.1 implies that \( \langle T_\omega, R, P \rangle \models \Delta_1 \) for this type \( \alpha \in T_\omega \) and these interpretations \( R \subseteq H_\alpha, P \subseteq H_\alpha \times H_\alpha \) of predicate symbols \( R \) and \( P \).

4. Lemma 16.1 implies that \( \Delta_0 \) is true in some model of binary relation, i.e., \( \Delta_0 \notin U \) (recall that \( U \) is a set of unsatisfiable sentences about a binary relation).

Thus, (1) is proved.

Remark 20 This part of the proof corresponds to the correctness of transformation (the first clauses of Lemmas 16–19): if the resulting higher-order matching problem \( \Delta_5 \) has a solution, then the initial sentence \( \Delta_0 \) is satisfied by some binary relation.

Case \( \Delta_0 \in FS \). Let a sentence \( \Delta_0 \) of \( L_0 \) be true in a finite model of binary relation \( \langle \{a_1, \ldots, a_n\}; B \rangle \) (where \( B \subseteq \{a_1, \ldots, a_n\}^2 \)) of cardinality \( n \in \omega \).

1. Lemma 16.2 implies that \( \langle T_\omega, R, P \rangle \models \Delta_1 \) for the type \( \alpha \equiv o^n \rightarrow o \) and for the finite interpretations \( R \subseteq H_\alpha, P \subseteq H_\alpha \times H_\alpha \) of predicate symbols \( R, P \) defined by \( R_\alpha = \{g_1, \ldots, g_n\} \) (where \( g_i = \lambda x_1^o \ldots \lambda x_n^o . x_i \) for \( i = 1, \ldots, n \)), \( P_\alpha = \{\langle g_i, g_j \rangle \mid \langle a_i, a_j \rangle \in B \} \).

2. Lemma 17.2 implies that \( \langle T_\omega, R, P \rangle \models \Delta_2,3 \), and, moreover, the witness functions for the existentially quantified variables \( w_1^o \ldots w_p^o \) in (4) may be chosen from the finite sets of functions \( (P_n)^k \rightarrow P_n \)

where \( P_n = \{\lambda x_1^o \ldots \lambda x_n^o . x_i \mid i = 1, \ldots, n \} \).

3. Lemma 18.2 implies that \( T_\omega \models \Delta_4 \), and, moreover, the witness functions for the existentially quantified variables in (8) may be chosen from the finite sets of functions \( (P_n)^k \rightarrow P_n \) or \( (P_n)^k \rightarrow \text{bool} \)

where \( P_n = \{\lambda x_1^o \ldots \lambda x_n^o . x_i \mid i = 1, \ldots, n \} \).

4. Lemma 19.2 implies that the matching problem \( \Delta_5 \) has a solution of order \( \leq 3 \).

Thus, (2) and the main claim of this paper are proved.

Remark 21 This part of the proof corresponds to the finite completeness of the transformation (the second clauses of Lemmas 16–19): if a sentence \( \Delta_0 \) is satisfied by a finite binary relation, then the resulting higher-order matching problem \( \Delta_5 \) has a closed \( \lambda \)-definable solution in the form of functionals (of order 3) over projection functions \( P_n \).
Summarizing, the whole proof works smoothly thanks to the helpful dichotomy in Trakhtenbrot-Vaught’s Theorem 12 “unsatisfiable vs. finitely satisfiable” plus the recursive inseparability. This is because we need not care about the case when a sentence $\Delta_0$ is satisfied by an infinite model of binary relation, when we may be unable to find a closed solution to the resulting higher-order matching problem $\Delta_5$. □

References


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

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

<table>
<thead>
<tr>
<th>Report</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI-I-97-2-005</td>
<td>L. Bachmair, H. Ganzinger</td>
<td>A Theory of Resolution</td>
</tr>
<tr>
<td>MPI-I-97-2-003</td>
<td>U. Hustadt, R.A. Schmidt</td>
<td>On evaluating decision procedures for modal logic</td>
</tr>
<tr>
<td>MPI-I-97-2-002</td>
<td>R.A. Schmidt</td>
<td>Resolution is a decision procedure for many propositional modal logics</td>
</tr>
<tr>
<td>MPI-I-97-2-001</td>
<td>D.A. Basin, S. Matthews, L. Viganò</td>
<td>Labelled modal logics: quantifiers</td>
</tr>
<tr>
<td>MPI-I-96-2-010</td>
<td>A. Nonnengart</td>
<td>Strong Skolemization</td>
</tr>
<tr>
<td>MPI-I-96-2-009</td>
<td>D.A. Basin, N. Klarlund</td>
<td>Beyond the Finite in Automatic Hardware Verification</td>
</tr>
<tr>
<td>MPI-I-96-2-008</td>
<td>S. Vorobyov</td>
<td>On the decision complexity of the bounded theories of trees</td>
</tr>
<tr>
<td>MPI-I-96-2-007</td>
<td>A. Herzig</td>
<td>SCAN and Systems of Conditional Logic</td>
</tr>
<tr>
<td>MPI-I-96-2-006</td>
<td>D.A. Basin, S. Matthews, L. Viganò</td>
<td>Natural Deduction for Non-Classical Logics</td>
</tr>
<tr>
<td>MPI-I-96-2-005</td>
<td>A. Nonnengart</td>
<td>Auxiliary Modal Operators and the Characterization of Modal Frames</td>
</tr>
<tr>
<td>MPI-I-96-2-004</td>
<td>G. Struth</td>
<td>Non-Symmetric Rewriting</td>
</tr>
<tr>
<td>MPI-I-96-2-003</td>
<td>H. Baumeister</td>
<td>Using Algebraic Specification Languages for Model-Oriented Specifications</td>
</tr>
<tr>
<td>MPI-I-96-2-001</td>
<td>H. Ganzinger, U. Waldmann</td>
<td>Theorem Proving in Cancellative Abelian Monoids</td>
</tr>
<tr>
<td>MPI-I-95-2-011</td>
<td>P. Barth, A. Bockmayr</td>
<td>Modelling Mixed-Integer Optimisation Problems in Constraint Logic Programming</td>
</tr>
<tr>
<td>MPI-I-95-2-010</td>
<td>D.A. Plaisted</td>
<td>Special Cases and Substitutes for Rigid E-Unification</td>
</tr>
<tr>
<td>MPI-I-95-2-009</td>
<td>L. Bachmair, H. Ganzinger</td>
<td>Ordered Chaining Calculi for First-Order Theories of Binary Relations</td>
</tr>
<tr>
<td>MPI-I-95-2-008</td>
<td>H.J. Ohlbach, R.A. Schmidt, U. Hustadt</td>
<td>Translating Graded Modalities into Predicate Logic</td>
</tr>
<tr>
<td>MPI-I-95-2-007</td>
<td>A. Nonnengart, A. Szalas</td>
<td>A Fixpoint Approach to Second-Order Quantifier Elimination with Applications to Correspondence Theory</td>
</tr>
</tbody>
</table>