The Variable Containment Problem

Stefan Kahrs[⋆]

University of Edinburgh
Laboratory for Foundations of Computer Science
King's Buildings,
Edinburgh EH9 3JZ
United Kingdom
email: smk@dcs.ed.ac.uk

Abstract. The *essentially* free variables of a term t in some λ-calculus, $\mathrm{FV}_\beta(t)$, form the set $\{x \mid \forall u.\, t =_\beta u \Rightarrow x \in \mathrm{FV}(u)\}$. This set is significant once we consider equivalence classes of λ-terms rather than λ-terms themselves, as for instance in higher-order rewriting.
An important problem for (generalised) higher-order rewrite systems is the *variable containment problem*: given two terms t and u, do we have for all substitutions θ and contexts $C[\,]$ that $\mathrm{FV}_\beta(C[t^\theta]) \supseteq \mathrm{FV}_\beta(C[u^\theta])$? This property is important when we want to consider $t \to u$ as a rewrite rule and keep n-step rewriting decidable. Variable containment is in general *not* implied by $\mathrm{FV}_\beta(t) \supseteq \mathrm{FV}_\beta(u)$. We give a decision procedure for the variable containment problem of the second-order fragment of $\lambda^\to$. For full $\lambda^\to$ we show the equivalence of variable containment to an open problem in the theory of PCF; this equivalence also shows that the problem is decidable in the third-order case.

1 Introduction

As soon as we make the step from terms to equivalence classes of terms as the objects of our interest, the question whether a variable occurs (free) in such an object becomes a bit delicate. Should the variable occur in all terms of the class or only in some, or can we sensibly ask this question at all?

Typically, the equivalence relation $\equiv$ in question is preserved by substitution application, i.e. $t \equiv u$ implies $t^\theta \equiv u^\theta$ for arbitary substitutions θ. In particular, if $t \equiv u$ and x is free in t but not in u then $t[y/x] \equiv u[y/x] = u$ for any variable (or term) y. This suggests the following definition:

Definition 1. Let $=_e$ be a substitutive equivalence relation. The *free variables modulo* $=_e$ of a term t, $\mathrm{FV}_e(t)$, are defined as follows:

$$\mathrm{FV}_e(t) = \bigcap_{t =_e u} \mathrm{FV}(u)$$

⋆ The research reported here was partially supported by SERC grant GR/J07303.

In the above definition I was deliberately a bit vague about basic notions such as term, substitution, and free variable, because the concept makes sense for various (typed or untyped) λ-calculi as well as first-order terms. In the following, we shall concentrate on the equivalence $=_\beta$ and call variables which are free modulo $=_\beta$ "essentially free".

For each equivalence class $[u]_\beta$ that contains a normal form $u{\downarrow}$, we have $\mathrm{FV}_\beta(u) = \mathrm{FV}(u{\downarrow})$. Unfortunately, the set $\mathrm{FV}_\beta(t)$ is in general (for the untyped λ-calculus) not recursive, i.e. the problem $x \in \mathrm{FV}_\beta(t)$ is undecidable: the set $M_x = \{t \mid x \in \mathrm{FV}_\beta(t)\}$ is closed under β-conversion and non-trivial which already implies that M_x is not recursive (theorem 6.6.2 (ii) in [1]); moreover, $t \in M_x \iff x \in \mathrm{FV}_\beta(t)$.

In [1] the notation $x \in_\beta M$ is used instead of $x \in \mathrm{FV}_\beta(M)$ (exercise 3.5.15, notation 4.1.4) — the concept is not really new, even though Barendregt defines it only for λ-theories rather than arbitrary (substitutive) equivalence relations. Similarly, Middeldorp [15] uses the notation $V_{\mathit{fix}}([t]_R)$ to describe the set of variables that occur in every term that is R-equivalent to t.

Most typed λ-calculi studied in the literature [3] have a strongly normalising β-reduction, which implies that $\mathrm{FV}_\beta(t)$ is recursive for each typable term t: we reduce t to its β-normal form $t{\downarrow}$ and find the set as $\mathrm{FV}(t{\downarrow})$.

Moving from terms to equivalence classes of terms is not entirely unproblematic: the property $\mathrm{FV}_\beta(t) \subseteq \mathrm{FV}_\beta(u)$ is in a sense less informative than the ordinary $\mathrm{FV}(t) \subseteq \mathrm{FV}(u)$. From the latter we can deduce $\mathrm{FV}(t^\theta) \subseteq \mathrm{FV}(u^\theta)$ and $\mathrm{FV}(C[t]) \subseteq \mathrm{FV}(C[u])$, which means that the property is a rewrite relation. But we *cannot* deduce from the former $\mathrm{FV}_\beta(t^\theta) \subseteq \mathrm{FV}_\beta([u^\theta])$ or $\mathrm{FV}_\beta(C[t]) \subseteq \mathrm{FV}_\beta(C[u])$. Example: the terms $t \equiv (x\ y)$ and $u \equiv (\lambda z.\ z\ x\ y)$ are in normal form and have the same essentially free variables $\{x, y\}$. But when we apply the substitution $\theta = [(\lambda v.\ x')/x]$ to both terms then x' is essentially free in both but y is only essentially free in u^θ; similarly, the context $C[\_] = \_\,(\lambda v.x')$ also distinguishes these terms: x is essentially free in $C[t]$ but not in $C[u]$.

Why does this matter?

The condition $\mathrm{FV}(t) \supseteq \mathrm{FV}(u)$ is typically used as a requirement for rewrite rules $t \to u$, to make sure that rewriting never introduces free variables. This property is desirable for a number of reasons:

1. Without it, the rewrite system could not be strongly normalising, because rewriting is substitutive and extra variables on the right-hand sides could be instantiated to (terms containing instances of) left-hand sides of rules, including the left-hand side of the very rule with the extra variables.
2. Without it, confluence is unlikely: if $t \to u$ and if u contains an extra variable x then also $t \to u[y/x]$ and confluence would require that u and $u[y/x]$ have a common reduct.
3. To decide the one-step rewrite relation $t \to u$, one has to decide matching problems, i.e. matching occurrences in t to left-hand sides of rules. This remains true if we allow extra variables though we have then an additional matching problem of (a subterm of) u against the instance of the right-hand side of the applied rule. However, if we consider n-step rewriting for $n > 1$,

then we have to solve unification problems if the rules have extra variables.

To make the last point clear: we can encode any unification problem as a two-step rewriting problem of a rewrite system with extra variables as we shall see shortly. By "unification problem" we mean the following.

Definition 2. The *unification problem* $t \doteq u$ is the property $\exists\sigma.\, t^\sigma = u^\sigma$.

Again, I am deliberately vague about what the terms t and u and the substitution σ range over, and how substitution application t^σ is actually defined — terms, substitutions, and their unification problems exist in a variety of formalisms.

Theorem 3. *For any first-order unification problem $t \doteq u$ there is a finite (generalised) rewrite system R and terms C, D such that $t \doteq u \iff C \to_R ; \to_R D$.*

Proof. We choose symbols C, D, F not occurring in t and u. R has two rules: $C \to F(t,u)$ and $F(x,x) \to D$. The required intermediate term E with $C \to_R E \to_R D$ must have the properties $\exists\sigma.\, E = F(t,u)^\sigma$, because we can only apply the first rule to C, and similarly $\exists\tau. E = F(x,x)^\tau$, because we can only apply the second rule backwards to D. Both conditions together are sufficient as neither C nor D are affected by substitution application. Thus, $C \to_R E \to_R D$ is equivalent to the problem $\exists\sigma.\, \exists\tau.\, F(t,u)^\sigma = F(x,x)^\tau$ which is equivalent to $\exists\sigma.\, \exists\tau.\, \tau(x) = t^\sigma \wedge \tau(x) = u^\sigma$ which in turn is equivalent to $t \doteq u$. $\Box$

The same kind of situation appears in higher-order rewriting, where it is even more significant: matching up to fourth-order is known to be decidable [7, 4, 17], but second-order unification is already undecidable [6]. The decidability of higher-order matching is still an open problem but it is often conjectured to be decidable [4].

Remark: In view of Loader's recent result [12] that (absolute) λ-definability for (arbitrary) finite models of $\lambda^\to$ is undecidable this conjecture is rather doubtful. Looking at the details of Loader's proof we can observe that it shows that *relative* λ-definability is already undecidable for third-order types which (see Loader's proof of Lemma 1 in [12]) implies that *absolute* λ-definability for fourth-order types is undecidable too; absolute λ-definability for third-order types is decidable [20].

The question about extra variables is generally asked for the *instance* of a rule, not for the rule itself. We have already seen that this difference matters in the presence of higher-order variables and indeed we do not need extra variables in the rules themselves to solve unification problems:

Theorem 4. *For any second-order unification problem $t \doteq u$ there is a finite (generalised) second-order rewrite system R (with all rules $l \to r$ satisfying $\mathrm{FV}_\beta(l) \supseteq \mathrm{FV}_\beta(r)$) and terms C, D such that $t \doteq u \iff C \to_R ; \to_R D$.*

Proof. We choose fresh symbols D, F, G such that the result type of F and G is first-order, and a fresh second-order variable y. The rules of R are $G\,(y\,(F\,t\,u)) \to (F\,t\,u)$ and as before $F\,x\,x \to D$. We can only apply the first rule to $C \equiv G\,D$ by taking *any* substitution σ such that $\sigma(y) = \lambda z.\, D$ and have the same situation as in the proof of theorem 3. $\Box$

This observation is based on *generalised* higher-order rewrite systems [14]. Originally, HRSs were defined with an additional condition for left-hand sides [16] which we shall not consider here; suffice it to say that the subterm $(y\,(F\,t\,u))$ in the above proof does not satisfy this condition.

Theorem 4 shows that $\mathrm{FV}_\beta(l) \supseteq \mathrm{FV}_\beta(r)$ is clearly not the right condition for general higher-order rewriting if we want to ban extra variables and keep n-step rewriting decidable. We need something stronger, a property which is also a rewrite relation.

There is a general principle behind the last remark. A rewrite relation is a relation closed under substitution application and context application, i.e. $>$ is a rewrite relation iff $t > u$ implies $t^\theta > u^\theta$ for arbitrary substitutions θ, and $C[t] > C[u]$ for arbitrary contexts $C[\;]$. In a typed scenario, the "arbitrary" comes with a typing proviso.

The typical use of the term "rewrite relation" is to form the rewrite closure of a relation R, i.e. the smallest rewrite relation $\rightarrow_R$ which contains R. This is well-defined, because rewrite relations are closed under arbitrary intersections. As they are also closed under arbitrary unions, the dual concept is also well-defined: the *rewrite interior* of R is the largest rewrite relation $\rightarrow^R$ contained in R.

The notion of rewrite interior is useful for the following reason. Sometimes we want to show that all terms in a rewrite relation (given by a rewrite system R) satisfy a certain property, i.e. $t \rightarrow_R u$ implies tSu, more briefly $\rightarrow_R \subseteq S$. The proof will hardly ever work directly, because $\rightarrow_R$ is almost always an infinitary object, it relates infinitely many terms. The solution is to prove instead a property about R itself, since R is typically a finite relation.

Theorem 5. *Let R and S be relations on terms. Then $\rightarrow_R \subseteq S \iff R \subseteq \rightarrow^S$.*

Proof. Trivial by exploiting the following facts: (i) Rewrite interior and rewrite closure are monotonic w.r.t. to $\subseteq$; (ii) any rewrite relation is a fixpoint of both the closure and the interior operator, in particular this applies to $\rightarrow_R$ and $\rightarrow^S$; (iii) $R \subseteq \rightarrow_R$ and $\rightarrow^S \subseteq S$. □

In words: to show that a rewrite closure $\rightarrow_R$ satisfies an invariant S we can show that R satisfies $\rightarrow^S$. In our situation, S is the relation $tSu \iff \mathrm{FV}_\beta(t) \supseteq \mathrm{FV}_\beta(u)$ and *variable containment* is the interior of this relation.

2 Variable Containment in General

Definition 6. Given two terms $t, u \in \Lambda$, their *variable containment problem*, $t \succeq u$, is defined as the following property:

$$t \succeq u \overset{def}{\iff} \forall C[\;].\,\forall\theta.\,\mathrm{FV}_\beta(C[t^\theta]) \supseteq \mathrm{FV}_\beta(C[u^\theta]).$$

For the untyped λ-calculus this is obviously an undecidable problem as even the sets $\mathrm{FV}_\beta(t)$ are generally non-recursive. We can also ignore the "$\forall\theta$" quantifier as any substitution application can occur as the substitution derived from a β-reduction.

For typed λ-calculi the problem has to be slightly restated, restricting t and u to be well-typed preterms in some context[2] Γ and θ a substitution mapping variables in Γ to preterms that type-check (with the same type) in some context Δ. An analogous restriction applies to $C[\ ]$. The exact formulation depends on the particular λ-calculus, though the general principle should be clear.

It is possible to formalise it uniformly for all type systems expressible in the formalism of Pure Type Systems (short: PTS; see [2, 3]), especially the "PTS with signature" as in [5] which support a proper treatment of constant symbols. However, this goes somewhat beyond the scope of this paper and therefore we concentrate on the simply typed λ-calculus $\lambda^{\rightarrow}$ and its fragments.

In order to formulate the appropriate notion of variable containment for typed λ-calculi we have to adapt the notion of substitution accordingly.

Definition 7. We write $\theta : \Gamma \rightarrow \Delta$ if θ is a function from variables to preterms and Γ and Δ are contexts such that:

$$\forall x \in \mathit{Dom}\,\Gamma.\, \Gamma \vdash_{\Sigma} x : \tau \Longrightarrow \Delta \vdash_{\Sigma} \theta(x) : \tau$$

For arbitrary type systems, we would have to formulate a similar though more awkward adaptations for contexts (in the sense: term with hole). However, for $\lambda^{\rightarrow}$ and its n-th order restrictions this is not really necessary due to the following observations. Suppose t and u have a function type, then $t \succeq u \iff t\,x \succeq u\,x$ for some fresh x. Thus we can reduce variable containment of arbitrary types to variable containment of base types. Moreover, if t and u have a base type then $\mathrm{FV}_{\beta}(t) \supseteq \mathrm{FV}_{\beta}(u)$ iff for all $C[\ ]$ we have $\mathrm{FV}_{\beta}(C[t]) \supseteq \mathrm{FV}_{\beta}(C[u])$. This way we can avoid the quantification over contexts by restricting our attention to variable containment for base types. To be precise: this trick requires that substitution does not affect the types, i.e. it does not apply to $\lambda\!\rightarrow$ as presented in [3] where base types are variables — we need them to be constants.

Definition 8. The variable containment problem for $\lambda^{\rightarrow}$ with signature Σ is the following:

$$\begin{aligned}\Gamma \vdash_{\Sigma} t \succeq u : \tau \overset{def}{\Longleftrightarrow} & (\tau : *) \in \Sigma \;\wedge \Gamma \vdash_{\Sigma} t : \tau \;\wedge\; \Gamma \vdash_{\Sigma} u : \tau \;\wedge \\ & \forall \Delta.\, \forall \theta : \Gamma \rightarrow \Delta.\, \mathrm{FV}_{\beta}(t^{\theta}) \supseteq \mathrm{FV}_{\beta}(u^{\theta})\end{aligned}$$

The property $(\tau : *) \in \Sigma$ just means that τ is a base type in the signature.

To decide the variable containment problem we would generally need that $\mathrm{FV}_{\beta}(t)$ is recursive which is the case for all strongly normalising type systems. Then we have to find a semantic domain $(D, \geq)$ to interpret the judgements $\Gamma \vdash_{\Sigma} t : \tau$ such that the predicate $\geq$ and the interpretation function are total

[2] I use the word "context" for terms with holes $C[\ ]$ and also for sets of pairs of variables and types Γ, since it is established terminology for both.

recursive functions and $[\![t]\!] \geq [\![u]\!]$ iff $t \succeq u$. We follow tradition by using double brackets $[\![\_]\!]$ for denoting the semantic interpretation of syntactic objects.

There is no other requirement we need for these domains, i.e. D is just a set and $\geq$ a binary relation on D. Since $\succeq$ is a preorder (easy to show), we would need that $\geq$ is a preorder as well if $[\![\_]\!]$ is surjective.

3 Variable Containment for $\lambda_2^{\rightarrow}$

We begin with the type theory $\lambda_2^{\rightarrow}$, the second-order fragment of the simply typed λ-calculus $\lambda^{\rightarrow}$. In $\lambda_2^{\rightarrow}$, free variables are restricted to at most second-order types, i.e. types of the form $\tau_1 \rightarrow \tau_2 \rightarrow \cdots \rightarrow \tau_n$ such that all τ_i are base types. It is possible to define $\lambda_2^{\rightarrow}$ as a PTS, but we shall not do that here, for it would distract too much from the major issues we want to tackle.

Combinatory reduction systems (CRSs) [10, 11] can be seen as a special class of rewrite systems in the type theory $\lambda_2^{\rightarrow}$ over extensions of the signature $\Sigma = \langle 0\colon *, \Lambda\colon (0 \rightarrow 0) \rightarrow 0\rangle$. To get an exact match, no further type constants (other that 0) or third-order constants (other than Λ) should be allowed. CRSs come equipped with an additional restriction for left-hand side of rules (each free variable is applied to a sequence of distinct bound variables) that makes the variable containment problem trivial — for CRS rules $l \rightarrow r$ the property $l \succeq r$ is equivalent to $\mathrm{FV}(l) \supseteq \mathrm{FV}(r)$.

However, we can drop the restriction for left-hand sides and generalise the definition of second-order rewrite rule.

Definition 9. Given a $\lambda_2^{\rightarrow}$-signature Σ, a Σ-rule is a tuple (Γ, l, r, τ), written $\Gamma \vdash_\Sigma l \rightarrow r : \tau$, such that (i) τ is a base type, (ii) $\Gamma \vdash_\Sigma l : \tau$, (iii) $\Gamma \vdash_\Sigma r : \tau$

An instance of a rule (Γ, l, r, τ) is given by a substitution $\theta : \Gamma \rightarrow \Delta$ and a context $C[\ ]$ such that $E \vdash C[x] : \sigma$ for some type σ, some fresh variable x, and some context E such that $(x : \tau) \in E$ and $\Delta \vdash x : \tau$ is a premise of $E \vdash C[x] : \sigma$, i.e. Δ is the context in which the hole of $C[\ ]$ is being type-checked. We omit the formal definition of the latter, but it can easily be formalised in the style of a type system. We have $t \rightarrow_R u$ for terms t, u with $E \vdash t : \tau$ (analogously for u) if there is a rule $\Gamma \vdash_\Sigma l \rightarrow r : \tau$, a substitution $\theta : \Gamma \rightarrow \Delta$ and a context $C[\ ]$ (as described) such that $C[l^\theta] =_{\beta\eta} t$ and $C[r^\theta] =_{\beta\eta} u$.

Since second-order matching is decidable, we can decide whether we have an instance of a rule, i.e. the rewrite relation $\rightarrow_R$ is decidable for finitely many rules. As for first-order rewriting, the transitive closure of $\rightarrow_R$ is undecidable. As already explained, the two-step rewrite relation $\rightarrow_R ; \rightarrow_R$ is undecidable for general second-order rewrite systems because of extra variables.

Therefore, it makes sense to require $\Gamma \vdash_\Sigma l \succeq r : \tau$ for all rules $\Gamma \vdash_\Sigma l \rightarrow r : \tau$. Since we require that the type τ of a rule is a base type, the proposition $\Gamma \vdash_\Sigma l \rightarrow r : \tau$ is well-formed and according to our general observations for base types it is equivalent to variable containment for the rewrite relation generated from this rule. In the following we shall omit the subscript Σ for judgements.

How can we decide variable containment in $\lambda_2^{\rightarrow}$ for two terms t and u? Take for instance the terms $t = F\,(x\,(y\,z))\,(y\,w)$ and $u = G\,(y\,(x\,z))$ (where w, x, y, z are variables, x and y second-order), do we have $t \succeq u$, or $u \succeq t$, or both, or neither, and how can we find out? For $t \succeq u$ we have to check $\mathrm{FV}_\beta(t^\theta) \supseteq \mathrm{FV}_\beta(u^\theta)$ for all substitutions θ, but this is an infinitary condition. For second-order variable containment only two things matter for a substitution: (i) which variables are free in the substitute, and (ii) for second-order variables v with substitute $\lambda v_1, \cdots, vn.s$, which of the (first-order) variables v_i is free in s? The former limits which variables can be free in the substituted term, from the latter we can find out which subterms will be erased during normalisation. Consider the variables x and y from the example and their substitutes $\theta(x) = \lambda x'.p$ and $\theta(y) = \lambda y'.q$: if x' is free in p and y' is free in q then the free variables in u^θ are exactly $\mathrm{FV}_\beta(\theta(x)) \cup \mathrm{FV}_\beta(\theta(y)) \cup \mathrm{FV}_\beta(\theta(z))$ and $\mathrm{FV}_\beta(t^\theta)$ contains those and also $\mathrm{FV}_\beta(\theta(w))$. Thus $u \not\succeq t$. If y' is not free in q then $\mathrm{FV}_\beta(u^\theta) = \mathrm{FV}_\beta(\theta(y))$ and $\mathrm{FV}_\beta(t^\theta) = \mathrm{FV}_\beta(\theta(x)) \cup \mathrm{FV}_\beta(\theta(y))$. Finally, if $y \in \mathrm{FV}_\beta(q), x \notin \mathrm{FV}_\beta(p)$ then $\mathrm{FV}_\beta(t^\theta) = \mathrm{FV}_\beta(\theta(w)) \cup \mathrm{FV}_\beta(\theta(x)) \cup \mathrm{FV}_\beta(\theta(y))$ and $\mathrm{FV}_\beta(u^\theta) = \mathrm{FV}_\beta(\theta(x)) \cup \mathrm{FV}_\beta(\theta(y))$. So, in all cases we have $\mathrm{FV}_\beta(t^\theta) \supseteq \mathrm{FV}_\beta(u^\theta)$ and thus we have $t \succeq u$.

The general picture is that we have to consider all free variable occurrences in a term and see in which argument positions of which other variables these occurrences are. The following semantic interpretation of terms captures these observations.

Definition 10. For $\lambda_2^{\rightarrow}$, we interpret judgements $\Gamma \vdash t : \tau$ as pointwise ordered functions in $\mathit{Dom}\,\Gamma \rightarrow \wp(\wp((\mathit{Dom}\,\Gamma) \times \mathcal{N}))$ where the order on the codomain is given as:

$$A \geq B \overset{def}{\iff} \forall M \in B.\, \exists N \in A.\, N \subseteq M.$$

We assume in the following that t is in β-normal form, i.e. if it is not then we take $[\![\Gamma \vdash t : \tau]\!] = [\![\Gamma \vdash t{\downarrow} : \tau]\!]$ where $t{\downarrow}$ is the normal form of t.

If t has the form $x\,t_1 \cdots t_n$ $(n \geq 0)$ with $x \in \mathit{Dom}\,\Gamma$ then:

$$\begin{aligned} [\![\Gamma \vdash x\,t_1 \cdots t_n : \tau]\!](x) &= \{\emptyset\} \\ [\![\Gamma \vdash x\,t_1 \cdots t_n : \tau]\!](y) &= \{M \cup \{(x,i)\} \mid 1 \leq i \leq n, M \in [\![\Gamma \vdash t_i : \tau_i]\!](y)\} \\ &\quad \text{if } x \neq y \end{aligned}$$

If t has the form $f\,t_1 \cdots t_n$ with $f \in \mathit{Dom}\,\Sigma$ then

$$[\![\Gamma \vdash f\,t_1 \cdots t_n : \tau]\!](y) = \bigcup_{1 \leq i \leq n} [\![\Gamma \vdash t_i : \tau_i]\!](y)$$

If t is an abstraction $\lambda x\colon \tau.\, u$ then

$$\begin{aligned} [\![\Gamma \vdash \lambda x\colon \tau.\, u : \tau \rightarrow \sigma]\!](x) &= \emptyset \\ [\![\Gamma \vdash \lambda x\colon \tau.\, u : \tau \rightarrow \sigma]\!](y) &= \{M \setminus (\{x\} \times \mathcal{N}) \mid M \in [\![\Gamma, x\colon \tau \vdash u : \sigma]\!](y)\} \\ &\quad \text{if } x \neq y \end{aligned}$$

The subtraction of $\{x\} \times \mathcal{N}$ (for the abstraction) is only necessary if the type of x is second-order. This situation can only occur on outermost level, and it cannot in our second-order rules. One could argue whether these terms exist in $\lambda_2^{\rightarrow}$, but they do indeed in a PTS-like formalisation.

The interpretation can be explained as follows: if $[\![\Gamma \vdash t : \tau]\!](x) = M$ then M contains for each free occurrence of x in t the set of argument positions in variable applications that lie above that occurrence. In particular: if $M = \emptyset$ then x is not free in t and if $\emptyset \in M$ then there is a topmost occurrence of x in t and all variables free in $\theta(x)$ will be free in t^θ as well.

Definition 11. A substitution $\theta : \Gamma \to \Delta$ *preserves* a set $M \subseteq (\mathcal{V} \times \mathcal{N})$, written $\theta \models M$, iff

$$\forall (x,i) \in M.\, x \in \mathit{Dom}\,\Gamma \Rightarrow \exists y_1, \ldots, y_i, t.$$
$$\theta(x) =_{\beta\eta} \lambda y_1, \ldots, y_i.\, t \wedge y_i \in \mathrm{FV}_\beta(t)$$

We can read the property $\exists M \in [\![\Gamma \vdash t : \tau]\!](x).\, \theta \models M$ as follows: "there is a free occurrence of x in t which is not erased when we apply θ to t".

Definition 12. A substitution $\theta : \Gamma \to \Delta$ is called *first-order* iff for all $(x : \sigma) \in \Gamma$ the preterm $\theta(x)$ is *not* an abstraction.

Thus, if t is a normal form and θ is a first-order substitution with only normal forms in its codomain then t^θ is in normal form too. Obviously, a first-order substitution preserves any M. This means that $\exists M \in [\![\Gamma \vdash t : \tau]\!](x).\, \theta \models M$ is equivalent to $x \in \mathrm{FV}_\beta(t)$ for first-order θ.

Lemma 13. *Let $\Gamma \vdash t : \tau$, $\theta : \Gamma \to \Delta$ and $y \in \mathit{Dom}\,\Delta$ be arbitrary. We have* $y \in \mathrm{FV}_\beta(t^\theta) \iff$
$\exists x \in \mathit{Dom}\,\Gamma.\, y \in \mathrm{FV}_\beta(\theta(x)) \wedge \exists M \in [\![\Gamma \vdash t : \tau]\!](x).\, \theta \models M$

Proof. We can w.l.o.g. assume that t is in normal form and that θ maps variables to terms in normal form.

First we prove the lemma for first-order substitutions. Using our assumptions about t and θ and the above observations about first-order substitutions, the lemma reduces to $y \in \mathrm{FV}(t^\theta) \iff \exists x \in \mathit{Dom}\,\Gamma.\, y \in \mathrm{FV}(\theta(x)) \wedge x \in \mathrm{FV}(t)$ which is an obvious property of substitutions.

Now let θ be arbitrary. We prove the lemma by induction on the term structure. We just show "$\Rightarrow$", "$\Leftarrow$" is similar. We only have to consider variable applications $z\, t_1 \cdots t_n$, constant applications $f\, t_1 \cdots t_n$ and abstractions $\lambda z.\, t'$.

Let t be a variable application $z\, t_1 \cdots t_n$. Let $\theta(z) = \lambda y_1 \cdots y_n.\, u$. Then $t^\theta{\downarrow} = u^v$ where $v : E \to \Delta$ is given by $v(y_i) = t_i^\theta{\downarrow}$ and $v(v) = v$ for $v \notin \{y_1, \ldots, y_n\}$. Observe that v is first-order, i.e. we can apply the lemma to it. We get: $y \in \mathrm{FV}_\beta(t^\theta) \iff y \in \mathrm{FV}_\beta(u^v) \iff \exists x' \in \mathit{Dom}\,E.(y \in \mathrm{FV}_\beta(v(x')) \wedge \exists M \in [\![E \vdash u : \tau]\!](x').\, v \models M) \iff \exists x' \in \mathit{Dom}\,E.\, y \in \mathrm{FV}_\beta(v(x')) \wedge x' \in \mathrm{FV}_\beta(u)$. For $x' \in \mathit{Dom}\,E$, we either have $x' = y_i$ for some y_i or $x' \in \mathit{Dom}\,\Delta$.

In the former case the condition reduces to $y_i \in \mathrm{FV}_\beta(u) \wedge y \in \mathrm{FV}_\beta(t_i^\theta)$ for some i. The first part means that θ preserves $M \cup \{(z,i)\}$ iff it preserves M. For the second we can apply the induction hypothesis and get a variable $x \in \mathit{Dom}\,\Gamma$ and a set $M_i \in [\![\Gamma \vdash t_i : \sigma_i]\!](x)$ with $\theta \models M_i$. Thus $\theta \models M_i' = M_i \cup \{(z,i)\}$ and clearly $M_i' \in [\![\Gamma \vdash z\,t_1 \cdots t_n : \tau]\!](x)$.

In the latter case, $x' \in \mathit{Dom}\,\Delta$, we have $y = x'$ and can choose $x = z$: since $\emptyset \in [\![\Gamma \vdash z\,t_1 \cdots t_n : \tau]\!](z)$ we only have to show $\theta \models \emptyset$, but this is trivially true.

For constant applications $f\,t_1 \cdots t_n$ we can directly apply the induction hypothesis: $y \in \mathrm{FV}_\beta(f\,t_1^\theta \cdots t_n^\theta) \iff \exists i.\, y \in \mathrm{FV}_\beta(t_i^\theta) \iff \exists x \in \mathit{Dom}\,\Gamma.\, y \in \mathrm{FV}_\beta(\theta(x)) \wedge \exists M \in [\![\Gamma \vdash t_i : \sigma_i]\!](x).\, \theta \models M \iff \exists x \in \mathit{Dom}\,\Gamma.\, y \in \mathrm{FV}_\beta(\theta(x)) \wedge \exists M \in [\![\Gamma \vdash f\,t_1 \cdots t_n : \tau]\!](x).\, \theta \models M$.

Finally, let t be an abstraction $\lambda z.u$. We define $\theta' = \theta[z \mapsto z]$. We get: $y \in \mathrm{FV}_\beta(t^\theta) \iff y \in \mathrm{FV}_\beta(u^{\theta'}) \wedge y \neq z \iff y \neq z \wedge \exists x \in \mathit{Dom}\,\Gamma \cup \{z\}.\, y \in \mathrm{FV}_\beta(\theta'(x)) \wedge \exists M \in [\![\Gamma, z : \sigma \vdash u : \sigma']\!](x).\, \theta' \models M \iff y \neq z \wedge \exists x \in \mathit{Dom}\,\Gamma.\, y \in \mathrm{FV}_\beta(\theta(x)) \wedge \exists M \in [\![\Gamma, z : \sigma \vdash u : \sigma']\!](x).\, \theta' \models M \iff y \neq z \wedge \exists x \in \mathit{Dom}\,\Gamma.\, y \in \mathrm{FV}_\beta(\theta(x)) \wedge \exists M \in [\![\Gamma, z : \sigma \vdash u : \sigma']\!](x).\, \theta \models M \setminus (\{z\} \times \mathcal{N}) \iff \exists x \in \mathit{Dom}\,\Gamma.\, y \in \mathrm{FV}_\beta(\theta(x)) \wedge \exists M \in [\![\Gamma \vdash \lambda x\colon \sigma.\, u : \sigma']\!](x).\, \theta \models M$. □

Lemma 14. $\Gamma \vdash t \succeq u : \tau \iff [\![\Gamma \vdash t : \tau]\!] \geq [\![\Gamma \vdash u : \tau]\!]$

Proof. This follows easily from a pointwise extension of lemma 13. Considering the "$\Rightarrow$" direction, notice that for each $N \in [\![\Gamma \vdash u : \tau]\!](x)$ we can construct a substitution θ such that $\theta \models M$ iff $M \subseteq N$ and $y \in \mathrm{FV}_\beta(\theta(x))$. □

Clearly, $[\![\_]\!]$ is a total computable function and so is the order $\geq$ when restricted to total computable functions. Therefore:

Theorem 15. *The variable containment problem for $\lambda_2^\rightarrow$ is decidable.*

4 Variable Containment for $\boldsymbol{\lambda^\rightarrow}$

We are going to reduce the general variable containment problem for $\lambda^\rightarrow$ to a more specific situation, in which we only consider a particular signature and substitutions into a particular context. This reduction also links the problem to a problem in the semantics of PCF.

Definition 16. A *pseudo-constant* in a $\lambda^\rightarrow$-signature Σ is a term c with $\langle\rangle \vdash_\Sigma c : \sigma$ for some type σ and:

$$\forall \Gamma.\, \forall \sigma.\, \forall t_1, \ldots, t_n.\, \Gamma \vdash_\Sigma c\,t_1 \cdots t_n : \sigma \Longrightarrow \\ \forall x \in \mathit{Dom}\,\Gamma.\, (x \in \mathrm{FV}_\beta(c\,t_1 \cdots t_n) \iff \exists i.\, x \in \mathrm{FV}_\beta(t_i))$$

Any symbol in the signature is obviously a pseudo-constant. The identity function $\lambda x\colon \sigma.\, x$ is a pseudo-constant if and only if σ is a type constant. The idea behind pseudo-constants is that they behave like constants in many ways, in particular with respect to the variable containment problem. It is sometimes useful to assume a constant for any type (for freezing variables), but this would require an infinite signature. For our purposes it is sufficient to have pseudo-constants for any type.

Definition 17. A $\lambda^{\rightarrow}$-signature Σ is called *rich* if (i) it includes a base type 0, (ii) there are constants $A : 0$ and $B : 0 \rightarrow 0 \rightarrow 0$ in Σ and (iii) for any other base type $\alpha \in \Sigma$ there are constants $C_\alpha : 0 \rightarrow \alpha$ and $D_\alpha : \alpha \rightarrow 0$ in Σ.

We can extend any signature to a rich signature just by adding the missing constants. One could also view signatures as rich if they have *pseudo*-constants of the required types, but we shall not do that as it only complicates the technicalities without adding anything substantial. In the following, we assume for simplicity that there is only one base type 0 in Σ. The corresponding adjustments to the general case are straightforward.

Definition 18. Let Σ be rich. For any type σ we define a term con_σ as follows:

$$\begin{aligned}
\mathrm{con}_0 &= A\\
\mathrm{con}_{0\rightarrow 0} &= \lambda x\colon 0.\, x\\
\mathrm{con}_{0\rightarrow(\sigma\rightarrow\tau)} &= \lambda x\colon 0.\, \lambda y\colon \sigma.\, \mathrm{con}_{0\rightarrow\tau}(\mathrm{B\ x\ }(\mathrm{con}_{\sigma\rightarrow 0}\ \mathrm{y}))\\
\mathrm{con}_{(\sigma\rightarrow\tau)\rightarrow\upsilon} &= \lambda f\colon \sigma\rightarrow\tau.\, \mathrm{con}_{\tau\rightarrow\upsilon}\ (\mathrm{f\ con}_\sigma)
\end{aligned}$$

The function con is well-defined as the right-hand sides of the equations use "fewer arrows" in the types of con than the corresponding left-hand sides. Clearly, each con_σ has type σ in the empty context.

Remark: it is worth noting that the terms con_σ have a more general significance, e.g. they show up in [19] where A is 0 and B is addition. As explained in [9], the map $\mathrm{con}_{\sigma\rightarrow 0}$ is the inverse of $\mathrm{con}_{0\rightarrow\sigma}$ whenever A and B form a monoid; moreover, in the terminology of category theory [13], they are even morphisms of (some) actions of this monoid.

Proposition 19. *Each* con_σ *is a pseudo-constant.*

One consequence of having pseudo-constants for all types is that we can slightly simplify the variable containment problem.

Lemma 20. *The variable containment problem* $\Gamma \vdash_\Sigma t \succeq u : \tau$ *is equivalent to the following property for a rich extension* Σ' *of* Σ*:*

$$\forall\theta : \Gamma \rightarrow X.\, \mathrm{FV}_\beta(t^\theta) \supseteq \mathrm{FV}_\beta(u^\theta) \quad (*)$$

where X *is the fixed context* $\langle x : 0\rangle$*.*

Proof. We prove both implications by contradiction, first ($\Rightarrow$). The property $(*)$ is an instance of the variable containment problem if Σ is already rich. Otherwise, let $\theta : \Gamma \rightarrow X$ be a Σ'-substitution such that $x \in \mathrm{FV}_\beta(u^\theta)$ and $x \notin \mathrm{FV}_\beta(t^\theta)$. We can create a counter-example for variable containment as follows: the context is $\Delta = \langle a : 0, b : 0 \rightarrow 0 \rightarrow 0, x' : 0\rangle$ and the substitution $\phi : \Gamma \rightarrow \Delta$ is given by $\phi(y) = \theta(y)[a/A, b/B, x'/x]$. Clearly, u^ϕ contains the variable x' essentially free whilst t^ϕ does not.

Now ($\Leftarrow$): suppose variable containment does not hold, i.e. for some context Δ, some variable $y \in \mathit{Dom}\ \Delta$, and some substitution $\theta : \Gamma \rightarrow \Delta$ we have that

$y \in \mathrm{FV}_\beta(u^\theta)$ but $y \notin \mathrm{FV}_\beta(t^\theta)$. We can define a substitution $\phi : \Delta \to X$ as follows:

$$\begin{aligned} \phi(z) &= \quad\ \ \mathrm{con}_\sigma \text{ if } z \neq y,\ z\colon \sigma \in \Delta \\ \phi(y) &= (\mathrm{con}_{0\to\sigma}\ \mathrm{x}) \text{ if } y\colon \sigma \in \Delta \end{aligned}$$

and from this we get a contradiction of $(*)$ using the substitution $\phi \circ \theta : \Gamma \to X$: the pseudo-constant property of all con_σ makes sure that x is in $\mathrm{FV}_\beta(u^{\phi\circ\theta})$ while $\mathrm{FV}_\beta(t^{\phi\circ\theta}) = \emptyset$. □

Variable containment is unaffected by replacing constants by pseudoconstants. Based on this observation and lemma 20 we can design a semantic interpretation for types, terms, and judgements to model variable containment. Since X has only one variable x of type 0, $\mathrm{FV}(t)$ is just a boolean information for any t with $X \vdash_\Sigma t : 0$. For higher types, we also have to model how the freeness of x can be affected by β-reduction.

Thus we can interpret $0 : * \in \Sigma$ by the partially ordered set $\{\bot, \top\}$ (with $\bot \leq \top$) and each function space $\sigma \to \tau$ by the set of λ-definable monotonic functions from $[\![\sigma]\!]$ to $[\![\tau]\!]$, ordered pointwise. Here, we take the constants $\bot$ and $\top$ and the function $\wedge \in [\![0 \to 0 \to 0]\!]$ (greatest lower bound) as primitively λ-definable. Thus, our semantic domain is a fully abstract model for PCF_1, i.e. finitary PCF over the unit type. We come to that later in more detail.

The restriction of the function space to λ-definable functions is crucial: the terms $f\,x\,x$ and $B\,(f\,A\,x)\,(f\,x\,A)$ are equivalent w.r.t. to variable containment but are different in the full Poset model over $[\![0]\!]$.

Definition 21. Given a context Γ an *environment* ρ for Γ is a finite map from the domain of Γ to the union of all $[\![\sigma]\!]$ (with $\langle\rangle \vdash_\Sigma \sigma : *$) such that: $\forall x : \sigma \in \Gamma.\, \rho(x) \in [\![\sigma]\!]$.

Let Σ be rich (otherwise we can make it rich by a signature extension). Given any context Γ and environment ρ for Γ, we can interpret judgements $\Gamma \vdash_\Sigma t : \tau$ as follows.

$$\begin{aligned} [\![\Gamma \vdash_\Sigma \lambda x\colon \sigma.\, t : \sigma \to \tau]\!]_\rho &= (v \mapsto [\![\Gamma, x\colon \sigma \vdash_\Sigma t : \tau]\!]_{\rho[x\mapsto v]}) \\ [\![\Gamma \vdash_\Sigma t\, u : \tau]\!]_\rho &= [\![\Gamma \vdash_\Sigma t : \sigma \to \tau]\!]_\rho([\![\Gamma \vdash_\Sigma u : \sigma]\!]_\rho) \\ [\![\Gamma \vdash_\Sigma x : \sigma]\!]_\rho &= \rho(x) \\ [\![\Gamma \vdash_\Sigma c : 0]\!]_\rho &= \top \\ [\![\Gamma \vdash_\Sigma c : 0 \to 0 \to 0]\!]_\rho &= (x \mapsto (y \mapsto x \wedge y)) \\ [\![\Gamma \vdash_\Sigma c : \sigma]\!]_\rho &= [\![\langle\rangle \vdash_\Sigma \mathrm{con}_\sigma : \sigma]\!]_{[]} \text{ if } \sigma \notin \{0, 0 \to 0 \to 0\} \end{aligned}$$

The reason for the special treatment of types 0 and $0 \to 0 \to 0$ is that con_σ terms can contain constants of only these two types, so this stops the recursion. The definition of the interpretation function $[\![\_]\!]$ is well-defined as the interpretation of each judgement $[\![\Gamma \vdash_\Sigma t : \sigma]\!]$ is in $[\![\sigma]\!]$. Moreover, for any given environment ρ, the function $[\![\_]\!]_\rho$ is clearly recursive.

The interpretation of judgements is in fact independent from the choice of signature, as all constants of the same type have equal interpretations. The idea behind this interpretation is the following: we use the fixed context $X = \langle x : 0 \rangle$ and take $\top$ for "x is not essentially free" and $\bot$ for "x is essentially free". Apparently, x does not occur free in any constant c of type 0, which we model by interpreting c as $\top$. Then, x is essentially free in $B\ t\ u$ iff it is essentially free in either t or u — this explains the interpretation of B (and any other constant of type $0 \to 0 \to 0$) as $\wedge$, the greatest lower bound. The rest of the definition is just book-keeping and reducing more complicated situations to simpler ones. In particular, $\beta\eta$-equivalent terms have equal interpretations as syntactic abstraction and application are modelled by semantic abstraction and application, and constants of any type can be replaced by pseudo-constants of the same type as they behave the same w.r.t. the variable containment problem.

As usual, we can compose substitutions and environments.

Definition 22. Let $\theta : \Gamma \to \Delta$ be a substitution and ρ be an environment for Δ. We define a function $\rho \circ \theta$ as follows:

$$(\rho \circ \theta)(x) = [\![\Delta \vdash \theta(x) : \tau]\!]_\rho \quad \text{if } \Gamma \vdash x : \tau$$

Lemma 23. *Let $\theta : \Gamma \to \Delta$ be a substitution and ρ be an environment for Δ.*

1. *$\rho \circ \theta$ is an environment for Γ.*
2. *For all $\Gamma \vdash t : \tau$ we have $[\![\Delta \vdash t^\theta : \tau]\!]_\rho = [\![\Gamma \vdash t : \tau]\!]_{\rho\circ\theta}$.*

Lemma 23 is standard for semantic interpretations of the λ-calculus, the proof is routine and needs hardly any adaptation from (for example) the proof of lemma 24 in [18].

Definition 24. We define an order $\leq$ on judgements of the same type and context as follows:

$$(\Gamma \vdash t : \tau) \leq (\Gamma \vdash u : \tau) \iff \forall \rho.\, [\![\Gamma \vdash t : \tau]\!]_\rho \leq_\tau [\![\Gamma \vdash u : \tau]\!]_\rho$$

Lemma 25. *Let J_1, J_2 be judgements $J_i = \Gamma \vdash_\Sigma t_i : \tau$.*
We have $J_1 \leq J_2 \iff \Gamma \vdash t_1 \succeq t_2 : \tau$.

Proof. By lemma 20 we can w.l.o.g. assume that Σ is rich and restrict our attention to variable containment w.r.t. the context X. Similarly we can require Σ to be the rich extension of the empty signature, because variable containment and $[\![\_]\!]$ are unaffected by replacing constants by arbitrary pseudo-constants. Since the interpretation of syntactic abstraction and application is by semantic abstraction and application, β-reduction does not affect the interpretation. From this it follows (by a straightforward induction on normal forms) that $[\![X \vdash u : 0]\!]_{x\mapsto\bot} = \bot \iff x \in \mathrm{FV}_\beta(u)$.

To prove ($\Leftarrow$) we need to be able to construct a substitution counterexample for $\Gamma \vdash t_1 \succeq t_2 : \tau$ whenever we have an environment ρ such that $\neg [\![J_1]\!]_\rho \leq [\![J_2]\!]_\rho$. Because we required each value in the model to be λ-definable relative to $\bot$, $\top$, and $\wedge$, we can find for each value v in $[\![\sigma]\!]$ a term t_v such that $X \vdash t_v : \sigma$ and $[\![X \vdash t_v : \sigma]\!]_{x\mapsto\bot} = v$. The substitution $\theta : \Gamma \to X$ with $\theta(y) = t_{\rho(y)}$ is then the substitution we were looking for. □

In other words, the variable containment problem is equivalent to deciding the inequality $\leq$ in a fully abstract model of PCF_1 (PCF over the unit type with constants $\perp$ and $\top$ and $\wedge$). To decide $\leq$, it would be sufficient to effectively construct such a model, because each type is interpreted by a finite poset. The connection is rather tight indeed: if we have a partial construction of the model for types up to order n then we can decide variable containment for $\lambda_n^{\rightarrow}$.

A recent result by Zaionc [21] means[3] that variable containment is decidable for $\lambda_3^{\rightarrow}$, as his technique of creating all λ-definable values by some grammar easily extends to the situation with predefined constants A and B. Sieber's PCF model of "logically sequential" elements [20] seems to be effective for finitary PCF and it is fully abstract up to order 4 and term-generated up to order 3; this also implies the decidability of variable containment of $\lambda_3^{\rightarrow}$, though the connection is less direct than in the case of Zaionc's result. This improves upon my theorem 15; but the decision procedures obtained that way are extremely inefficient and of solely theoretical interest, while the decision procedure outlined earlier for $\lambda_2^{\rightarrow}$ is of polynomial complexity.

For PCF_2 (PCF over the booleans with constants $\perp$, `tt`, `ff`, `if`), effectively constructing a fully abstract model was posed as an open problem by Jung and Stoughton in [8]; it is yet unclear whether this is equivalent to our problem.

We can also show that the "effective" construction of a model for PCF_1 is *necessary* to decide $\leq$ and even the indistinguishability relation $\approx$.

Theorem 26. *The problem of deciding the indistinguishability relation $\approx$ for* PCF_1 *is equivalent to effectively constructing a fully abstract model.*

Proof. As explained before, one implication is trivial. It remains to show that $\approx$ gives us a way of constructing a fully abstract model.

We can construct $[\![0]\!] = \{\perp, \top\}$ with $\perp \leq \top$. Suppose we have constructed the sets $[\![\sigma_i]\!]$ then we can construct $[\![\sigma_1 \to \cdots \to \sigma_n \to 0]\!]$ as follows. Each element in this set is a function mapping n-tuples to either $\perp$ or $\top$. There are only finitely many such functions (as all the $[\![\sigma_i]\!]$ are finite).

To decide whether a particular function F is λ-definable we consider the term $\chi_F = \lambda f.\, f\, a_{11} \cdots a_{1n} \wedge \cdots \wedge f\, a_{k1} \cdots a_{kn}$ where the tuples $a_{i1} \cdots a_{in}$ are tuples of terms representing exactly those tuples (of values) mapped by F to $\top$. Since the construction of each $[\![\sigma_j]\!]$ is assumed to be complete we can effectively find a term a for each value v in these sets.

Now take $\succeq$ to be the pointwise extension of the $\geq_j$ such that it is defined on *all* monotonic functions, not just the λ-definable ones. Now consider any other function $G \succ F$ and its characteristic function χ_G. Suppose F is defined by a term t then $[\![\chi_G\, t]\!] = \perp$ and $[\![\chi_F\, t]\!] = \top$. Consequently, χ_F and χ_G are distinguishable if F is λ-definable, and thus if $\chi_G \approx \chi_F$ for any $G \succ F$ then F cannot be λ-definable. Now suppose that χ_F is distinguishable from χ_G for each $G \succ F$. This means that there has to be a term t_G for each G such that $[\![\chi_F\, t_G]\!] = \top$ and $[\![\chi_G\, t_G]\!] = \perp$. We can define a term $t = \lambda x_1 \cdots x_n . t_{G_1}\, x_1 \cdots x_n \wedge$

[3] The title of Zaionc's paper is a little misleading — he gives the base type order 0 instead of 1, following a deplorable custom in programming language semantics.

$\cdots \wedge t_{G_r}\, x_1 \cdots x_n$ where $G_1 \cdots G_r$ are all functions greater than F. We obviously have $[\![\chi_F\, t]\!] = \top$ and $[\![\chi_{G_i}\, t]\!] = \bot$ for all G_i. But this exactly means $[\![t]\!] = F$, i.e. F is λ-definable.

Hence we can construct $[\![\sigma_1 \to \cdots \to \sigma_n \to 0]\!]$ as the set of all monotonic functions that pass the outlined test, i.e. whose characteristic functions are distinguishable. $\square$

Unsurprisingly a similar result holds for PCF_2, though the proof is a bit messier, involving pairs of characteristic functions (one for `tt`, one for `ff`). We do not go into that.

5 Conclusion and Open Problems

We have explained why the usual condition $\mathrm{FV}(l) \supseteq \mathrm{FV}(r)$ for higher-order rewrite rules $l \to r$ is inadequate and why it should be replaced by the "variable containment" property $l \succeq r$, the rewrite interior of $\mathrm{FV}_\beta(l) \supseteq \mathrm{FV}_\beta(r)$.

We have shown that variable containment is decidable for the third-order fragment of $\lambda^\to$, also giving a constructive solution for the second-order fragment. The general problem for $\lambda^\to$ is equivalent to effectively constructing a fully abstract model for finitary PCF over the unit type.

Open problems are:

- Is the problem for $\lambda^\to$ decidable? I have seen a preliminary version of an unpublished paper which claims that it is indeed. The proof in the paper is rather complicated and without thorough revision I would not say that the problem is settled.
- Is variable containment equivalent to providing a fully abstract model for PCF_2? This is very delicate. I had a promising proof idea which I pursued for a few weeks without getting it to work. One of the referees conjectured that the PCF_2 model is not recursive.
- For which type systems is variable containment undecidable?
- Finally: what about other type systems of the λ-cube, is there a similar correspondence between full abstraction and variable containment for those systems? Probably yes, but to make any sense of this, one first has to generalise the definition of full abstraction to these type systems.

Acknowledgments

Many thanks to Alex Simpson with whom I had a number of fruitful discussions on the subject, especially on the PCF part and theorem 26. Also thanks to the HOA referees.

References

1. Hendrik P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics.* North-Holland, 1984.

2. Hendrik P. Barendregt. Introduction to generalised type systems. *Journal of Functional Programming*, 1(2):124–154, 1991.
3. Hendrik P. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science, Vol.2*, pages 117–309. Oxford Science Publications, 1992.
4. Gilles Dowek. Third order matching is decidable. In *Proceedings of the 7th Symposium on Logic in Computer Science*, pages 2–10, 1992.
5. Philippa Gardner. *Representing Logics in Type Theory.* PhD thesis, University of Edinburgh, 1992.
6. W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
7. Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.
8. Achim Jung and Allen Stoughton. Studying the fully abstract model of PCF within its continuous function model. In *Typed Lambda Calculi and Applications*, 1993. LNCS 664.
9. Stefan Kahrs. Towards a domain theory for termination proofs. In *Rewriting Techniques and Applications*, pages 241–255, 1995. LNCS 914.
10. Jan Willem Klop. *Combinatory Reduction Systems.* PhD thesis, Centrum voor Wiskunde en Informatica, 1980.
11. Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.
12. Ralph Loader. The undecidability of λ-definability, 1994.
13. Saunders MacLane. *Categories for the Working Mathematician.* Springer, 1971.
14. Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. Technical Report TUM-I94333, Technische Universität München, 1994.
15. Aart Middeldorp. Modular aspects of properties of term rewriting systems related to normal forms. In *Rewriting Techniques and Applications*, pages 263–277, 1989. LNCS 355.
16. Tobias Nipkow. Higher order critical pairs. In *Proceedings of the 6th Symposium on Logic in Computer Science*, pages 342–349, 1991.
17. Vincent Padovani. On equivalence classes of interpolation equations. In *Typed Lambda Calculi and Applications*, pages 335–249, 1995. LNCS 902.
18. Jaco van de Pol. Termination proofs for higher-order rewrite systems. In *Higher-Order Algebra, Logic, and Term Rewriting*, pages 305–325, 1993. LNCS 816.
19. Jaco van de Pol and Helmut Schwichtenberg. Strict functionals for termination proofs. In *Typed Lambda Calculi and Applications*, pages 350–364, 1995. LNCS 902.
20. Kurt Sieber. Reasoning about sequential functions via logical relations. In M.P. Fourman, P.T. Johnstone, and A.M. Pitts, editors, *Applications of Categories in Computer Science*, pages 258–269. Cambridge University Press, 1992.
21. Marek Zaionc. Lambda definability is decidable for second order types and for regular third order types. Unpublished Manuscript, University of New York at Buffalo, 1995.

This article was processed using the LaTeX macro package with LLNCS style