Information and Computation **179**, 1–18 (2002)
doi:10.1006/inco.2001.2950

# The Subtyping Problem for Second-Order Types Is Undecidable

Jerzy Tiuryn[1] and Paweł Urzyczyn[2]

*Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warsaw, Poland*
E-mail: tiuryn@mimuw.edu.pl, urzy@mimuw.edu.pl

Received January 15, 1997; published online May 16, 2002

We prove that the subtyping problem induced by Mitchell's containment relation for second-order polymorphic types is undecidable. It follows that type-checking is undecidable for the polymorphic lambda-calculus extended by an appropriate subsumption rule. © 2002 Elsevier Science (USA)

## 1. INTRODUCTION

Among various forms of subtyping, one that seems to be quite fundamental for the analysis of polymorphic languages is induced by the containment relation of Mitchell [6]. An expression of a universal type, say $\forall\alpha\sigma$, can be understood as having all types that are polymorphic instances of $\forall\alpha\,\sigma$. If we choose the second-order polymorphic lambda-calculus as the basic language, then (taking into account the possibility of polymorphic generalization) we may formalize the notion of containment as

$$\forall\alpha_1\cdots\forall\alpha_n\,\sigma \preceq \forall\beta_1\cdots\forall\beta_m\,\sigma(\rho_1/\alpha_1,\ldots,\rho_n/\alpha_n), \tag{1}$$

where $\beta_1,\ldots,\beta_m$ are not free in $\forall\alpha_1\cdots\forall\alpha_n\,\sigma$. The containment relation $\preceq$ is a quasi-order (it is reflexive and transitive) and the subsumption rule for $\preceq$

$$\frac{E \vdash M : \tau,\ \tau \preceq \sigma}{E \vdash M : \sigma}$$

is admissible for type inference in (Curry style) second-order lambda-calculus (system **F**). However, it lacks compositionality: no conclusion of the form $\sigma \preceq \tau$ can be derived from the $\preceq$-related properties of the components of $\sigma$ and $\tau$. For instance, one cannot have e.g. $\tau \to \sigma \preceq \tau \to \sigma'$ even if $\sigma \preceq \sigma'$. Thus, one would like to postulate that $\preceq$ be extended to a subtyping relation $\sqsubseteq$, so that arrow is monotonic with respect to $\sqsubseteq$ in the second argument and antimonotonic in the first argument That is,

$$\sigma_1 \to \tau_1 \sqsubseteq \sigma_2 \to \tau_2, \quad \text{whenever } \sigma_2 \sqsubseteq \sigma_1 \ \text{ and } \ \tau_1 \sqsubseteq \tau_2. \tag{2}$$

Of course, the extended relation should be monotonic with respect to quantification:

$$\forall\alpha\,\sigma \sqsubseteq \forall\alpha\,\sigma', \quad \text{whenever } \sigma \sqsubseteq \sigma'. \tag{3}$$

In the presence of (3), instead of (1) in its full generality, it is enough to postulate two special cases:

$$\forall\alpha\,\sigma \sqsubseteq \sigma(\rho/\alpha), \tag{4}$$

and

$$\sigma \sqsubseteq \forall\alpha\,\sigma, \quad \text{whenever } \alpha \text{ does not occur free in } \sigma. \tag{5}$$

[1] The first author is partly supported by NSF Grants CCR-9417382 and CCR-9304144, by ESPRIT BRA7232 GENTZEN, and by KBN Grant 2 P301 031 06.

[2] The second author is partly supported by NSF Grant CCR-9417382, by ESPRIT BRA7232 GENTZEN, and by KBN Grant 2 P301 031 06.

Another postulate, which has an obvious logical counterpart (quantifier distributivity over arrow) can be, in a weak form, written as follows:

$$\forall\alpha(\sigma \to \tau) \sqsubseteq \sigma \to \forall\alpha\, \tau, \quad \text{provided } \alpha \text{ is not free in } \sigma. \tag{6}$$

As we noted, this is a weak form of distributivity, and we choose it because it seems that its necessity is more appealing to the intuition. However, the general form

$$\forall\alpha(\sigma \to \tau) \sqsubseteq \forall\alpha\, \sigma \to \forall\alpha\, \tau \tag{7}$$

can be derived from the weak one with the help of the other postulates, provided we also require that $\sqsubseteq$ be transitive. This has to be done explicitly, and thus $\sqsubseteq$ is defined as the least quasi-order satisfying all the postulates (2), (4), (5), (3), and (6).

The relation $\sqsubseteq$ is a proper extension of $\preceq$, and the subsumption rule for $\sqsubseteq$

$$\frac{E \vdash M : \tau,\ \tau \sqsubseteq \sigma}{E \vdash M : \sigma} \tag{8}$$

is no longer admissible for system **F**. To see this, observe that, e.g., $x : \alpha \to \forall\beta\,\beta \vdash x : \alpha \to \beta$ is not derivable, while obviously we have $\alpha \to \forall\beta\,\beta \sqsubseteq \alpha \to \beta$. In fact, one can show a stronger property. There are pure lambda terms which are typable in system **F** extended by rule (8), but untypable in system **F** alone. One possible example is

$$\lambda a.\,[\lambda yz.\,a((\lambda v.\,v(\lambda xy.\,yay))y)(zy)(z(\lambda x.\,xxx))]YZ,$$

where $Y$ is $\lambda x.\,K(xx)(xaa)$ and $Z$ is $\lambda u.\,u(\lambda xy.\,a)$. The existence of such terms may also be derived using the "constants-for-free" approach of Wells [10]. This means that subtyping provides for a non-trivial extension of system **F**. Let us call this extension $\mathbf{F}_{\sqsubseteq}$.

To avoid possible confusion, let us stress that our system $\mathbf{F}_{\sqsubseteq}$ has very little in common with the system $\mathbf{F}_{\leq}$ of [8]. The latter is based on the notion of *bounded quantification*, i.e., allows for types of the form $(\forall\alpha \leq \sigma)\tau$, and has no counterpart of our condition (1) which is related to type instantiation. In particular, the main result of [8] and our Theorem 2.1 do not imply each other.

A fundamental property of subtyping is that it is, in a sense, equivalent to extensionality. More precisely, let us write $\mathbf{F}_\eta$ for system **F** extended by the following rule:

$$\frac{E \vdash M : \tau,\ M \to_\eta N}{E \vdash N : \tau}.$$

As shown in [6], systems $\mathbf{F}_{\sqsubseteq}$ and $\mathbf{F}_\eta$ derive exactly the same type assignments. That is, $\Gamma \vdash M : \tau$ holds in $\mathbf{F}_{\sqsubseteq}$ iff it holds in $\mathbf{F}_\eta$. This fact is equivalent to the following property of subtyping: an inequality $\tau \sqsubseteq \sigma$ is true if and only if there exists a *coercion* ("retyping function" in the terminology of [6]) from $\tau$ to $\sigma$, i.e., a lambda term $C$, which can be assigned type $\tau \to \sigma$ in system **F** and which is $\eta$-reducible to identity (has no computational meaning).

Subtyping is also an important notion from the point of view of semantics. The completeness theorem (see [6]) states that $\sigma \sqsubseteq \tau$ holds iff $\sigma$ is a subtype, i.e., a sub-PER of $\tau$ in all PER models. To be precise, this completeness result is stated only implicitly in [6], not even using the name PER. Theorem 7 of [6] states completeness for all the so-called simple inference models. On the other hand, a simple quotient-set model defined in Section 4.4 is just a PER model. The last paragraph of Section 4.4 explains that every simple inference model can be made into a simple quotient-set model. Therefore Theorem 7 can be seen as stating completeness of the containment relation in all PER models.

As we said above, system $\mathbf{F}_{\sqsubseteq}$, or equivalently system $\mathbf{F}_\eta$, is a proper extension of **F**. Although the type reconstruction problem

*Given a term $M$, does there exist $E$ and $\tau$ such that $E \vdash M : \tau$?*

for $\mathbf{F}$ is undecidable [10], the proof does not apply to $\mathbf{F}_{\sqsubseteq}$. Thus, decidability of type reconstruction remained an open question for our extended system. A recent work of Jim [4] shows that type reconstruction for $\mathbf{F}_{\sqsubseteq}$ reduces to the *subtyping problem:*

*Given $\sigma$ and $\tau$; decide whether $\sigma \sqsubseteq \tau$ holds.*

Unfortunately, Jim's reduction does not help much. Our main Theorem 2.1 states that $\sqsubseteq$ is undecidable. In addition, one can easily see that $\sigma \sqsubseteq \tau$ is equivalent to $x : \sigma \vdash x : \tau$ being derivable in $\mathbf{F}_{\sqsubseteq}$. Thus, the subtyping problem reduces to the *type-checking* problem

*Given $M$, $E$ and $\tau$; does $E \vdash M : \tau$ hold?*

It follows that type-checking for $\mathbf{F}_{\sqsubseteq}$ is also undecidable. However, undecidability of subtyping does not directly imply undecidability of type-reconstruction. This fact was shown by Wells [12] very soon after our main result was announced. The work of Wells uses another proof of the undecidability of subtyping [11] which is based on an entirely different approach.

The undecidability of subtyping should be contrasted with a result of [9], which states that the equivalence relation $\sim$ generated by the quasi-order $\sqsubseteq$, called *bicoercibility*, is decidable. (The relation $\sim$ is defined so that $\tau \sim \sigma$ holds iff $\tau \sqsubseteq \sigma$ and $\sigma \sqsubseteq \tau$.) This contrast can be explained as follows: our subtyping relation is close to being antisymmetric; i.e., elements of one equivalence class differ very little. Indeed, the characterization of [9] implies that bicoercible types are of identical shapes, when seen as binary trees, and no substantial instantiation steps are needed to obtain one from the other (see the axioms (B1)–(B3) in Section 3 and the remark following Proposition 3.1).

One more remark is in order here. The condition (6) is not essential for our undecidability result, because our reduction also applies to the appropriate restriction of the subtyping relation [2]. On the other hand, the proof method of Wells [11], at the cost of applying (6), yields undecidability of $\sqsubseteq$ restricted to cases when instantiations cannot be nested more than twice, while an unbounded nesting is used in our approach. It remains an open question whether there is really a trade-off between these two features.

It should be stressed, however, that undecidability of $\sqsubseteq$ is always obtained as a result of a special kind of interplay between the *instantiation* property of (4) and antimonotonicity of $\to$ in (2). If the instantiation is restricted to *monomorphic* types $\rho$, i.e., types without any quantifiers in them, and the distributivity axiom (6) is removed, then, as shown in [7], the system becomes decidable. Also, it is not difficult to see that the whole system with $\to$ made monotone in both arguments is decidable. Indeed, one can think of our postulates (4), (5), and (6) as rewrite rules oriented from left to right. The subtyping problem now becomes the reachability problem for such a rewriting system. But applying our rules to a type can only increase its size, and thus an exhaustive search over a finite space suffices to decide reachability.

The proof of our main result consists of two steps. The first one is to design a machine-oriented variant of the problem. In Section 5 we define a particular automaton, called a stack-register machine, and we prove in Section 6 that the halting problem for stack-register machines effectively reduces to the subtyping problem. For this we encode machine ID's as subtyping inequalities, and we prove that a machine halts on a given ID iff the corresponding inequality is true.

The next step is to show in Section 7 that the halting problem for stack-register machines is undecidable. For this, we show how to simulate a two-counter automaton with the help of a stack-register machine, and we use the well-known fact that the halting problem for two-counter automata is undecidable.

The organization of the paper is as follows: Section 2 introduces the main definitions. Sections 3 and 4 contain some technical results needed in Section 6. Sections 5 to 7 are devoted to the main proof, as explained above.

## 2. SECOND-ORDER SUBTYPING

We use the following syntax for second-order types

$$\sigma ::= \alpha \mid (\sigma' \to \sigma'') \mid (\forall\alpha\, \sigma).$$

Symbols $\alpha$, $\beta$, $\gamma$, possibly with indices, will be ranging over type variables. Symbols $\sigma$, $\tau$, $\rho$, $\xi$, $\nu$ will be ranging over second-order types. In order to avoid too many parentheses we adopt the convention that arrows associate to the right (i.e., $\sigma \to \tau \to \rho$ means $\sigma \to (\tau \to \rho)$), and that quantifiers have higher priority than arrows (so that $\forall\alpha\, \tau \to \sigma$ means $(\forall\alpha\, \tau) \to \sigma$, rather than $\forall\alpha\, (\tau \to \sigma)$). Types are taken with respect to alpha-conversion; that is, we identify types that differ only in names of their bound variables.

We will write $\sigma(\tau/\alpha)$ to indicate the result of substituting $\tau$ for all free occurrences of $\alpha$ in $\sigma$ — bound variables in $\sigma$ may have to be renamed in order to avoid capture of free variables of $\tau$.

In this paper we will also use the notation $C[\,]$ to denote a *context* with at most a single hole $[\,]$, and we will write $C[\tau]$ for the result of filling the hole with $\tau$, where $\tau$ may be a type or a context. This should be understood as a literal substitution in which some of the free variables of $\tau$ become captured by quantifiers occurring in $C[\,]$ (cf. Chapter 2 in [1]). This is not inconsistent with our alpha conversion convention, because a context is not a type.

It is sometimes useful to view second-order types as binary trees such that:

- inner nodes are labelled $\to$;
- leaves are labelled with type variables;
- every node is labelled by a finite (possibly empty) sequence of quantifiers $\forall\alpha_1\forall\alpha_2\ldots$.

A node in a type $\sigma$ will be identified with a path (a sequence of 0's and 1's) leading from the root to that node. Similarly, we can assign a path of 0's and 1's to a hole $[\,]$ in a context $C[\,]$. The parity of a path $\pi$ is positive if the number of 0's in $\pi$ is even; otherwise the parity of $\pi$ is negative.

We say that $\rho$ occurs as a subtype in $\sigma$ along a path $\pi$ if there is a context $C[\,]$ with a single hole, the node where $[\,]$ occurs is $\pi$, and $\sigma = C[\rho]$. Note that, due to possible occurrences of quantifiers, there may be more than one subtype occurring along the same path.

## 2.1. Mitchell's System

First we present Mitchell's system for polymorphic subtyping (see [6]). The original definition of $\sqsubseteq$ consisted of containment axioms (1) and (7) and containment rules (2), (3) and transitivity. We prefer, however, to split the general axiom (1) into three specific postulates to highlight the three different basic forms of containment: identity, instantiation, and generalization. We also use the weak form of quantifier distribution. It is left to the reader to verify that these two definitions are equivalent. Formally, Mitchell's system derives formulas of the form $\sigma \sqsubseteq \tau$, where $\sigma$ and $\tau$ are polymorphic types.

Axioms:

**(refl)** $\sigma \sqsubseteq \sigma$;

**(inst)** $\forall\alpha\,\sigma \sqsubseteq \sigma(\rho/\alpha)$;

**(dummy)** $\sigma \sqsubseteq \forall\alpha\,\sigma$, ($\alpha$ not free in $\sigma$);

**(distr)** $\forall\alpha\,(\sigma \to \tau) \sqsubseteq \sigma \to \forall\alpha\,\tau$, ($\alpha$ not free in $\sigma$).

Rules:

$$(\to)\ \frac{\sigma' \sqsubseteq \sigma \qquad \tau \sqsubseteq \tau'}{\sigma \to \tau \sqsubseteq \sigma' \to \tau'} \qquad (\forall)\ \frac{\sigma \sqsubseteq \tau}{\forall\alpha\,\sigma \sqsubseteq \forall\alpha.\,\tau}$$

$$(\text{trans})\ \frac{\sigma \sqsubseteq \rho \qquad \rho \sqsubseteq \tau}{\sigma \sqsubseteq \tau}.$$

Our main result is as follows:

THEOREM 2.1. *It is effectively unsolvable to decide, for given types $\sigma$ and $\tau$, whether the formula $\sigma \sqsubseteq \tau$ can be derived in the above system.*

## 2.2. The Longo–Milsted–Soloviev System

The system which we are about to present is due to Longo, Milsted, and Soloviev (see [5]). It will play an auxiliary role in our proof of the main result. The crucial notion of a weight of an inequality, which we introduce in this section, will be used as a measure with respect to which we will carry on the induction argument in the main technical lemma of our proof. This notion is directly related to the system of Longo, Milsted, and Soloviev. Obviously, it can be introduced as well for any other system which derives inequalities, including Mitchell's system of the previous section. The benefit of working with the system of Longo, Milsted, and Soloviev is that it will be easier to control the weight of inequalities in Sections 3 and 4.

The system is designed for deriving sequents of the form $\sigma \vdash \tau$, where $\sigma$ and $\tau$ are polymorphic types.

Axiom: $\alpha \vdash \alpha$

Rules:

$$(\to) \qquad \frac{\sigma' \vdash \sigma \qquad \tau \vdash \tau'}{\sigma \to \tau \vdash \sigma' \to \tau'}$$

$$(\forall\text{-left}) \qquad \frac{\sigma(\xi/\alpha) \vdash \tau}{\forall\alpha\, \sigma \vdash \tau}$$

$$(\forall_n\text{-right}) \qquad \frac{\sigma \vdash \tau_1 \to \cdots \to \tau_n \to \rho}{\sigma \vdash \tau_1 \to \cdots \to \tau_n \to \forall\alpha\, \rho}$$

In the above rule we assume that $n \geq 0$ and $\alpha$ does not occur free in $\sigma, \tau_1, \ldots, \tau_n$. (Originally, the axiom scheme proposed in [5] was "$\sigma \vdash \sigma$," for arbitrary $\sigma$. It is a routine exercise to see that our weaker axiom, which we prefer for technical reasons, is sufficient.)

By $\sigma \vdash_{LMS} \tau$ we denote derivability of the sequent $\sigma \vdash \tau$ in the above system.

Strength of the above system lies in simplicity of its rules. In particular, as it follows from the next result, the rule of transitivity is admissible in it. This can be seen as a form of cut elimination.

THEOREM 2.2 (Longo–Milsted–Soloviev [5]). *For all types $\sigma$ and $\tau$,*

$$\sigma \sqsubseteq \tau \Leftrightarrow \sigma \vdash_{LMS} \tau,$$

Next we introduce the notion of a *weight* of an inequality. We set the weight of (an instance of) the ($\forall$-left) rule

$$(\forall\text{-left}) \quad \frac{\sigma(\xi/\alpha) \vdash \tau}{\forall\alpha\, \sigma \vdash \tau}$$

as the number of $\to$'s in $\xi$ (which we denote $|\xi|$), multiplied by the number of free occurrences of $\alpha$ in $\sigma$, i.e., it is the total number of $\to$'s eliminated by this rule. The weight of a derivation in $\vdash_{LMS}$ is the sum of the weights of all instances of ($\forall$-left) used in this derivation (or zero if ($\forall$-left) does not occur). Finally, the weight of an inequality $\sigma \sqsubseteq \tau$ is the least weight of a derivation which derives $\sigma \vdash \tau$. The weight is assigned only to inequalities which are true.

## 3. INVARIANCE OF THE WEIGHT UNDER BICOERCIONS

Recall that types $\tau$ and $\sigma$ are *bicoercible* (written $\tau \sim \sigma$) iff $\tau \sqsubseteq \sigma$ and $\sigma \sqsubseteq \tau$. The goal of this section is to establish invariance of the weight introduced in Section 2.2 under the bicoercibility relation. We will need a series of lemmas which leads to this result. Throughout this section we denote by $\perp$ the type $\forall\alpha\, \alpha$.

Before we start let us recall the equational axiomatization of $\sim$ from [9]. This is the least congruence (i.e., preserved by $\to$ and $\forall$) which satisfies the following three axioms.

$$\begin{array}{ll} \textbf{(B1)} & \forall\alpha\forall\beta\,\sigma \sim \forall\beta\forall\alpha\,\sigma \\ \textbf{(B2)} & \forall\alpha\,(\sigma \to \tau) \sim (\sigma \to \forall\alpha\,\tau) \quad (\alpha \notin FV(\sigma)) \\ \textbf{(B3)} & \forall\alpha\,\sigma \sim \sigma(\bot/\alpha) \quad \text{(all occurrences of } \alpha \text{ in } \sigma \text{ are positive)} \end{array}$$

We will say that the inequality $\sigma' \sqsubseteq \tau'$ is obtained from $\sigma \sqsubseteq \tau$ by one application of (B1) if $\sigma'$ or $\tau'$ is obtained from $\sigma$ or $\tau$, respectively, by performing one application of (B1) treated as a rewrite rule ordered from left to right. In a similar way we treat (B2) and (B3) as rewrite rules, ordered from left to right.

We start with rearranging derivations. We say that a derivation in $\vdash_{LMS}$ is in *normal form* iff a conclusion of a ($\forall_i$-right) rule is never a premise of ($\forall$-left), nor the right premise of ($\to$).

LEMMA 3.1. *If $\sigma \vdash \tau$ is derivable in $\vdash_{LMS}$, then there is a derivation in a normal form of the same sequent in $\vdash_{LMS}$ with the same weight.*

*Proof.* The result of applying first a ($\forall_i$-right) rule and then a ($\forall$-left) rule is the same as first applying the ($\forall$-left) rule and then applying the ($\forall_i$-right) rule (but not conversely). A similar permutation is possible if a conclusion of ($\forall_i$-right) is the right premise of ($\to$). It may happen that the variable which gets bound by the ($\forall_i$-right) rule occurs free in the left premise of ($\to$). In that case we rename, using a brand new variable, all occurrences of that variable in the sub-derivation whose conclusion is the ($\forall_i$-right) rule. Clearly, such a renaming does not change the weight. We leave the details for the reader. ■

LEMMA 3.2. *Let $\sigma \vdash \rho_1 \to \cdots \to \rho_k \to \forall\beta\,\tau$ be derivable in $\vdash_{LMS}$ and let $\beta$ be not free in $\sigma, \rho_1, \ldots, \rho_k$. Then there is a normal derivation of $\sigma \vdash \rho_1 \to \cdots \to \rho_k \to \tau$ of the same weight.*

*Proof.* An easy induction with respect to derivations. (In fact, every normal derivation of $\sigma \vdash \rho_1 \to \cdots \to \rho_k \to \forall\beta\,\tau$ must end with an application of ($\forall_k$-right) introducing $\forall\beta$. ■

LEMMA 3.3. *If $\sigma' \sqsubseteq \tau'$ is obtained from $\sigma \sqsubseteq \tau$ by one application of* (B1), *then the weight of $\sigma' \sqsubseteq \tau'$ is equal to the weight of $\sigma \sqsubseteq \tau$.*

*Proof.* We proceed by induction on the length of the derivation of $\sigma \vdash \tau$ in $\vdash_{LMS}$. Let us assume that the derivation is in normal form, as described in Lemma 3.1. The case of the axiom and ($\to$) is obvious. Assume now that the last rule is ($\forall$-left). The only interesting case to consider here is when $\sigma$ is of the form $\forall\alpha\forall\beta\,\sigma_1$ and $\sigma'$ is $\forall\beta\forall\alpha\,\sigma_1$. All other cases are handled in a routine way by the induction hypothesis. Since the derivation is in normal form it follows that the last two rules are ($\forall$-left), one which introduced the $\forall\beta$ and another which introduced the $\forall\alpha$ in $\sigma$. Swapping these two rules results in deriving $\sigma' \vdash \tau$. An easy calculation shows that the weight of this derivation remains unchanged.

Let the last rule be ($\forall_i$-right). The only nontrivial case, when $\sigma$ has the form $\rho_1 \to \cdots \to \rho_i \to \forall\alpha\beta\,\tau$, follows easily from Lemma 3.2. This is because we can assume that the preceding step is also an application of ($\forall_i$-right), introducing $\forall\beta$. ■

LEMMA 3.4. *If $\sigma' \sqsubseteq \tau'$ is obtained from $\sigma \sqsubseteq \tau$ by one application of* (B2), *then the weight of $\sigma' \sqsubseteq \tau'$ is equal to the weight of $\sigma \sqsubseteq \tau$.*

*Proof.* First we prove that $\sigma' \sqsubseteq \tau'$ is of weight less than or equal to the weight of $\sigma \sqsubseteq \tau$. Again we proceed by induction on the length of derivation of $\sigma \vdash \tau$ in $\vdash_{LMS}$. Assume that the derivation is in normal form (see Lemma 3.1). We discuss here only the case when the last rule in the derivation is ($\forall$-left). The remaining cases are easier and are left for the reader. Thus the last step of the derivation looks as follows,

$$\frac{\sigma_0(\xi/\alpha) \vdash \tau}{\forall\alpha\,\sigma_0 \vdash \tau},$$

and the only interesting case is when $\sigma_0 = \sigma_1 \to \sigma_2$, the variable $\alpha$ does not occur freely in $\sigma_1$ and $\sigma' = \sigma_1 \to \forall\alpha\, \sigma_2$. (The other cases are being dealt with by induction hypothesis.) Thus we have derived

$$\sigma_1 \to \sigma_2(\xi/\alpha) \vdash \tau,$$

and this is followed by ($\forall$-left). Since the derivation is in normal form, it follows that the previous step must have been the ($\to$) rule. Thus $\tau = \tau_1 \to \tau_2$ and we get that $\tau_1 \vdash \sigma_1$ and $\sigma_2(\xi/\alpha) \vdash \tau_2$ are derivable in $\vdash_{LMS}$. Applying ($\forall$-left) and then ($\to$) to the latter sequent we conclude that

$$\sigma_1 \to \forall\alpha\, \sigma_2 \vdash \tau$$

is derivable in $\vdash_{LMS}$ and the weight of this derivation is the same as the weight of the original $\sigma \vdash \tau$.

Assume now that we have a derivation of $\sigma' \vdash \tau'$ with a weight smaller than the weight of $\sigma \vdash \tau$, and choose $\sigma$, $\tau$, and $\sigma'$ so that our derivation is the shortest possible normal derivation of this property. There are two cases depending on whether (B2) was applied to $\sigma$ or to $\tau$. In the latter case a contradiction is derived easily with help of Lemma 3.2, so let us consider the former case. We have $\tau' = \tau$, and let $\sigma = \forall\alpha\,(\sigma_1 \to \sigma_2)$ and $\sigma' = \sigma_1 \to \forall\alpha\, \sigma_2$. Since our derivation is normal it must end with either an application of ($\forall_i$-right) or an application of ($\to$). If the last rule is ($\forall_i$-right), then by removing the last step we obtain a "bad" derivation which is shorter than the original one. Thus the last rule must be ($\to$), with its right premise being obtained from an application of ($\forall$-left). That is, we have $\forall\alpha\, \sigma_2 \vdash \tau_2$ derived from $\sigma_2(\rho/\alpha) \vdash \tau_2$, for some $\rho$, where $\tau = \tau_1 \to \tau_2$. The other premise of ($\to$) is $\tau_1 \vdash \sigma_1$. With no change of weight we can permute these two rules to obtain $\forall\alpha\,(\sigma_1 \to \sigma_2) \vdash \tau$; that is $\sigma \vdash \tau$. ■

The next three lemmas are needed to establish the invariance of the weight under rewrites induced by (B3).

LEMMA 3.5. *If $\sigma \sqsubseteq \tau$, then for every variable $\alpha$, the weight of $\sigma(\bot/\alpha) \sqsubseteq \tau(\bot/\alpha)$ is not greater than the weight of $\sigma \sqsubseteq \tau$.*

*Proof.* Induction on length of derivation of $\sigma \vdash \tau$ in $\vdash_{LMS}$. We leave the routine details for the reader. ■

LEMMA 3.6. *Let $\sigma$, $\xi$, and $\tau$ be types and assume that a variable $\alpha$ has $n$ occurrences in $\sigma$. Let $\vec{\rho}$ be a sequence of types and let $\vec{\beta}$ be a sequence of type variables of the same length as $\vec{\rho}$. Then*

(i) *If $\alpha$ occurs only positively in $\sigma$ and $\sigma(\xi/\alpha, \vec{\rho}/\vec{\beta}) \sqsubseteq \tau$, then $\sigma(\bot/\alpha, \vec{\rho}/\vec{\beta}) \sqsubseteq \tau$ and the weight of the latter inequality is not greater than the weight of the former inequality plus $n \cdot |\xi|$.*

(ii) *If $\alpha$ occurs only negatively in $\sigma$ and $\tau \sqsubseteq \sigma(\xi/\alpha, \vec{\rho}/\vec{\beta})$, then $\tau \sqsubseteq \sigma(\bot/\alpha, \vec{\rho}/\vec{\beta})$ and the weight of the latter inequality is not greater than the weight of the former inequality plus $n \cdot |\xi|$.*

*Proof.* The proof is by mutual induction on $\sigma$. The induction hypothesis is that (i) and (ii) hold for all possible substitutions $\vec{\rho}/\vec{\beta}$. The base step consists in considering two cases: when $\sigma = \alpha$ and when $\sigma = \beta \neq \alpha$. For the first case, use the easy fact that $\tau \sqsubseteq \tau$ is of weight zero; for the second case note that $\gamma \sqsubseteq \tau$ implies $\gamma = \tau$ if $\gamma$ is a variable.

The induction step is completely routine as well. For the case of $\to$ we mutually use (i) and (ii) as induction hypotheses and for the case of $\forall$ it is essential that $\vec{\rho}/\vec{\beta}$ is arbitrary. We leave the details for the reader. ■

LEMMA 3.7. *Let $\sigma$ and $\tau$ be types. Let $\vec{\rho}$ be a sequence of types and let $\vec{\beta}$ be a sequence of type variables, of the same length as $\vec{\rho}$. Then*

(i) *If $\alpha$ occurs only negatively in $\sigma$ and $\sigma(\bot/\alpha, \vec{\rho}/\vec{\beta}) \sqsubseteq \tau$, then $\sigma(\vec{\rho}/\vec{\beta}) \sqsubseteq \tau$ and the weight of the latter inequality is not greater than the weight of the former inequality.*

(ii) *If $\alpha$ occurs only positively in $\sigma$ and $\tau \sqsubseteq \sigma(\bot/\alpha, \vec{\rho}/\vec{\beta})$, then $\tau \sqsubseteq \sigma(\vec{\rho}/\vec{\beta})$ and the weight of the latter inequality is not greater than the weight of the former inequality.*

*Proof.* The proof is by mutual induction on $\sigma$. Let us consider the base case $\sigma = \alpha$ and let us prove (ii). Assume that $\tau \vdash_{LMS} \forall\alpha\,\alpha$ and assume that the derivation is in normal form (see Lemma 3.1). It follows that the last rule in this derivation must have been ($\forall_0$-right) and we have had $\tau \vdash \alpha$.

The rest of the proof is completely routine and we leave it for the reader. ■

LEMMA 3.8. *If $\sigma' \sqsubseteq \tau'$ is obtained from $\sigma \sqsubseteq \tau$ by one application of* (B3), *then the weight of $\sigma' \sqsubseteq \tau'$ is equal to the weight of $\sigma \sqsubseteq \tau$.*

*Proof.* We proceed by induction on the length of derivation of $\sigma \vdash \tau$ in $\vdash_{LMS}$, of the least possible weight. Again, the base case and the case of rule($\to$) are routine. Consider the case when the last rule in this derivation was ($\forall$-left). Then $\sigma = \forall\alpha\,\sigma_1$ and

$$\frac{\sigma_1(\xi/\alpha) \vdash \tau}{\forall\alpha\,\sigma_1 \vdash \tau}.$$

The nontrivial case is when $\sigma' = \sigma_1(\bot/\alpha)$ and $\alpha$ occurs only positively in $\sigma_1$. By Lemma 3.6(i) we have $\sigma_1(\bot/\alpha) \sqsubseteq \tau$ and the weight of this inequality is not greater than the weight of $\forall\alpha\,\sigma_1 \sqsubseteq \tau$.

Now suppose that the weight of $\sigma_1(\bot/\alpha) \sqsubseteq \tau$ is strictly less than the weight of $\forall\alpha\,\sigma_1 \sqsubseteq \tau$. Then we can take a derivation of the latter inequality and complete it with

$$\frac{\sigma_1(\bot/\alpha) \vdash \tau}{\forall\alpha\,\sigma_1 \vdash \tau}$$

to a derivation of a smaller weight, a contradiction.

Finally let us consider the case when the last rule was ($\forall_n$-right). Then $\tau = \tau_1 \to \cdots \to \tau_n \to \forall\alpha\,\rho$ and the nontrivial case is when $\alpha$ occurs only positively in $\rho$. We have

$$\frac{\sigma \vdash \tau_1 \to \cdots \to \tau_n \to \rho}{\sigma \vdash \tau_1 \to \cdots \to \tau_n \to \forall\alpha\,\rho}$$

and $\alpha$ does not occur free in $\sigma, \tau_1, \ldots, \tau_n$. By Lemma 3.5 we have

$$\sigma \sqsubseteq \tau_1 \to \cdots \to \tau_n \to \rho(\bot/\alpha)$$

with weight not increased.

Now assume that the weight of $\sigma \sqsubseteq \tau_1 \to \cdots \to \tau_n \to \rho(\bot/\alpha)$ is strictly smaller than the weight of $\sigma \sqsubseteq \tau_1 \to \cdots \to \tau_n \to \forall\alpha\,\rho$. By Lemma 3.5(ii) we conclude that the weight of $\sigma \vdash \tau_1 \to \cdots \to \tau_n \to \rho$ is strictly smaller than the weight of $\sigma \sqsubseteq \tau_1 \to \cdots \to \tau_n \to \forall\alpha\,\rho$, which is a contradiction. This proves that both weights are equal. ■

We conclude this section with the following invariance result.

PROPOSITION 3.1. *If $\sigma$ and $\sigma'$ are bicoercible and $\tau$ and $\tau'$ are bicoercible, then the weight of $\sigma \sqsubseteq \tau$ is equal to the weight of $\sigma' \sqsubseteq \tau'$, provided both inequalities hold.*

*Proof.* Type $\sigma$ can be rewritten into $\sigma'$ through a sequence of atomic steps induced by the axioms (B1), (B2), and (B3), viewed as left-to-right rules as well as right-to-left rules. By Lemmas 3.3, 3.4, and 3.8 all these steps do not change the weight of the inequalities. The same argument applies to $\tau$ and $\tau'$. ■

Let us observe the following consequence of Proposition 3.1. If $\tau \sim \sigma$ then the weight of $\tau \sqsubseteq \sigma$ is the same as the weight of $\tau \sqsubseteq \tau$, and it is easy to see that the latter equals zero. That is, inequalities between bicoercible types are of weight zero (although the reader will easily find examples showing that the converse does not hold).

## 4. FORCING INEQUALITIES

In this section we prove several properties of subtyping, which will be used as an essential tool in the proofs of Section 6.

LEMMA 4.1. *If $\sigma_1 \to \sigma_2 \sqsubseteq \tau_1 \to \tau_2$, then $\tau_1 \sqsubseteq \sigma_1$ and $\sigma_2 \sqsubseteq \tau_2$. Moreover the weight of the inequalities in the conclusion is smaller than or equal the weight of $\sigma_1 \to \sigma_2 \sqsubseteq \tau_1 \to \tau_2$.*

*Proof.* Take the derivation of $\sigma_1 \to \sigma_2 \vdash \tau_1 \to \tau_2$ in $\vdash_{LMS}$. The proof is by induction on the number of ($\forall_n$-right) rules that end this derivation. If this number is 0, then the last rule is ($\to$), and we are done. Otherwise the last step in the derivation looks as follows.

$$(\forall_n\text{-right}) \quad \frac{\sigma_1 \to \sigma_2 \vdash \tau_1 \to \tau_2'}{\sigma_1 \to \sigma_2 \vdash \tau_1 \to \tau_2}, \tag{9}$$

where $n > 0$. By the induction hypothesis we obtain $\tau_1 \sqsubseteq \sigma_1$ and $\sigma_2 \sqsubseteq \tau_2'$. Thus $\sigma_2 \vdash_{LMS} \tau_2'$ holds. It follows from (9) that

$$(\forall_{n-1}\text{-right}) \quad \frac{\sigma_2 \vdash \tau_2'}{\sigma_2 \vdash \tau_2}$$

is a legal derivation. Thus $\sigma_2 \sqsubseteq \tau_2$. It is easy to see that the weight of $\tau_1 \sqsubseteq \sigma_1$ and $\sigma_2 \sqsubseteq \tau_2$ is not greater than the weight of $\sigma_1 \to \sigma_2 \sqsubseteq \tau_1 \to \tau_2$. ■

LEMMA 4.2. *Let $\sigma \sqsubseteq \tau$ and let $\rho$ and $\nu$ be subtypes in $\sigma$ and $\tau$, respectively, which occur along the same path $\pi$. Assume that no free variable of $\rho$ is bound in $\sigma$ and no free variable of $\nu$ is bound in $\tau$. If $\pi$ is positive, then $\rho \sqsubseteq \nu$ holds; otherwise, $\nu \sqsubseteq \rho$.*

*Proof.* We prove the lemma by induction on the length of $\pi$, denoted $|\pi|$. If $|\pi| = 0$, then $\sigma$ is of the form

$$\sigma = \forall\beta_1 \cdots \forall\beta_n\, \rho$$

and similarly $\tau$ is of the form

$$\tau = \forall\gamma_1 \cdots \forall\gamma_m\, \nu,$$

where the quantifiers $\forall\beta_1 \cdots \forall\beta_n$ and $\forall\gamma_1 \cdots \forall\gamma_m$ are dummies in $\sigma$ and $\tau$, respectively. Thus $\tau \sim \nu$ and $\sigma \sim \rho$. It follows that

$$\rho \sqsubseteq \nu.$$

Let $|\pi| > 0$. Let us observe that we can assume without loss of generality that $\tau$ does not start with a quantifier. Indeed, it immediately follows from the following property. For all types $\xi$ and $\tau'$ and a type variable $\alpha \notin FV(\xi)$

$$\xi \sqsubseteq \forall\alpha\, \tau' \Leftrightarrow \xi \sqsubseteq \tau'.$$

The proof of ($\Rightarrow$) follows from (inst) and (trans), while the proof of ($\Leftarrow$) follows from ($\forall_0$-right).

Thus we may assume that $\tau$ is of the form $\tau_1 \to \tau_2$. Let

$$\sigma = \forall\alpha_1 \cdots \forall\alpha_n\, (\sigma_1 \to \sigma_2).$$

We prove the hypothesis by induction on $n$. For $n = 0$ the conclusion immediately follows from Lemma 4.1 and the main induction hypothesis applied either to $\tau_1 \sqsubseteq \sigma_1$, or to $\sigma_2 \sqsubseteq \tau_2$, depending on where the subterms occur in $\sigma$ and $\tau$.

Let $n > 0$ and let $\sigma = \forall\alpha_1\, \sigma'$. Consider a normal derivation of $\forall\alpha_1\, \sigma' \vdash \tau_1 \to \tau_2$ in $\vdash_{LMS}$. Since $\forall\alpha_1\, \sigma' \vdash \tau_1 \to \tau_2$ is not an axiom, it follows that the $\forall\alpha_1$ in $\sigma$ has been introduced by ($\forall$-left) rule, followed by a sequence of ($\forall_i$-right) rules: ($\forall_{i_1}$-right)$\cdots$($\forall_{i_m}$-right), with $i_j > 0$, for $j = 1, \ldots, m$ (where $m$ can be equal to 0). We consider the last application of ($\forall$-left) rule and allow $m = 0$. Let

$$\frac{\sigma'(\xi/\alpha_1) \vdash \tau_1 \to \tau_2'}{\forall\alpha_1\, \sigma' \vdash \tau_1 \to \tau_2'} \quad (\forall\text{-left})$$

be the last application of ($\forall$-left) rule in this derivation.

Let us observe that the number of leading quantifiers in $\sigma'(\xi/\alpha_1)$ is smaller than $n$; thus we can apply the induction hypothesis to $\sigma'(\xi/\alpha_1) \sqsubseteq \tau_1 \to \tau_2'$. Assume first that the ($\forall_i$-right) rules following the last ($\forall$-left) rule do not introduce a quantifier below the point where $\nu$ occurs in $\tau_1 \to \tau_2$. Then $\nu$ also occurs in $\tau_1 \to \tau_2'$ along $\pi$ and, by the induction hypothesis, we can conclude that $\rho \sqsubseteq \nu$ or $\nu \sqsubseteq \rho$ holds, depending on the parity of $\pi$. (This is because $\rho$ having no free occurrence of $\alpha_1$ is not affected by the instantiation and thus remains in $\sigma(\xi/\alpha_1)$ along $\pi$.) Otherwise, let $\nu'$ be a subtype of $\tau_1 \to \tau_2'$ along $\pi$ which is further transformed into $\nu$ by the sequence of ($\forall_i$-right) rules. It follows that $\pi$ must be of the form $1^k$, for some $k \geq 0$. Thus, by the induction hypothesis we have

$$\rho \sqsubseteq \nu'.$$

Thus the sequent $\rho \vdash \nu'$ is derivable in $\vdash_{LMS}$. Applying the rules ($\forall_{i_1-1}$-right), ..., ($\forall_{i_m-1}$-right) we obtain

$$\rho \vdash_{LMS} \nu.$$

Hence $\rho \sqsubseteq \nu$ holds, as required. ■

LEMMA 4.3. *If $\rho \sim \tau$ then $\sigma(\rho/\alpha) \sim \sigma(\tau/\alpha)$, for all $\sigma$.*

*Proof.* Obvious induction. ■

LEMMA 4.4. *Let $\sigma$ and $\tau$ be types satisfying the following two conditions.*

(i) *$\tau$ contains no quantifier on paths of the form $1^m$.*

(ii) *There are two paths, positive $\pi_1$ and negative $\pi_2$, such that they lead in $\sigma$ and in $\tau$ to an occurrence of the same free variable $\alpha$.*

*Then, for every type $\rho$,*

$$\forall\alpha\, \sigma \sqsubseteq \tau(\rho/\alpha) \Leftrightarrow \sigma(\rho/\alpha) \sqsubseteq \tau(\rho/\alpha).$$

*If both these inequalities are true, then the weight of $\sigma(\rho/\alpha) \sqsubseteq \tau(\rho/\alpha)$ is less than or equal to the weight of $\forall\alpha\, \sigma \sqsubseteq \tau(\rho/\alpha)$. In addition, if the number of $\to$'s in $\rho$ is larger than 0 (i.e., $|\rho| > 0$), and both these inequalities are true, then the weight of $\sigma(\rho/\alpha) \sqsubseteq \tau(\rho/\alpha)$ is strictly less than the weight of $\forall\alpha\, \sigma \sqsubseteq \tau(\rho/\alpha)$.*

*Proof.* Part ($\Leftarrow$) is obvious since $\forall\alpha\, \sigma \sqsubseteq \sigma(\rho/\alpha)$. For the proof of ($\Rightarrow$) let us assume that $\forall\alpha\, \sigma \sqsubseteq \tau(\rho/\alpha)$. Consider a normal derivation of the sequent $\forall\alpha\, \sigma \vdash \tau(\rho/\alpha)$ in $\vdash_{LMS}$, and choose one with the minimal weight, say $n$. Since $\forall\alpha\, \sigma \vdash \tau(\rho/\alpha)$ is not an axiom, it follows from (i) that the derivation must end with ($\forall$-left). Thus there is a type $\xi$ such that

$$\sigma(\xi/\alpha) \vdash_{LMS} \tau(\rho/\alpha),$$

Therefore $\sigma(\xi/\alpha) \sqsubseteq \tau(\rho/\alpha)$ and applying Lemma 4.2 twice to the paths $\pi_1$ and $\pi_2$, we obtain

$$\xi \sqsubseteq \rho \quad \text{and} \quad \rho \sqsubseteq \xi.$$

Thus $\xi$ and $\rho$ are bicoercible and therefore $\sigma(\rho/\alpha) \sqsubseteq \tau(\rho/\alpha)$, by Lemma 4.3. Clearly $\sigma(\xi/\alpha) \sqsubseteq \tau(\rho/\alpha)$ has weight less than or equal to $n$. By Proposition 3.1, $\sigma(\rho/\alpha) \sqsubseteq \tau(\rho/\alpha)$ also has weight less than or equal to $n$.

In addition, if $|\rho| > 0$ then $|\xi| > 0$ and the weight of $\sigma(\xi/\alpha) \sqsubseteq \tau(\rho/\alpha)$ is strictly less than $n$. It follows from Proposition 3.1 that the weight of $\sigma(\rho/\alpha) \sqsubseteq \tau(\rho/\alpha)$ is equal to the weight of $\sigma(\xi/\alpha) \vdash \tau(\rho/\alpha)$; i.e., it is less than $n$. ■

## 5. THE STACK-REGISTER MACHINES

Below we describe a computing device, which we call a *stack-register machine*. The halting problem for stack-register machines will be used as an intermediate step in our reduction. That is, we first reduce the halting problem for two-counter machines to the halting problem for stack-register machines, and then the latter will be reduced to the subtyping problem.

Informally speaking, a stack-register machine is a deterministic one-state device that has two main registers, and a finite number of auxiliary registers $r_1, \ldots, r_n$, where each register is capable of holding a (nonempty) stack of instruction labels. At each step, the machine reads the top label from one of the main registers (in an alternating manner) and modifies the auxiliary registers according to the appropriate instruction. That is, the contents of every register $r_i$ are replaced by the contents of another register $r_j$, with some (or none) labels added at the top of it. An exception is when the top label is the last one so that the currently scanned main register $R$ becomes empty after reading it. In this case, its contents are restored by copying $r_1$ onto $R$ *before* executing the instruction. There is one conditon that must be satisfied: if $\ell$ is a label of an instruction $I$, then all labels mentioned in $I$ must be less than $\ell$ with respect to a fixed partial order.

To describe our machines more formally, let us first assume that $\langle \Sigma, \leq \rangle$ is a finite partially ordered set. Elements of $\Sigma$ are called *instruction labels*. An *n-ary instruction* over $\Sigma$ is formally defined as a tuple of pairs $(\langle i_1, w_1 \rangle, \ldots, \langle i_n, w_n \rangle)$, where $i_1, \ldots, i_n \in \{1, \ldots, n\}$ and $w_1, \ldots, w_n$ are words over $\Sigma$, but one may prefer to see it as a (simultaneous) assignment command:

$$(r_1, \ldots, r_n) := \big(w_1 \cdot r_{i_1}, \ldots, w_n \cdot r_{i_n}\big).$$

A label is said to *occur* in the instruction $(\langle i_1, w_1 \rangle, \ldots, \langle i_n, w_n \rangle)$ iff it occurs in $w_1 \cdots w_n$.

The result of performing such an instruction on a tuple of strings $(v_1, \ldots, v_n)$ is of course defined as $(w_1 \cdot v_{i_1}, \ldots, w_n \cdot v_{i_n})$. The intended meaning is as follows: suppose $(v_1, \ldots, v_n)$ are the current contents of registers $(r_1, \ldots, r_n)$, respectively. Then, for each $k$, the word $w_k v_{i_k}$ is assigned to the register $r_k$.

An $n$-ary *stack-register machine* is defined as a quadruple of the form

$$\mathcal{M} = \langle \Sigma, \leq, \mathcal{I}, e \rangle,$$

where $\langle \Sigma, \leq \rangle$ is a finite partially ordered set, the symbol $e$ denotes a fixed minimal element of $\Sigma$, called an *end label*, and finally $\mathcal{I}$ is a function that assigns an $n$-ary instruction $\mathcal{I}(\ell)$ over $\Sigma$ to every label in $\Sigma - \{e\}$ in such a way that $\ell' < \ell$ holds for all labels $\ell'$ occurring in $\mathcal{I}(\ell)$.

An instantaneous description (ID) of an $n$-ary stack-register machine $\mathcal{M}$ is an $n+2$-tuple of nonempty words $(V_1, V_2, v_1, \ldots, v_n)$. The $v_i$'s are values of auxiliary registers, while $V_1$ and $V_2$ are values of main registers, the first one being currently scanned. Given such an ID, the next ID is obtained as follows:

- If $V_1 = e \cdot V_1'$, then there is no next ID—the machine stops.
- If $V_1 = \ell \cdot V_1'$ with $\ell \neq e$, and $(v_1', \ldots, v_n')$ is the result of performing the instruction $\mathcal{I}(\ell)$ on $(v_1, \ldots, v_n)$, then the next ID is $(V_2, V_1', v_1', \ldots, v_n')$, provided $V_1'$ is not empty. (Note that the main registers are switched.)
- If $V_1 = \ell \cdot V_1'$ with $\ell \neq e$, but $V_1'$ is empty, then the next ID is $(V_2, v_1, v_1', \ldots, v_n')$, where $v_1', \ldots, v_n'$ are as above. (That is, the main register is first assigned the value of the auxiliary register $r_1$, and only then the instruction is normally executed.)

The halting problem for stack-register machines is whether a given stack-register machine halts for a given ID.

## 6. ENCODING OF A STACK-REGISTER MACHINE BY INEQUALITIES

Given a stack-register machine $\mathcal{M} = \langle \Sigma, \leq, \mathcal{I}, e \rangle$ we will encode the halting problem of $\mathcal{M}$ by two types $\sigma$ and $\tau$, so that $\mathcal{M}$ halts iff $\sigma \sqsubseteq \tau$ holds.

First we encode finite sequences of instructions of $\mathcal{M}$. For a word $w \in \Sigma^*$ we define a type $\hat{\sigma}_w$. For this definition we fix two type variables $\beta$ and $\gamma$. These variables are going to occur freely in all types $\hat{\sigma}_w$, except for those $w$'s which start with the end label $e$.

In order to properly define types $\hat{\sigma}_w$, let us assume that $\Sigma = \{\ell_1, \dots, \ell_r\}$ is listed in such a way that $e = \ell_1$ and $\ell_i < \ell_j$ implies $i < j$, for all $i, j = 1, \dots r$. The definition of $\hat{\sigma}_w$ is by induction with respect to two parameters:

(1) the largest $i$ such that $\ell_i$ occurs in $w$ (zero if none);

(2) the length of $w$.

First we encode the end label $e$ by

$$\hat{\sigma}_e = \forall\alpha\, \alpha$$

and the empty word by

$$\hat{\sigma}_\varepsilon = \beta.$$

If $\hat{\sigma}_\ell$ has been defined for all labels which occur in $w$, then we can define $\hat{\sigma}_w$ by induction on the length of $w$.

$$\hat{\sigma}_{\ell\cdot w} = \hat{\sigma}_\ell(\hat{\sigma}_w/\beta).$$

The encoding for other labels is as follows. Let $\mathcal{I}(\ell) = (\langle i_1, w_1 \rangle, \dots, \langle i_n, w_n \rangle)$. We define

$$\begin{aligned}\hat{\sigma}_\ell = \forall\alpha_1 \dots \forall\alpha_n \big[&(\alpha_1 \to \alpha_1) \to \cdots \to (\alpha_n \to \alpha_n)\\ &\to \big[\big(\hat{\sigma}_{w_1}\big(\alpha_{i_1}/\beta\big) \to \hat{\sigma}_{w_1}\big(\alpha_{i_1}/\beta\big)\big) \to \cdots \to \big(\hat{\sigma}_{w_n}\big(\alpha_{i_n}/\beta\big) \to \hat{\sigma}_{w_n}\big(\alpha_{i_n}/\beta\big)\big)\\ &\to \beta \to \gamma\big] \to \gamma\big].\end{aligned}$$

Note that, according to our restriction on machine instructions, the words $w_1, \dots, w_n$ may only contain labels less than $\ell$, and we can assume that $\hat{\sigma}_w$ are already defined.

Clearly, $\hat{\sigma}_w$ has at most one free occurrence of $\beta$, and we can assume that there is no bound occurrence of $\beta$. We finally define for $w \neq \varepsilon$ the type

$$\sigma_w = C[\alpha_1],$$

where $C[\,]$ is a context such that $\hat{\sigma}_w = C[\beta]$. Note that the variable $\alpha_1$ gets caught by a quantifier of $\hat{\sigma}_w$.

The following property follows immediately from our definitions. Let $w = w' \cdot w''$ be such that $e$ does not occur in $w'$ and $w'' \neq \varepsilon$. Then

$$\sigma_w = \hat{\sigma}_{w'}(\sigma_{w''}/\beta). \tag{10}$$

The proof of (10) is by induction on the length of $w'$. We leave it for the reader.

We also have for all $w'$ and $w''$,

$$\sigma_{w' \cdot e \cdot w''} = \sigma_{w' \cdot e}.$$

In particular, $|\sigma_w| > 0$ iff $w$ does not start with $e$. We will use this observation in the proof of the next result.

The reduction is based on the following fact.

PROPOSITION 6.1. *$\mathcal{M}$ halts for an instantaneous description $(V_1, V_2, v_1, \ldots, v_n)$ iff*

$$\sigma_{V_1} \sqsubseteq \big(\sigma_{v_1} \to \sigma_{v_1}\big) \to \cdots \to \big(\sigma_{v_n} \to \sigma_{v_n}\big) \to \sigma_{V_2} \to \gamma.$$

*Proof.* We prove the "only if" part by induction with respect to the length of computation and the "if" part by induction with respect to the weight of the inequality.

If $(V_1, V_2, v_1, \ldots, v_n)$ is a terminal ID, i.e., if $V_1 = e \cdot V_1'$, then $\sigma_{V_1} = \forall\alpha\, \alpha$, and the inequality in the conclusion of the proposition holds.

Assume that $(V_1, V_2, v_1, \ldots, v_n)$ is not a terminal ID, i.e., that $V_1$ begins with a label $\ell \neq e$. Let $\mathcal{I}(\ell) = (\langle i_1, w_1 \rangle, \ldots, \langle i_n, w_n \rangle)$ and let $(v_1', \ldots, v_n')$ be the result of performing the instruction $\mathcal{I}(\ell)$ on $(v_1, \ldots, v_n)$. Thus

$$v_j' = w_j \cdot v_{i_j}, \quad \text{for } j = 1, \ldots, n. \tag{11}$$

For the induction to work we need the following two properties.

If $V_1 = \ell \cdot V_1'$ with $V_1' \neq \varepsilon$, then

$$\begin{aligned} &\sigma_{V_1} \sqsubseteq \big(\sigma_{v_1} \to \sigma_{v_1}\big) \to \cdots \to \big(\sigma_{v_n} \to \sigma_{v_n}\big) \to \sigma_{V_2} \to \gamma \\ &\quad \Leftrightarrow \sigma_{V_2} \sqsubseteq \big(\sigma_{v_1'} \to \sigma_{v_1'}\big) \to \cdots \to \big(\sigma_{v_n'} \to \sigma_{v_n'}\big) \to \sigma_{V_1'} \to \gamma. \end{aligned} \tag{12}$$

If $V_1 = \ell$, then

$$\begin{aligned} &\sigma_{V_1} \sqsubseteq \big(\sigma_{v_1} \to \sigma_{v_1}\big) \to \cdots \to \big(\sigma_{v_n} \to \sigma_{v_n}\big) \to \sigma_{V_2} \to \gamma \\ &\quad \Leftrightarrow \sigma_{V_2} \sqsubseteq \big(\sigma_{v_1'} \to \sigma_{v_1'}\big) \to \cdots \to \big(\sigma_{v_n'} \to \sigma_{v_n'}\big) \to \sigma_{v_1} \to \gamma. \end{aligned} \tag{13}$$

We prove (12). Let $V_1 = \ell \cdot V_1'$ with $V_1' \neq \varepsilon$. By (10) we have

$$\begin{aligned} \sigma_{V_1} = \forall\alpha_1 \cdots \forall\alpha_n \big[&(\alpha_1 \to \alpha_1) \to \cdots \to (\alpha_n \to \alpha_n) \\ &\to \big[\big(\hat{\sigma}_{w_1}\big(\alpha_{i_1}/\beta\big) \to \hat{\sigma}_{w_1}\big(\alpha_{i_1}/\beta\big)\big) \to \cdots \to \big(\hat{\sigma}_{w_n}\big(\alpha_{i_n}/\beta\big) \to \hat{\sigma}_{w_n}\big(\alpha_{i_n}/\beta\big)\big) \\ &\to \sigma_{V_1'} \to \gamma\big] \to \gamma\big]. \end{aligned}$$

Let $\nu = (\alpha_1 \to \alpha_1) \to \cdots \to (\alpha_n \to \alpha_n) \to [(\hat{\sigma}_{w_1}(\alpha_{i_1}/\beta) \to \hat{\sigma}_{w_1}(\alpha_{i_1}/\beta)) \to \cdots \to (\hat{\sigma}_{w_n}(\alpha_{i_n}/\beta) \to \hat{\sigma}_{w_n}(\alpha_{i_n}/\beta)) \to \sigma_{V_1'} \to \gamma] \to \gamma$. Applying $n$ times Lemma 4.4, we have the following equivalence

$$\begin{aligned} &\sigma_{V_1} \sqsubseteq \big(\sigma_{v_1} \to \sigma_{v_1}\big) \to \cdots \to \big(\sigma_{v_n} \to \sigma_{v_n}\big) \to \sigma_{V_2} \to \gamma \\ &\quad \Leftrightarrow S(\nu) \sqsubseteq \big(\sigma_{v_1} \to \sigma_{v_1}\big) \to \cdots \to \big(\sigma_{v_n} \to \sigma_{v_n}\big) \to \sigma_{V_2} \to \gamma, \end{aligned} \tag{14}$$

where $S$ is a substitution which assigns to each $\alpha_i$ type $\sigma_{v_i}$, for $i = 1, \ldots, n$. Now, applying $n + 1$ times Lemma 4.1, we obtain

$$\begin{aligned} &S(\nu) \sqsubseteq \big(\sigma_{v_1} \to \sigma_{v_1}\big) \to \cdots \to \big(\sigma_{v_n} \to \sigma_{v_n}\big) \to \sigma_{V_2} \to \gamma \\ &\quad \Leftrightarrow \sigma_{V_2} \sqsubseteq \big(\hat{\sigma}_{w_1}\big(\sigma_{v_{i_1}}/\beta\big) \to \hat{\sigma}_{w_1}\big(\sigma_{v_{i_1}}/\beta\big)\big) \to \cdots \\ &\quad \to \big(\hat{\sigma}_{w_n}\big(\sigma_{v_{i_n}}/\beta\big) \to \hat{\sigma}_{w_n}\big(\sigma_{v_{i_n}}/\beta\big)\big) \to \sigma_{V_1'} \to \gamma. \end{aligned} \tag{15}$$

By (10), (11), (14) and (15) we obtain (12). The proof of (13) is very similar. The difference is that, instead of $\sigma_{V_1'}$, we now have $\sigma_{v_1}$ at the appropriate place at the right-hand side of (15), because of the bound occurrence of $\alpha_1$ at the end of $\sigma_{V_1}$. We leave the details for the reader.

Clearly, the weights of the right-hand sides of (12) and (13) are not greater than the weight of the respective left-hand sides. Now, if at least one $v_i$ does not start with $e$, then $|\sigma_{v_i}| > 0$ and by Lemma 4.4, in both cases the weight of the right-hand side is strictly smaller than the weight of the left-hand side.

Now, using (12) and (13) the proof of the "only if" part follows by a completely routine induction on the length of computation.

We prove the "if" part by induction on the weight of the inequality. Assume that

$$\sigma_{V_1} \sqsubseteq (\sigma_{v_1} \to \sigma_{v_1}) \to \cdots \to (\sigma_{v_n} \to \sigma_{v_n}) \to \sigma_{V_2} \to \gamma$$

holds and let the next ID be $(U_1, U_2, u_1, \ldots, u_n)$. Using (12) and (13) we obtain that

$$\sigma_{U_1} \sqsubseteq (\sigma_{u_1} \to \sigma_{u_1}) \to \cdots \to (\sigma_{u_n} \to \sigma_{u_n}) \to \sigma_{U_2} \to \gamma$$

must also hold. In addition, if at least one $v_i$ does not start with $e$ then the weight of the latter inequality is strictly less than that of the former. Thus, by induction hypothesis, we are done. On the other hand, if all $v_i$'s start with $e$, then we perform the computation steps of $\mathcal{M}$ until it either pushes on one of the auxiliary stacks a symbol different than $e$, in which case we can apply the previous reasoning, or else it terminates by eventually copying the first auxiliary register to $V_1$. Hence, in either case $\mathcal{M}$ will terminate. This completes the proof of Proposition 6.1. ■

## 7. PROGRAMMING WITH STACK-REGISTER MACHINES

We want to prove that the halting problem for stack-register machines is undecidable. For this, we show how to simulate the behaviour of a two-counter automaton with the help of a stack-register machine. Before we begin our main construction, let us first make a few observations which will simplify the consideration.

Let us start with the following remark. We defined machine ID's as tuples $(V_1, V_2, v_1, \ldots, v_n)$ with alternating main registers and the current register listed first. This is convenient in Section 6, but for our present goal we would prefer to have fixed main registers (each given a name) and a flag showing which one is currently scanned. Formally, this would mean that an ID of a stack-register machine should now be a tuple $(\alpha, V_1, V_2, v_1, \ldots, v_n)$, where $\alpha \in \{1, 2\}$. The new ID of the form $(1, V_1, V_2, v_1, \ldots, v_n)$ is understood as the old $(V_1, V_2, v_1, \ldots, v_n)$, while $(2, V_1, V_2, v_1, \ldots, v_n)$ replaces $(V_2, V_1, v_1, \ldots, v_n)$. There is one subtlety here: each old ID of the form $(V_1, V_2, v_1, \ldots, v_n)$ corresponds to two new ID's, namely $(1, V_1, V_2, v_1, \ldots, v_n)$ and $(2, V_2, V_1, v_1, \ldots, v_n)$. This is, however, not essential—the reader can easily check that the new approach is equivalent to the old one, after the obvious modification of the "next ID" function.

The above suggests one more convention: since we can distinguish the two main registers, we do not have to read their contents from the same auxiliary register. Instead, we could assume that there are two registers $r_1$ and $r_2$, such that $r_1$ is copied onto the first main register whenever the latter becomes empty and $r_2$ is used in the same way for the second main register. One possible way to simulate this by an ordinary stack-register machine may be as follows: the contents of both the main registers are always read from a new auxilary register $r_0$, but the latter is assigned the same value as $r_1$ in every even step and the same value as $r_2$ in every odd step. This is not as simple as it can appear at first, because we have to distinguish between odd steps (reading from the first main register) and the even steps (reading from the second one). Fortunately, the machine to be constructed below will have a nice property: the sets of labels occurring in the two main registers will be disjoint, and in this case the above trick does not cause any difficulty. (But, of course, a simulation is possible in the general case, e.g., by using two different copies of the alphabet, and two different sets of registers.)

Another convention that may simplify the construction is to present the instruction of a machine in the form of a sequence of assignment instructions rather than one simultaneous assignment. Of course, we mention only relevant assignments, skipping those of the form $r_i := r_i$. Fortunately, below we can always choose the order of assignments so that there is no confusion possible.

The last observation is as follows. With no loss of generality one can assume that an instruction may assign an arbitrary fixed string to a register, i.e., that instructions may involve assignments of the form $r_j := w$. To simulate such instructions one needs to keep an extra register $r_w$ for every such string $w$. This register is assigned the value $w$ in the initial ID and is never changed. Note that $w$ does not itself occur in the actual assignment $r_j := r_w$. Thus, the symbols occurring in $w$ are not counted as occurring in the instruction, and their positions in the partial order $\leq$ do not matter.

After this preparation, let us turn to our main subject. Recall that a *two-counter automaton* has a finite set of states, some of them final, and in each state it performs one of the following actions:

- Increase a counter by 1; go to another state;
- Decrease a counter by 1; go to another state;
- Test a counter for zero; go to another state, depending on the result.

The halting problem for two-counter automata (given an automaton, an initial state, and the initial values of the counters, decide if the machine reaches a final state) is well known to be undecidable (see [3]). Thus, it suffices for our needs to prove the following:

LEMMA 7.1. *The halting problem for two-counter automata is effectively reducible to the halting problem for stack-register machines.*

The proof of this lemma covers the rest of the section. Assume that a two-counter automaton $\mathcal{A}$ is given and that it has only one final state. A configuration of the automaton can be seen as a triple $(q, n, m)$, where $q$ is a current state, and $n$ and $m$ are the values of the first and second counter, respectively. We construct a stack-register machine $\mathcal{M}$ to simulate $\mathcal{A}$.

The general idea of the simulation is as follows. The first main register, called `State`, always holds only one label representing a state of $\mathcal{A}$. There are five labels: $q_0$, $q_1$, $q_2$, $q_3$, $q_4$, for each state $q$ of $\mathcal{A}$. This is because $\mathcal{M}$ makes a sequence of steps for each single step of $\mathcal{A}$, and we want to distinguish different phases of this process. The values of the counters are stored on two auxiliary registers `c1` and `c2` so that a number $n$ is coded as a string of the form $1 \dots 1\$$, with $n$ occurrences of "1." Each operation on a counter, say `c1`, is simulated by first copying its value to the second main register, called `Counter`, erasing `c1`, and then rebuilding it to the appropriate new value.

The label alphabet $\Sigma$ of our machine will consist of two parts: labels used for the `State` register will not occur in the `Counter` register and conversely.

- Labels $q_0$, $q_1$, $q_2$, $q_3$, $q_4$, where $q$ is a state of $\mathcal{A}$, will be used by the `State` register.
- Labels B (for "begin"), E (for "end"), and the labels 1, \$ for coding numbers, will occur in the `Counter` register.

The partial order on the set of labels is such that $q_3 > 1$ and $q_4 > 1$ holds for all $q$, and all other elements are incomparable. This means that 1 is the only label that can be pushed to the stacks and that it can only happen when the machine scans labels $q_3$ or $q_4$. The end label is $q_0^f$, where $q^f$ is the final state of $\mathcal{A}$.

Our machine has eight auxiliary registers: `next-State`, `next-Counter`, `c1`, `c2`, `new-0`, `new-1`, `alt-new-0`, `alt-new-1`. These registers are used as follows:

- The value of `next-Counter` is read to `Counter`, whenever the latter becomes empty.
- Similarly, `next-State` always holds the next value for `State`.
- Registers `c1`, `c2` store the values of counters.
- The role of the other four registers is to store information on the next state of the automaton $\mathcal{A}$. That is, when $\mathcal{A}$ is in state $q$ and will move to state $p$ after completing a counter operation, then the desired values of `new-0` and `new-1` are $p_0$ and $p_1$, respectively. Registers `alt-new-0` and `alt-new-1` are necessary in case of a test for zero, since there are two possibilities for the next state of $\mathcal{A}$.

For an arbitrary $n$, let $\mathbf{n}$ stand for the string $1^n\$$. An ID of $\mathcal{A}$ of the form $(q, n, m)$ will be represented by an ID of $\mathcal{M}$, such that the values of registers are as follows,

$$\texttt{State} = q_0, \quad \texttt{Counter} = \mathrm{B}, \quad \texttt{c1} = \mathbf{n}, \quad \texttt{c2} = \mathbf{m}, \quad \texttt{next-State} = q_1,$$

and that `State` is currently scanned. The values of other registers do not matter. Note that a final ID of $\mathcal{A}$ is only represented by final ID's of $\mathcal{M}$. Our goal is to construct $\mathcal{M}$ so that $(*)$ below implies $(**)$, for all triples $(q, n, m)$ and $(p, n', m')$.

- $(*)$ The automaton $\mathcal{A}$ moves from $(q, n, m)$ to $(p, n', m')$ in one step;
- $(**)$ The machine $\mathcal{M}$, when started in an ID representing $(q, n, m)$, after a sequence of moves reaches an ID representing $(p, n', m')$.

The behaviour of $\mathcal{M}$ obeys the alternating pattern of passing control from one main register to the other. That is, we have two processes, which are, to some extent, independent from each other (but communicating with the help of the common auxiliary registers). Let us begin with explaining the actions associated to the `Counter` register. The contents of `Counter` will always be either a numeral $\mathbf{n} = 1^n\$$ or a single symbol E or B.

If the symbol on the top of `Counter` is B then the machine executes the instruction $\mathcal{I}(\mathrm{B})$, which consists of only one assignment (i.e., other registers remain unchanged):

$$\texttt{next-Counter} := \mathrm{E}.$$

Of course, the symbol B is removed from the stack, and it is normally the last symbol. This means that the previous contents of `next-Counter` is copied to `Counter` before the above assignment takes place.

The instruction $\mathcal{I}(1)$ to be performed when “1” is read from `Counter` is:

$$\texttt{new-0} := \texttt{alt-new-0}; \quad \texttt{new-1} := \texttt{alt-new-1}.$$

We explain the meaning of this instruction later, and now we just note that “1” will normally not occur at the bottom of the stack and that reading it amounts to decreasing the stored number by 1.

If the symbol \$ is read from `Counter`, it will normally be the last symbol, so the new value of `Counter` is read from `next-Counter`, and then there are the assignments:

$$\texttt{next-State} := \texttt{new-0}; \quad \texttt{next-Counter} := \mathrm{B}.$$

Finally, if the symbol on the top of `Counter` is E, then the assignment to be performed is:

$$\texttt{next-State} := \texttt{new-1}.$$

Suppose now that the machine is started in an ID representing $(q, n, m)$ with the value of `next-Counter` set to a numeral $\mathbf{n} = 1^n\$$. It should be easy to see that the contents of `Counter` change as follows: first B is replaced by $\mathbf{n}$. This string is being read symbol by symbol until the stack is empty. The next symbol read from `next-Counter` is E and then again B.

Now we describe the instructions associated to labels occurring on the `State` register and their meaning. Suppose for the beginning that the action performed by $\mathcal{A}$ in state $q$ is to decrease the value of the first counter and to change the state to $p$. The instruction $\mathcal{I}(q_0)$ can be written as the following sequence of assignments:

```
next-Counter := c1;
c1 := $\mathbf{0}$;
next-State := $q_2$;
new-0 := $p_0$; alt-new-0 := $p_0$;
new-1 := $p_1$; alt-new-1 := $p_1$.
```

The other instructions related to $q$ are much simpler:

- $\mathcal{I}(q_1)$ is "`next-State` $:= q_3$";
- $\mathcal{I}(q_2)$ is "`next-State` $:= q_4$";
- $\mathcal{I}(q_3)$ is empty;
- $\mathcal{I}(q_4)$ is "`c1` $:= 1 \cdot$ `c1`".

Assume again that the machine is started in an ID representing $(q, n, m)$. Since `State` $= q_0$, the value of `next-Counter` is set to **n**. Thus, according to what we observed before, it takes $2n + 6$ steps to reach another ID of $\mathcal{M}$ satisfying `Counter` $=$ B, and such that `State` is scanned. The contents of `Counter` during these $2n + 6$ steps changes as follows:

$$\mathrm{B} \to \underline{\mathrm{B}} \to \mathbf{n} \to \underline{\mathbf{n}} \to \mathbf{n-1} \to \underline{\mathbf{n-1}} \to \cdots \to \mathbf{1} \to \underline{\mathbf{1}} \to \$ \to \underline{\$} \to \mathrm{E} \to \underline{\mathrm{E}} \to \mathrm{B}.$$

The underlined symbols refer to the ID's in which `Counter` is being read.

Consider now the behaviour of the registers `new-0` and `alt-new-0` during these $2n+6$ steps. They are both set to $p_0$ at the beginning, and no further instruction can change it. Thus, executing $\mathcal{I}(\$)$ will result in `next-State` $= p_0$. We conclude that at the last step, the value of `State` is $p_0$. Similarly, `new-1` and `alt-new-1` are assigned $p_1$, and thus at the last moment `next-State` $= p_1$.

In order to show that the final ID of our $2n + 6$ steps does represent the configuration $(p, n, m)$ of $\mathcal{A}$, it remains to check that `c1` $= \mathbf{n-1}$ (as it should be obvious that the contents of `c2` remain unchanged). For this it suffices to note that we have the following sequence of labels occurring on the `State` register (scanned labels are underlined).

$$\underline{q_0} \to q_1 \to \underline{q_1} \to q_2 \to \underline{q_2} \to q_3 \to \underline{q_3} \to q_4 \to \underline{q_4} \to \cdots \to q_4 \to \underline{q_4} \to q_4 \to \underline{p_0}$$

with $\mathcal{I}(q_4)$ executed exactly $n - 1$ times. (If $n = 0$ or $n = 1$ then $\mathcal{I}(q_4)$ is never executed.)

If the action performed by $\mathcal{A}$ in state $q$ is to increase the first counter rather than to decrease it, the instructions $\mathcal{I}(q_1)$, $\mathcal{I}(q_2)$, and $\mathcal{I}(q_4)$ are defined as in the previous case. The instruction $\mathcal{I}(q_3)$ is "`c1` $:= 1 \cdot$ `c1`"; that is, it is identical to $\mathcal{I}(q_4)$ (we keep a different label just for uniformity). Finally, $\mathcal{I}(q_0)$ differs from the previous case in that it contains the assignment "`c1` $:= \mathbf{1}$" instead of "`c1` $:= \mathbf{0}$." It is left to the reader to check that after $2n + 6$ steps we again obtain a desired ID.

The last case is a test for zero. Assume that $\mathcal{A}$ changes its state from $q$ to $p$ if the first counter is zero, and to $s$ otherwise. Then we define $\mathcal{I}(q_0)$ as follows:

```
next-Counter := c1;
c1 := 0;
next-State := q₂;
new-0 := p₀; alt-new-0 := s₀;
new-1 := p₁; alt-new-1 := s₁;
```

and $\mathcal{I}(q_i)$, for $i = 1, 2, 3, 4$, are the same as for the instructions increasing the first counter (i.e., $q_3$ adds 1 to `c1`). Since this time `c1` begins with zero and is increased $n$ times, its final value is the same. However, the final contents of `State` and `next-State` will now depend on whether $n$ is zero or not. Indeed, the values $p_0$ and $p_1$ will remain in registers `new-0` and `new-1` only if $\mathcal{I}(1)$ is never executed, i.e., if $n = 0$. The details are left for the reader.

Of course, instructions related to `c2` are defined analogously, and the instructions $\mathcal{I}(q_i^f)$, for $i > 0$, may be arbitrary, because these labels will never show up on the main registers. The reader may now easily conclude that the condition $(*)$ implies $(**)$, for every configuration of $\mathcal{A}$ and every ID of $\mathcal{M}$ representing it. Since both machines are deterministic, it follows that we have the equivalence of the following two conditions:

(•) The automaton $\mathcal{A}$ terminates when started in $(q, n, m)$;

(••) The machine $\mathcal{M}$ terminates when started in an ID representing $(q, n, m)$.

Thus, in order to answer the question "does $\mathcal{A}$ halt if started in the configuration $(q, n, m)$" it is enough to take any ID of $\mathcal{M}$ that represents $(q, n, m)$ and ask if $\mathcal{M}$ halts when started in this ID. This concludes the proof of Lemma 7.1 and of Theorem 2.1.

## ACKNOWLEDGMENT

The authors thank Damian Niwiński for pointing out a gap in an earlier version of this proof.

## REFERENCES

1. Barendregt, H. P. (1984), "The Lambda Calculus: Its Syntax and Semantics," North-Holland, Amsterdam.
2. Chrząszcz, J. (1998), Polymorphic subtyping without distibutivity, *in* "Proc. Mathematical Foundations of Computer Science" (L. Brim, J. Gruska, and J. Zlatuska, Eds.), Lecture Notes in Computer Science, Vol. 1450, pp. 346–355, Springer-Verlag, Berlin.
3. Hopcroft, J. E., and Ullman, J. D. (1979), "Introduction to Automata Theory, Languages and Computation," Addison-Wesley, Reading, MA.
4. Jim, T. (1995), System F plus subsumption reduces to Mitchell's subtyping relation, manuscript.
5. Longo, G., Milsted, K., and Soloviev, S. (1995), A logic of subtyping, *in* "Proc. 10th IEEE Symp. Logic in Computer Science, San Diego," pp. 292–299.
6. Mitchell, J. C. (1988), Polymorphic type inference and containment, *Inform. and Comput.* **76**, 211–249.
7. Odersky, M., and Läufer, K. (1995), Putting type annotations to work, *presented at* the Newton Institute Workshop Advances in Type Systems for Computing, August 1995.
8. Pierce, B. C. (1994), Bounded quantification is undecidable, *Inform. and Comput.* **112**, 131–165.
9. Tiuryn, J. (1995), Equational axiomatization of bicoercibility for polymorphic types, *in* "Proc. Conf. Foundations of Software Technology and Teoretical Computer Science'95" (P. S. Thiagarajan, Ed.), Lecture Notes in Computer Science, Vol. 1026, pp. 166–179, Springer-Verlag, Berlin.
10. Wells, J. B. (1999), Typability and type checking in the second-order λ-calculus calculus are equivalent and undecidable, *Ann. Pure Appl. Logic* **98**, 111–156. (Preliminary version *in* "Proc. 9th IEEE Symposium on Logic in Computer Science, Paris, France, 1994," pp. 176–185.)
11. Wells, J. B. (1995), "The Undecidability of Mitchell's Subtyping Relationship," Technical Report 95-019, Computer Science Department, Boston University.
12. Wells, J. B. (1996), "Typability is Undecidable for F+Eta," Technical Report 96-022, Computer Science Department, Boston University.