NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Bounded Quantification is Undecidable

Benjamin C. Pierce
July 21, 1991
CMU-CS-91-161 2

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

$F_{\leq}$ is a typed λ-calculus with subtyping and bounded second-order polymorphism. First introduced by Cardelli and Wegner, it has been widely studied as a core calculus for type systems with subtyping.

Curien and Ghelli proved the partial correctness of a recursive procedure for computing minimal types of $F_{\leq}$ terms and showed that the termination of this procedure is equivalent to the termination of its major component, a procedure for checking the subtype relation between $F_{\leq}$ types. Ghelli later claimed that this procedure is also guaranteed to terminate, but the discovery of a subtle bug in his proof led him recently to observe that, in fact, there are inputs on which the subtyping procedure diverges. This reopens the question of the decidability of subtyping and hence of typechecking.

This question is settled here in the negative, using a reduction from the halting problem for two-counter Turing machines to show that the subtype relation of $F_{\leq}$ is undecidable.

This research was supported in part by the Office of Naval Research and in part by the Defense Advanced Research Projects Agency (DOD), monitored by the Office of Naval Research under Contract N00014-84-K-0415, ARPA Order No. 5404.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of ONR, DARPA or the U.S. government.

510.7808
C28r
91-161
c.2

Keywords: Lambda calculus and related systems, Language theory, Programming, Type structure, Data types and structures, Polymorphism, Subtyping, Bounded Quantification.

1 Introduction

The notion of *bounded quantification* was introduced by Cardelli and Wegner [14] in the language Fun. Based on informal ideas by Cardelli and formalized using techniques developed by Mitchell [27, 9], Fun combined Girard-Reynolds polymorphism [23, 29] and Cardelli's first-order calculus of subtyping [5, 6].

Fun and its relatives have been studied extensively by programming language theorists and designers. Cardelli and Wegner's survey paper gives the first programming examples using bounded quantification; more are developed in Cardelli's study of power kinds [7]. Curien and Ghelli [18] and Ghelli's Ph.D. thesis [21] address a number of syntactic properties of $F_{\leq}$. Semantic aspects of closely related systems have been studied by Bruce and Longo [3], Martini [26], Breazu-Tannen, Coquand, Gunter, and Scedrov [1], Cardone [15], Cardelli and Longo [11], Cardelli, Martini, Mitchell, and Scedrov [12], and Curien and Ghelli [18, 19]. $F_{\leq}$ has been extended to include record types and richer notions of inheritance by Cardelli and Mitchell [13], Bruce [2], Cardelli [10], and Canning, Cook, Hill, Olthoff, and Mitchell [4]; an extension with intersection types [17, 30] is the subject of the present author's Ph.D. thesis [28]. The proof theory of a version of $F_{\leq}$ with a rule of extensionality has been studied by Curien and Ghelli [19]. Bounded quantification also plays a key role in Cardelli's programing language Quest [8, 11] and in the Abel language developed at HP Labs [4, 16].

The original Fun was simplified by Bruce and Longo [3] for their inventigation of its semantics, and again by Curien and Ghelli [18], who gave a proof of the coherence of typechecking. Curien and Ghelli's formulation, called *minimal Bounded Fun* or $F_{\leq}$ ("F sub"), is the one considered here.

As in other second-order λ-calculi, the terms of $F_{\leq}$ include variables, abstractions, applications, type abstractions, and type applications, with the refinement that each type abstraction gives a *bound* for the type variable it introduces and each type application must satisfy the constraint that the argument type is a *subtype* of the bound of the polymorphic function being applied. The well-typed terms of $F_{\leq}$ are defined by means of a collection of rules for inferring statements of the form $\Gamma \vdash e \in \tau$ ("e has type τ in context Γ").

Variables, abstractions, and applications have the typing rules familiar from other λ-calculi:

$$\Gamma \vdash x \in \Gamma(x)$$

$$\frac{\Gamma,\ x{:}\sigma \vdash e \in \tau}{\Gamma \vdash \lambda x{:}\sigma.\ e \in \sigma \rightarrow \tau}$$

$$\frac{\Gamma \vdash e_1 \in \sigma \rightarrow \tau \qquad \Gamma \vdash e_2 \in \sigma}{\Gamma \vdash e_1\, e_2 \in \tau}$$

Type abstractions are treated as in other second-order λ-calculi, except that they also give a bound, with respect to the subtype relation, for the variable they introduce; they are checked by moving this assumption into the context and checking the body of the abstraction under the enriched set of assumptions:

$$\frac{\Gamma,\ \alpha{\leq}\theta \vdash e \in \tau}{\Gamma \vdash \Lambda\alpha{\leq}\theta.\ e \in \forall\alpha{\leq}\theta.\ \tau}$$

Type applications check that the type being passed as a parameter is indeed a subtype of the bound of the corresponding quantifier:

University Libraries
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

$$\frac{\Gamma \vdash e \in \forall\alpha{\leq}\theta.\ \tau \qquad \Gamma \vdash \sigma \leq \theta}{\Gamma \vdash e[\sigma] \in [\sigma/\alpha]\tau}$$

Finally, like other λ-calculi with subtyping, $F_{\leq}$ includes a rule of *subsumption*, which allows the type of a term to be promoted to any supertype:

$$\frac{\Gamma \vdash e \in \sigma \qquad \Gamma \vdash \sigma \leq \tau}{\Gamma \vdash e \in \tau}$$

The rules for type application and subsumption rely on a separately axiomatized subtype relation $\Gamma \vdash \sigma \leq \tau$ ("σ is a subtype of τ under assumptions Γ"). This relation, which forms the main object of study in the present paper, is presented as follows.

Subtyping is both reflexive and transitive:

$$\Gamma \vdash \tau \leq \tau$$

$$\frac{\Gamma \vdash \tau_1 \leq \tau_2 \qquad \Gamma \vdash \tau_2 \leq \tau_3}{\Gamma \vdash \tau_1 \leq \tau_3}$$

Every type is a subtype of a maximal type called *Top*:

$$\Gamma \vdash \sigma \leq \mathit{Top}$$

One of the main uses of *Top* (and the one for which it was introduced by Cardelli and Wegner) is to recover ordinary unbounded quantification as a special case of bounded quantification: $\forall\alpha.\ \tau$ becomes $\forall\alpha{\leq}\mathit{Top}.\ \tau$.

Type variables are subtypes of their bounds:

$$\Gamma \vdash \alpha \leq \Gamma(\alpha)$$

The subtype relation between arrow types is contravariant in their left-hand sides and covariant in their right-hand sides:

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1{\rightarrow}\sigma_2 \leq \tau_1{\rightarrow}\tau_2}$$

Similarly, subtyping of quantified types is covariant in their bounds and contravariant in their bodies:

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2}$$

The last rule deserves a closer look, since it causes all the trouble we will be discussing for the rest of the paper. Intuitively, it reads as follows:

> A type $\tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$ describes a collection of polymorphic values (functions from types to values) each mapping subtypes of τ_1 to instances of τ_2. If τ_1 is a subtype of σ_1, then the domain of τ is smaller than that of $\sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2$, so τ is a weaker constraint and describes a larger collection of polymorphic values. Moreover, if, for each type θ that is an acceptable argument to the functions in both collections (i.e., one that satisfies the more stringent requirement $\theta \leq \tau_1$), the corresponding instance of σ_2 is a subtype of the corresponding instance of τ_2, then τ is a "pointwise weaker" constraint and again describes a larger collection of polymorphic values.

Though semantically appealing, this rule creates serious problems for reasoning about the subtype relation. In a quantified type $\forall\alpha\leq\sigma_1.\ \sigma_2$, instances of α in σ_2 are naturally thought of as being bounded by their lexically declared bound σ_1. But this connection is destroyed by the second premise of the quantifier subtyping rule: when $\forall\alpha\leq\sigma_1.\ \sigma_2$ is compared to $\forall\alpha\leq\tau_1.\ \tau_2$, instances of α in *both* σ_2 and τ_2 are bounded by τ_1 in the premise $\Gamma, \alpha\leq\tau_1 \vdash \sigma_2 \leq \tau_2$. As we shall see, this "re-bounding" behavior makes it impossible to give a decision procedure for the subtype relation.

Cardelli and Wegner's definition of **Fun** [14] used a weaker quantifier subtyping rule in which $\forall\alpha\leq\sigma_1.\ \sigma_2$ is a subtype of $\forall\alpha\leq\tau_1.\ \tau_2$ only if σ_1 and τ_1 are identical. (This variant of the system can easily be shown to be decidable.) Later authors, including Cardelli, have chosen to work with the more powerful formulation given here.

Curien and Ghelli used a proof-normalization argument to show that $F_{\leq}$ typechecking is *coherent* — that is, that all derivations of a statement $\Gamma \vdash e \in \tau$ have the same meaning. One corollary of their proof is the soundness and completeness of a natural syntax-directed procedure for computing minimal typings of $F_{\leq}$ terms with a subroutine for checking the subtype relation; the same procedure had been developed by the group at Penn and by Cardelli for use in his sf Quest typechecker [24]. The termination of the Curien and Ghelli's typechecking procedure is equivalent to the termination of the subtyping algorithm. Ghelli, in his Ph.D. thesis [21], gave a proof of termination; unfortunately, this proof was later discovered — by Curien and Reynolds, independently — to contain a subtle mistake (see Appendix A). In fact, Ghelli soon realized that there are inputs for which the subtyping algorithm does *not* terminate [22]. Worse yet, these cases did not seem amenable to any simple form of cycle detection: when presented with one of them, the algorithm would generate an infinite sequence of different recursive calls with larger and larger contexts. This discovery reopened the question of the decidability of $F_{\leq}$.

The undecidability result presented here began as an attempt to formulate a more refined algorithm capable of detecting the kinds of divergence that could be induced in the simpler one. A series of partial results about decidable subsystems eventually led to the discovery of a class of input problems in which increasing the size the input by a constant factor would increase the search depth of a *succeeding* execution of the algorithm by an exponential factor. In addition to dispelling a number of intuitions about why the problem ought to be decidable, the technique used to construct this example suggested a trick for encoding natural numbers, from which it was a short step to an encoding of two-counter Turing machines.

After formally defining the $F_{\leq}$ subtype relation (Section 2), reviewing Curien and Ghelli's subtyping algorithm (Section 3), and presenting an example where the algorithm fails to terminate (Section 4), we identify a fragment of $F_{\leq}$ that forms a convenient target for the reductions to follow (Sections 5 and 6). The main result is then presented in two steps:

1. We first define an intermediate abstraction, called *rowing machines* (Section 7). These machines bridge the gap between $F_{\leq}$ subtyping problems and two-counter machines by retaining the notions of bound variables and substitution from $F_{\leq}$ while simultaneously introducing a computational abstraction with a finite collection of registers and an evaluation regime based on state transformation.

   An encoding of rowing machines as $F_{\leq}$ subtyping statements is given and proven correct in the sense that a rowing machine R halts iff its translation $\mathcal{F}(R)$ is a derivable statement in $F_{\leq}$ (Section 8).

2. We then review the definition of *two-counter machines* (Section 9) and show how a two-counter machine T may be encoded as a rowing machine $\mathcal{R}(T)$ such that T halts iff $\mathcal{R}(T)$ does (Section 10).

Section 11 shows that the undecidability of subtyping implies the undecidability of typechecking. Section 12 briefly discusses the pragmatic import of our results.

2 The Subtype Relation

We begin the detailed development of the undecidability of $F_{\leq}$ by establishing some notational conventions and defining the subtype relation formally.

2.1. Notation: We write $X \equiv Y$, where X and Y are types, contexts, statements, etc., to indicate that "X has the form Y." If Y contains free metavariables, then $X \equiv Y$ denotes pattern matching; for example

> "If $\tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2$, then ..."

means

> "If τ has the form $\forall\alpha{\leq}\tau_1.\ \tau_2$ for some α, τ_1, and τ_2, then ..."

2.2. Definition: The *types* of $F_{\leq}$ are defined by the following abstract grammar:

$$
\begin{array}{lcll}
\tau & ::= & \alpha & \text{type variables} \\
 & | & \tau_1 \rightarrow \tau_2 & \text{function types} \\
 & | & \forall\alpha{\leq}\tau_1.\ \tau_2 & \text{bounded quantifiers} \\
 & | & \mathit{Top} & \text{top type.}
\end{array}
$$

2.3. Definition: *Typing contexts* in $F_{\leq}$ are lists of type variables and associated bounds,

$$\Gamma \quad ::= \quad \mathit{empty} \quad | \quad \Gamma, \alpha{\leq}\tau$$

with all variables distinct. (If we were dealing formally with the $F_{\leq}$ typing relation, we would also need bindings of the form $x{:}\tau$.)

The comma operator is used to denote both extension $(\Gamma, \alpha{\leq}\tau)$ and concatenation (Γ_1, Γ_2) of contexts. The set of variables bound by a context Γ is written $\mathit{dom}(\Gamma)$. When $\Gamma \equiv \Gamma_1, \alpha{\leq}\tau, \Gamma_2$, we call τ the *bound* of α in Γ and write $\tau = \Gamma(\alpha)$.

2.4. Definition: A *subtyping statement* is a phrase of the form

$$\Gamma \vdash \sigma \leq \tau.$$

The portion of a statement to the right of the turnstile is called the *body*.

2.5. Definition: The set of free type variables in a type τ is written $FTV(\tau)$.

2.6. Definition: A type τ is *closed* with respect to a context Γ if $FTV(\tau) \subseteq \mathit{dom}(\Gamma)$. A context Γ is closed if

1. $\Gamma \equiv \mathit{empty}$, or
2. $\Gamma \equiv \Gamma_1, \alpha{\leq}\tau$, with Γ_1 closed and τ closed with respect to Γ_1.

A statement $\Gamma \vdash \sigma \leq \tau$ is closed if Γ is closed and σ and τ are closed with respect to Γ.

2.7. Convention: In the following, we assume that all statements under discussion are closed. In particular, we allow only closed statements in instances of inference rules.

2.8. Convention: The metavariables σ, τ, θ, and ϕ range over types; α, β, and γ range over type variables; Γ ranges over contexts; J ranges over (closed) statements.

2.9. Definition: $F_{\leq}$ is the least three-place relation closed under the following rules:

$$\Gamma \vdash \tau \leq \tau \qquad (\text{REFL})$$

$$\frac{\Gamma \vdash \tau_1 \leq \tau_2 \qquad \Gamma \vdash \tau_2 \leq \tau_3}{\Gamma \vdash \tau_1 \leq \tau_3} \qquad (\text{TRANS})$$

$$\Gamma \vdash \sigma \leq \mathit{Top} \qquad (\text{TOP})$$

$$\Gamma \vdash \alpha \leq \Gamma(\alpha) \qquad (\text{VAR})$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1 \rightarrow \sigma_2 \leq \tau_1 \rightarrow \tau_2} \qquad (\text{ARROW})$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad (\text{ALL})$$

2.10. Convention: Types, contexts, and statements that differ only in the names of bound variables are considered to be identical. (In a statement $\Gamma_1, \alpha{\leq}\theta, \Gamma_2 \vdash \sigma \leq \tau$, the variable α is bound in Γ_2, σ, and τ.)

It is formally clearer to think of variables not as names but, as suggested by deBruijn [20], as pointers into the surrounding context. This point of view is notationally too inconvenient to adopt explicitly in what follows, but can be a significant aid in understanding the behavior of the rules here (VAR and ALL) that manipulate variables. It is developed in detail in Appendix B.

2.11. Definition: The capture-avoiding substitution of σ for α in τ is written $[\sigma/\alpha]\tau$. Substitution is extended pointwise to contexts: $[\sigma/\alpha]\Gamma$.

2.12. Definition: The *positive* and *negative occurrences* in a statement $\Gamma \vdash \sigma \leq \tau$ are defined as follows.

- The bounds in Γ and the type σ are negative occurrences; τ is a positive occurrence.
- If $\tau_1 \rightarrow \tau_2$ is a positive (resp. negative) occurrence, then τ_1 is a negative (positive) occurrence and τ_2 is a positive (negative) occurrence.
- If $\forall\alpha{\leq}\tau_1.\tau_2$ is a positive (resp. negative) occurrence, then τ_1 is a negative (positive) occurrence and τ_2 is a positive (negative) occurrence.

2.13. Fact: The rules defining $F_{\leq}$ preserve the signs of occurrences: if a metavariable τ appears in a premise of one of the rules, then it has the same sign as the corresponding occurrence of τ in the conclusion.

2.14. Definition: In the examples below, it will be convenient to rely on a few abbreviations:

$$\begin{array}{lcl} \forall\alpha.\ \tau & \stackrel{\text{def}}{=} & \forall\alpha{\leq}Top.\ \tau \\ \forall\alpha_1{\leq}\phi_1..\alpha_n{\leq}\phi_n.\ \tau & \stackrel{\text{def}}{=} & \forall\alpha_1{\leq}\phi_1.\ ..\ \forall\alpha_n{\leq}\phi_n.\ \tau \\ \neg\tau & \stackrel{\text{def}}{=} & \forall\alpha{\leq}\tau.\ \alpha \end{array}$$

The salient property of the last of these is that it allows the right- and left-hand sides of subtyping statements to be swapped:

2.15. Fact: $\Gamma \vdash \neg\sigma \leq \neg\tau$ is derivable iff $\Gamma \vdash \tau \leq \sigma$ is.

3 A Subtyping Algorithm

The rules defining $F_\leq$ do not constitute an algorithm for checking the subtype relation, since they are not syntax-directed. In particular, the rule TRANS cannot effectively be applied backwards, since this would involve "guessing" an appropriate intermediate type τ_2. Curien and Ghelli (as well as Cardelli and others) use the following reformulation:

3.1. Definition: $F_\leq^N$ (N for normal form) is the least relation closed under the following rules:

$$\Gamma \vdash \sigma \leq Top \qquad \text{(NTOP)}$$

$$\Gamma \vdash \alpha \leq \alpha \qquad \text{(NREFL)}$$

$$\frac{\Gamma \vdash \Gamma(\alpha) \leq \tau}{\Gamma \vdash \alpha \leq \tau} \qquad \text{(NVAR)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1{\rightarrow}\sigma_2 \leq \tau_1{\rightarrow}\tau_2} \qquad \text{(NARROW)}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall\alpha{\leq}\sigma_1.\ \sigma_2 \leq \forall\alpha{\leq}\tau_1.\ \tau_2} \qquad \text{(NALL)}$$

The reflexivity rule here is restricted to type variables. Transitivity is eliminated, except for instances of the form

$$\frac{\Gamma \vdash \alpha \leq \Gamma(\alpha) \qquad \Gamma \vdash \Gamma(\alpha) \leq \tau}{\Gamma \vdash \alpha \leq \tau,}$$

which are packaged together as instances of the new rule NVAR.

3.2. Lemma: [Curien and Ghelli] The relations $F_\leq$ and $F_\leq^N$ coincide: $\Gamma \vdash \sigma \leq \tau$ is derivable in $F_\leq$ iff it is derivable in $F_\leq^N$.

3.3. Definition: The rules defining $F^N_{\leq}$ may be read as an algorithm (i.e., a recursively defined procedure, not necessarily always terminating) for checking the subtype relation:

$$
\begin{array}{l}
\mathit{check}(\Gamma \vdash \sigma \leq \tau) = \\
\quad \mathit{if}\ \tau \equiv \mathit{Top} \\
\quad\quad \mathit{then\ true} \\
\quad \mathit{else\ if}\ \sigma \equiv \sigma_1 \rightarrow \sigma_2\ \mathit{and}\ \tau \equiv \tau_1 \rightarrow \tau_2 \\
\quad\quad \mathit{then}\quad\quad \mathit{check}(\Gamma \vdash \tau_1 \leq \sigma_1) \\
\quad\quad\quad\quad \mathit{and}\ \mathit{check}(\Gamma \vdash \sigma_2 \leq \tau_2) \\
\quad \mathit{else\ if}\ \sigma \equiv \forall\alpha{\leq}\sigma_1.\ \sigma_2\ \mathit{and}\ \tau \equiv \forall\alpha{\leq}\tau_1.\ \tau_2 \\
\quad\quad \mathit{then}\quad\quad \mathit{check}(\Gamma \vdash \tau_1 \leq \sigma_1) \\
\quad\quad\quad\quad \mathit{and}\ \mathit{check}(\Gamma, \alpha{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2) \\
\quad \mathit{else\ if}\ \sigma \equiv \alpha\ \mathit{and}\ \tau \equiv \alpha \\
\quad\quad \mathit{then\ true} \\
\quad \mathit{else\ if}\ \sigma \equiv \alpha \\
\quad\quad \mathit{then}\ \mathit{check}(\Gamma \vdash \Gamma(\alpha) \leq \tau) \\
\quad \mathit{else} \\
\quad\quad \mathit{false}.
\end{array}
$$

We write $F^N_{\leq}$ to refer either to the algorithm or to the inference system, depending on context.

This algorithm may be thought of as incrementally building (or attempting to build) a normal form derivation of a statement J, starting from the root and recursively building subderivations for the premises. By Lemma 3.2, if there is any derivation whatsoever of a statement J, there is one in normal form; the algorithm is guaranteed to recapitulate this derivation and halt in finite time.

3.4. Fact: [Curien and Ghelli] If $\Gamma \vdash \sigma \leq \tau$ is derivable in $F^N_{\leq}$, then the algorithm $F^N_{\leq}$ halts and returns *true* when given this statement as input.

An easy corollary is that if $\Gamma \vdash \sigma \leq \tau$ is not derivable and the algorithm halts, it will correctly return *false*, having verified that there is no normal form derivation of this statement. Another corollary, more important for our purposes, is the following:

3.5. Lemma: If the algorithm $F^N_{\leq}$ fails to terminate on input $\Gamma \vdash \sigma \leq \tau$, then $\Gamma \vdash \sigma \leq \tau$ is not derivable in $F_{\leq}$.

4 Nontermination of the Algorithm

Ghelli recently dispelled the widely held belief that the algorithm $F^N_{\leq}$ terminates on all inputs by discovering the following example.

4.1. Example: Let

$$\theta \equiv \forall\alpha.\ \neg(\forall\beta{\leq}\alpha.\ \neg\alpha_2).$$

Then executing the algorithm $F^N_{\leq}$ on the input problem

$$\alpha_0{\leq}\theta \vdash \alpha_0 \leq (\forall\alpha_1{\leq}\alpha_0.\ \neg\alpha_1)$$

leads to the following infinite sequence of recursive calls:

$$
\begin{array}{llcl}
\alpha_0{\leq}\theta & \vdash \alpha_0 & \leq & \forall\alpha_1{\leq}\alpha_0.\ \neg\alpha_1 \\
\alpha_0{\leq}\theta & \vdash \forall\alpha_1.\ \neg(\forall\alpha_2{\leq}\alpha_1.\ \neg\alpha_2) & \leq & \forall\alpha_1{\leq}\alpha_0.\ \neg\alpha_1 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0 & \vdash \neg(\forall\alpha_2{\leq}\alpha_1.\ \neg\alpha_2) & \leq & \neg\alpha_1 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0 & \vdash \alpha_1 & \leq & \forall\alpha_2{\leq}\alpha_1.\ \neg\alpha_2 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0 & \vdash \alpha_0 & \leq & \forall\alpha_2{\leq}\alpha_1.\ \neg\alpha_2 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0 & \vdash \forall\alpha_2.\ \neg(\forall\alpha_3{\leq}\alpha_2.\ \neg\alpha_3) & \leq & \forall\alpha_2{\leq}\alpha_1.\ \neg\alpha_2 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0, \alpha_2{\leq}\alpha_1 & \vdash \neg(\forall\alpha_3{\leq}\alpha_2.\ \neg\alpha_3) & \leq & \neg\alpha_2 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0, \alpha_2{\leq}\alpha_1 & \vdash \alpha_2 & \leq & \forall\alpha_3{\leq}\alpha_2.\ \neg\alpha_3 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0, \alpha_2{\leq}\alpha_1 & \vdash \alpha_1 & \leq & \forall\alpha_3{\leq}\alpha_2.\ \neg\alpha_3 \\
\alpha_0{\leq}\theta, \alpha_1{\leq}\alpha_0, \alpha_2{\leq}\alpha_1 & \vdash \alpha_0 & \leq & \forall\alpha_3{\leq}\alpha_2.\ \neg\alpha_3 \\
\text{etc.} & & &
\end{array}
$$

(The α-conversion steps necessary to maintain the well-formedness of the context when new variables are added are performed tacitly here, choosing new names so as to clarify the pattern of infinite regress. Appendix B presents the same example using deBruijn indices instead of named variables.)

5 A Deterministic Fragment of $F_{\leq}$

The pattern of recursion in Ghelli's example is, in fact, an instance of a more general scheme — one so general that it can be used to encode termination problems for two-counter Turing machines. We now turn our attention to demonstrating this fact.

In what follows, it will be convenient to work with a fragment of $F_{\leq}$ with somewhat simpler behavior. In particular:

- we drop the $\rightarrow$ type constructor and its subtyping rule;
- we introduce a negation operator explicitly into the syntax and include a rule for comparing negated expressions;
- we drop the left-hand premise from the rule for comparing quantifiers, requiring instead that when two quantified types are compared, the bound of the one on the left must be *Top*;
- we consider only statements where no variable occurs positively, allowing us to drop the REFL rule.

Since the $F_{\leq}^{N}$ rules preserve positive and negative occurrences, we may redefine the set of types so that positive and negative types (i.e. those that may appear in positive and negative positions) are separate syntactic categories. At the same time, we simplify each category appropriately.

5.1. Definition: The sets of *positive types* τ_p and *negative types* τ_n are defined by the following abstract grammar:

$$
\begin{array}{lcl}
\tau_p & ::= & Top \quad | \quad \neg\tau_n \quad | \quad \forall\alpha{\leq}\tau_n.\ \tau_p \\
\tau_n & ::= & \alpha \quad | \quad \neg\tau_p \quad | \quad \forall\alpha.\ \tau_n
\end{array}
$$

A *negative context* Γ_n is one whose bounds are all negative types.

5.2. Definition: $F_{\leq}^{D}$ (D for deterministic) is the least relation closed under the following rules:

$$\Gamma_n \vdash \tau_n \leq Top \qquad \text{(DTOP)}$$

$$\frac{\Gamma_n \vdash \Gamma(\alpha) \leq \tau_p}{\Gamma_n \vdash \alpha \leq \tau_p} \qquad \text{(DVAR)}$$

$$\frac{\Gamma_n,\, \alpha{\leq}\phi_n \vdash \sigma_n \leq \tau_p}{\Gamma_n \vdash \forall\alpha.\ \sigma_n \leq \forall\alpha{\leq}\phi_n.\ \tau_p} \qquad \text{(DALL)}$$

$$\frac{\Gamma_n \vdash \tau_n \leq \sigma_p}{\Gamma_n \vdash \neg\sigma_p \leq \neg\tau_n} \qquad \text{(DNEG)}$$

5.3. Convention: To reduce clutter, we drop the subscripts p and n below.

Using the earlier abbreviations for negation and unbounded quantification, we may read every $F_{\leq}^{D}$-statement as an $F_{\leq}^{N}$-statement. Under this interpretation, the two relations coincide for statements in their common domain.

5.4. Lemma: $F_{\leq}^{N}$ is a conservative extension of $F_{\leq}^{D}$: if J is an $F_{\leq}^{D}$-statement, then J is derivable in $F_{\leq}^{D}$ iff it is derivable in $F_{\leq}^{N}$.

Proof: ($\Longrightarrow$) An $F_{\leq}^{D}$ derivation may be transformed straightforwardly into an $F_{\leq}^{N}$ derivation with the same conclusion (modulo abbreviations): if J is an instance of DTOP, then it is an instance of NTOP; if J is an instance of one of the other three $F_{\leq}^{D}$ rules and J' is the corresponding premise, then the same instance of the appropriate $F_{\leq}^{N}$ rule has J' as one of its premises and the other premise (if any) may be proved in one step using either NTOP or NREFL.

($\Longleftarrow$) Similarly, an $F_{\leq}^{N}$ derivation of an $F_{\leq}^{D}$ statement may be transformed into an $F_{\leq}^{D}$ derivation of the same statement. □

These simplifications justify a useful change of perspective. Since the only rule in $F_{\leq}^{N}$ with two premises has been replaced by two rules with one premise each, derivations in this fragment are linear: each node has at most one subderivation. The syntax-directed construction of such a derivation may be viewed as a deterministic state transformation process, where the subtyping statement being verified is the current state and the single premise that must be recursively verified (if there is one) is the next state. In other words, a subtyping statement may be thought of as an instantaneous description of a kind of automaton.

From now on we use terminology that makes the intuition of "subtyping as state transformation" more explicit. Analogous notation will be used to describe the execution behavior of the other calculi introduced below.

5.5. Definition: The *one-step elaboration* function $\mathcal{E}$ for $F_{\leq}^{D}$-statements is the partial mapping defined by:

$$\mathcal{E}(J) = \begin{cases} J' & \text{if } J \text{ is the conclusion of an instance of DVAR, DALL, or DNEG} \\ & \text{and } J' \text{ is the corresponding premise} \\ \text{undefined} & \text{if } J \text{ is an instance of DTOP.} \end{cases}$$

5.6. Definition: J' is an *immediate subproblem* of J in $F_{\leq}^{D}$, written $J \longrightarrow_D J'$, if $J' = \mathcal{E}(J)$.

5.7. Definition: J' is a *subproblem* of J in $F_{\leq}^{D}$, written $J \stackrel{*}{\longrightarrow}_D J'$, if either $J \equiv J'$ or $J \longrightarrow_D J_1$ and $J_1 \stackrel{*}{\longrightarrow}_D J'$.

5.8. Definition: The *elaboration* of a statement J is the sequence of subproblems encountered by the subtyping algorithm given J as input.

6 Eager Substitution

To make a smooth transition between the subtyping statements of $F_{\leq}$ and the rowing machine abstraction to be introduced in Section 7, we need one more variation in the definition of subtyping, where, instead of maintaining a context with the bounds of free variables, the bounds are immediately substituted into the body of the statement by the quantifier rule.

6.1. Definition: $F^F_{\leq}$ (F for flattened) is the least relation closed under the following rules:

$$\vdash \tau \leq Top \qquad \text{(FTop)}$$

$$\frac{\vdash [\phi/\alpha]\sigma \leq [\phi/\alpha]\tau}{\vdash \forall\alpha.\ \sigma \leq \forall\alpha{\leq}\phi.\ \tau} \qquad \text{(FAll)}$$

$$\frac{\vdash \tau \leq \sigma}{\vdash \neg\sigma \leq \neg\tau} \qquad \text{(FNeg)}$$

An analogous reformulation of full $F_{\leq}$ would not be correct. For example, in the non-derivable statement

$$\vdash (\forall\alpha{\leq}Top.\ Top) \leq (\forall\alpha{\leq}Top.\ \alpha)$$

substituting the bound Top for α in the body of the quantifiers yields the derivable subproblem

$$\vdash Top \leq Top.$$

But having restricted our attention to statements where variables appear only negatively, we are guaranteed that the only position where the elaboration of a statement can cause a variable to appear by itself in the body of a subproblem is the left-hand side, where it will immediately be replaced by its bound. We are therefore safe in making the substitution eagerly.

In the remainder of this section, we show that $F^D_{\leq}$ is a conservative extension of $F^F_{\leq}$.

6.2. Lemma: Let ϕ be a negative type and assume that the statement $\alpha{\leq}\phi, \Gamma \vdash \sigma \leq \tau$ is closed. Then if $[\phi/\alpha]\Gamma \vdash [\phi/\alpha]\sigma \leq [\phi/\alpha]\tau$ is derivable in $F^D_{\leq}$, so is $\alpha{\leq}\phi, \Gamma \vdash \sigma \leq \tau$.

Proof: By induction on the size of the given derivation.

Case DTop: $[\phi/\alpha]\tau \equiv Top$

Since variables can only occur negatively, τ cannot be a variable, so $\tau \equiv Top$ and the result is immediate.

Case DVar: $[\phi/\alpha]\sigma \equiv \beta$

We may assume that $\sigma \not\equiv \alpha$, since otherwise we would have $\phi \equiv \beta$ and the statement $[\phi/\alpha]\Gamma \vdash [\phi/\alpha]\sigma \leq [\phi/\alpha]\tau$ would not be closed. So σ must itself be β. By assumption, we have a subderivation

$$[\phi/\alpha]\Gamma \vdash ([\phi/\alpha]\Gamma)(\beta) \leq [\phi/\alpha]\tau.$$

that is,

$$[\phi/\alpha]\Gamma \vdash [\phi/\alpha](\Gamma(\beta)) \leq [\phi/\alpha]\tau.$$

By the induction hypothesis,

$$\alpha \leq \phi, \Gamma \vdash \Gamma(\beta) \leq \tau.$$

By DVAR,

$$\alpha \leq \phi, \Gamma \vdash \beta \leq \tau.$$

Case DALL: $[\phi/\alpha]\sigma \equiv \forall\beta.\ \sigma_2' \qquad [\phi/\alpha]\tau \equiv \forall\beta{\leq}\tau_1'.\ \tau_2'$

Since τ cannot be a variable, we have

$$\begin{array}{lcl} \tau & \equiv & \forall\beta{\leq}\tau_1.\ \tau_2 \\ \tau_1' & \equiv & [\phi/\alpha]\tau_1 \\ \tau_2' & \equiv & [\phi/\alpha]\tau_2. \end{array}$$

For σ, there are two cases to consider:

Subcase: $\sigma \equiv \alpha$

Then

$$\phi \equiv \forall\beta.\ \sigma_2'.$$

By assumption, there is a subderivation

$$[\phi/\alpha]\Gamma, \beta{\leq}[\phi/\alpha]\tau_1 \vdash \sigma_2' \leq \tau_2'.$$

i.e. (since $\alpha \notin FTV(\phi)$),

$$[\phi/\alpha]\Gamma, \beta{\leq}[\phi/\alpha]\tau_1 \vdash [\phi/\alpha]\sigma_2' \leq [\phi/\alpha]\tau_2.$$

By the induction hypothesis,

$$\alpha \leq \phi, \Gamma, \beta{\leq}\tau_1 \vdash \sigma_2' \leq \tau_2.$$

By DALL,

$$\alpha \leq \phi, \Gamma \vdash \forall\beta.\ \sigma_2' \leq \forall\beta{\leq}\tau_1.\ \tau_2.$$

By DVAR,

$$\alpha \leq \phi, \Gamma \vdash \alpha \leq \forall\beta{\leq}\tau_1.\ \tau_2.$$

Subcase: $\sigma \not\equiv \alpha$

Then

$$\begin{array}{lcl} \sigma & \equiv & \forall\beta.\ \sigma_2 \\ \sigma_2' & \equiv & [\phi/\alpha]\sigma_2. \end{array}$$

By assumption, we again have a subderivation

$$[\phi/\alpha]\Gamma, \beta{\leq}[\phi/\alpha]\tau_1 \vdash \sigma_2' \leq \tau_2'.$$

that is,

$$[\phi/\alpha]\Gamma, \beta{\leq}[\phi/\alpha]\tau_1 \vdash [\phi/\alpha]\sigma_2 \leq [\phi/\alpha]\tau_2.$$

By the induction hypothesis,

$$\alpha \leq \phi, \Gamma, \beta{\leq}\tau_1 \vdash \sigma_2 \leq \tau_2.$$

By DALL,

$$\alpha \leq \phi, \Gamma \vdash \forall\beta.\ \sigma_2 \leq \forall\beta{\leq}\tau_1.\ \tau_2.$$

Case DNEG: $[\phi/\alpha]\sigma \equiv \neg(\sigma_1') \qquad [\phi/\alpha]\tau \equiv \neg(\tau_1')$

Similar. □

6.3. Lemma: If $\vdash \sigma \leq \tau$ is derivable in $F_{\leq}^{F}$, then it is derivable in $F_{\leq}^{D}$.

Proof: By induction on the original derivation, using Lemma 6.2 for the FALL case. □

6.4. Lemma: If $\alpha{\leq}\phi, \Gamma \vdash \sigma \leq \tau$ is derivable in $F^{D}_{\leq}$, then $[\phi/\alpha]\Gamma \vdash [\phi/\alpha]\sigma \leq [\phi/\alpha]\tau$ has an $F^{D}_{\leq}$-derivation of equal or lesser size.

Proof: By straightforward induction on the given derivation. □

6.5. Lemma: If $\vdash \sigma \leq \tau$ is an $F^{F}_{\leq}$-statement and is derivable in $F^{D}_{\leq}$, then it is derivable in $F^{F}_{\leq}$.

Proof: Induction on the size of the original derivation, using Lemma 6.4 for the DALL case. □

6.6. Lemma: $F^{D}_{\leq}$ is a conservative extension of $F^{F}_{\leq}$.

Proof: By Lemmas 6.3 and 6.5. □

7 Rowing Machines

The reduction from two-counter turing machines to $F_{\leq}$ subtyping statements is easiest to understand in terms of an intermediate abstraction called a rowing machine that makes more stylized use of bound variables.

A rowing machine is a tuple of *registers*

$$\langle \rho_1 \,..\, \rho_n \rangle,$$

where the contents of each register is a *row*. By convention, the first register is the machine's *program counter* (or *PC*). To move to the next state, the *PC* is used as a template to construct the new contents of each of the registers from the current contents of all of the registers (including the *PC*).

7.1. Definition: The set of *rows* (of width n) is defined by the following abstract grammar:

$$\begin{array}{lcll} \rho & ::= & \alpha_m & \text{for } 1 \leq m \leq n \\ & | & [\alpha_1..\alpha_n]\langle\rho_1..\rho_n\rangle & \\ & | & \text{HALT} & \end{array}$$

The variables $\alpha_1..\alpha_n$ in $[\alpha_1..\alpha_n]\langle\rho_1..\rho_n\rangle$ are binding occurrences whose scope is the rows ρ_1 through ρ_n. We regard rows that differ only in the names of bound variables as identical.

7.2. Definition: A *rowing machine* (of width n) is a tuple $\langle\rho_1..\rho_n\rangle$, where each ρ_i is a row of width n with no free variables.

7.3. Definition: The *one-step elaboration* function $\mathcal{E}$ for rowing machines of width n is the partial mapping

$$\mathcal{E}(\langle\rho_1..\rho_n\rangle) = \begin{cases} \langle[\rho_1/\alpha_1 \,..\, \rho_n/\alpha_n]\rho_{11} \,..\, [\rho_1/\alpha_1 \,..\, \rho_n/\alpha_n]\rho_{1n}\rangle & \text{if } \rho_1 = [\alpha_1..\alpha_n]\langle\rho_{11}..\rho_{1n}\rangle \\ \text{undefined} & \text{if } \rho_1 = \text{HALT}. \end{cases}$$

(Since rowing machines consist only of closed rows, we do not need to define the evaluation relation for the case where the *PC* is a variable.)

7.4. Notational conventions:

1. When the symbol "—" appears as the ith component of a compound row $[\alpha_1..\alpha_n]\langle\rho_1..\rho_n\rangle$, it stands for the variable α_i.

2. To avoid a proliferation of variable names in the examples and definitions below, we sometimes use numerical indices (like deBruijn indices; see Appendix B) rather than names for variables: the "variable" $\#n$ refers to the n^{th} bound variable of the row in which it appears; $\#\#n$ refers to the n^{th} bound variable of the row enclosing the one in which it appears; and so on.

   For example, the row

   $$[\alpha_1..\alpha_3]\langle\alpha_1, [\beta_1..\beta_3]\langle\alpha_1, \beta_1, \beta_3\rangle, \alpha_1\rangle$$

   would be abbreviated

   $$\langle —, \langle \#\#1, \#1, —\rangle, \#1\rangle.$$

3. It is convenient to introduce names for rows and use these to build up descriptions of other rows. For example, the compound row

   $$\langle\langle\langle \#1, \#1, \#1\rangle, \#3, \#2\rangle, \langle —, —, —\rangle, \langle \#1, \#1, \#1\rangle\rangle$$

   might be written as

   $$\langle \mathrm{Z}, \mathrm{Y}, \mathrm{X}\rangle,$$

   where

   $$\begin{aligned} \mathrm{X} &\equiv \langle \#1, \#1, \#1\rangle \\ \mathrm{Y} &\equiv \langle —, —, —\rangle \\ \mathrm{Z} &\equiv \langle \mathrm{X}, \#3, \#2\rangle. \end{aligned}$$

7.5. Definition: A rowing machine R *halts* if there is a machine R' such that $R \xrightarrow{*}_R R'$ and the PC of R' contains the instruction HALT.

7.6. Example: The simplest rowing machine, $\langle$HALT$\rangle$, halts immediately. The next simplest, $\langle\langle$HALT$\rangle\rangle$, takes one step and then halts. Another simple one, $\langle\langle —\rangle\rangle$, leads to an infinite elaboration with every state identical to the first.

7.7. Example: The machine

$$\langle \text{LOOP}, \text{A}, \text{B}\rangle,$$

where

$$\begin{aligned} \text{LOOP} &\equiv \langle —, \#3, \#2\rangle \\ \text{A} &\equiv \text{an arbitrary row} \\ \text{B} &\equiv \text{an arbitrary row} \end{aligned}$$

executes an infinite loop where the contents of the second and third register are exchanged at successive steps:

$$\begin{aligned} & \langle \text{LOOP}, \text{A}, \text{B}\rangle \\ \longrightarrow_R\ & \langle \text{LOOP}, \text{B}, \text{A}\rangle \\ \longrightarrow_R\ & \langle \text{LOOP}, \text{A}, \text{B}\rangle \\ \longrightarrow_R\ & \cdots \end{aligned}$$

7.8. Example: The row

$$\text{BRI} \equiv \langle \#2, - \rangle$$

encodes an *indirect branch* to the contents of register 2 at the moment when BRI is executed. The machine

$$\langle \text{BRI}, \langle \text{BRI}, \langle \text{BRI}, \text{HALT} \rangle \rangle \rangle$$

elaborates as follows:

$$\begin{array}{ll} & \langle \text{BRI}, \langle \text{BRI}, \langle \text{BRI}, \text{HALT} \rangle \rangle \rangle \\ \longrightarrow_R & \langle \langle \text{BRI}, \langle \text{BRI}, \text{HALT} \rangle \rangle, \langle \text{BRI}, \langle \text{BRI}, \text{HALT} \rangle \rangle \rangle \\ \longrightarrow_R & \langle \text{BRI}, \langle \text{BRI}, \text{HALT} \rangle \rangle \\ \longrightarrow_R & \langle \langle \text{BRI}, \text{HALT} \rangle, \langle \text{BRI}, \text{HALT} \rangle \rangle \\ \longrightarrow_R & \langle \text{BRI}, \text{HALT} \rangle \\ \longrightarrow_R & \langle \text{HALT}, \text{HALT} \rangle. \end{array}$$

8 Encoding Rowing Machines as Subtyping Problems

We now show how a rowing machine R can be encoded as a subtyping problem $\mathcal{F}(R)$ such that R halts iff $\mathcal{F}(R)$ is derivable in $F_{\leq}^{F}$.

The idea of the translation is that a rowing machine $R = \langle \rho_1..\rho_n \rangle$ becomes a subtyping statement of the form

$$\vdash \ldots \leq (\ldots \mathcal{F}(\rho_1) \ldots),$$

constructed so that

- if $\rho_1 = \text{HALT}$, the elaboration of $\mathcal{F}(R)$ halts after reaching a subproblem where *Top* appears on the right-hand side:
- if $\rho_1 = [\alpha_1..\alpha_n]\langle \rho_{11}..\rho_{1n} \rangle$, the elaboration of $\mathcal{F}(R)$ reaches a subproblem that encodes the rowing machine $\langle [\rho_1/\alpha_1 \,..\, \rho_n/\alpha_n]\,\rho_{11} \,..\, [\rho_1/\alpha_1 \,..\, \rho_n/\alpha_n]\,\rho_{1n} \rangle$.

In more detail, if $R = \langle [\alpha_1..\alpha_n]\langle \rho_{11}..\rho_{1n} \rangle .. \rho_n \rangle$, then $\mathcal{F}(R)$ is roughly the following:

$$\begin{array}{lll} \vdash & \forall \gamma_1..\gamma_n. & \neg(\forall \gamma_1' \leq \gamma_1 \,..\, \gamma_n' \leq \gamma_n.\ \neg \ldots \qquad\qquad) \\ \leq & \forall \gamma_1 \leq \mathcal{F}(\rho_1) \,..\, \gamma_n \leq \mathcal{F}(\rho_n).\ \neg(\forall \alpha_1..\alpha_n. & \neg(\forall \alpha_1' \leq \mathcal{F}(\rho_{11}) \,..\, \alpha_n' \leq \mathcal{F}(\rho_{1n}).\ \neg \mathcal{F}(\rho_{11}))). \end{array}$$

The elaboration of this statement proceeds as follows:

1. The current contents of the registers $\rho_1 \,..\, \rho_n$ are temporarily saved by matching the quantifiers on the right with the ones on the left; this has the effect of substituting the bounds $\mathcal{F}(\rho_1) \,..\, \mathcal{F}(\rho_n)$ for free occurrences of the variables $\gamma_1 \,..\, \gamma_n$ on the left-hand side.

2. The right- and left-hand sides are swapped (using a $\neg$ constructor on both sides), so that what now appears on the left is a sequence of variable bindings for the free variables $\alpha_1..\alpha_n$ of ρ_1:

$$\vdash (\forall \alpha_1 \,..\, \alpha_n.\ \ldots \mathcal{F}(\rho_{11})..\mathcal{F}(\rho_{1n}) \ldots) \leq (\forall \gamma_1' \leq \mathcal{F}(\rho_1) \,..\, \gamma_n' \leq \mathcal{F}(\rho_n).\ \ldots)$$

3. The saved contents of the original registers now appear on the right-hand side. When these are matched with the quantifiers on the left, the result is that the old values of the registers are substituted for the variables α_1 .. α_n in the body

$$\ldots \mathcal{F}(\rho_{11})..\mathcal{F}(\rho_{1n})\ldots$$

of the left-hand side.

4. Swapping right- and left-hand sides again yields a statement of the same form as the original, where the appropriate instances of $\mathcal{F}(\rho_{11})$ .. $\mathcal{F}(\rho_{1n})$ appear as the bounds of the outer quantifiers on the right:

$$\begin{aligned}\vdash \ldots \leq (\forall\gamma_1{\leq}[\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11}) \,..\\ \gamma_n{\leq}[\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{1n}).\\ \ldots[\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11})\ldots\end{aligned}$$

To be able to get back to a statement of the same form as the original, one piece of additional mechanism is required: besides the n variables used to store the old state of the registers, a variable γ_0 is used to hold the original value of the entire left-hand side of $\mathcal{F}(R)$. This variable is used at the end of a cycle to set up the left hand side of the statement encoding the next state of the encoded rowing machine.

The formal definition of the translation is as follows.

8.1. Definition: Let ρ be a row of width n. The *$F^F_\leq$-translation* of ρ, written $\mathcal{F}(\rho)$, is the negative type

$$\mathcal{F}(\rho) = \begin{cases} \alpha_i & \text{if } \rho = \alpha_i \\ \forall\gamma_0, \alpha_1 \,..\, \alpha_n.\ \neg\Big(\forall\gamma_0'{\leq}\gamma_0, \alpha_1'{\leq}\mathcal{F}(\rho_1) \,..\, \alpha_n'{\leq}\mathcal{F}(\rho_n).\ \neg\mathcal{F}(\rho_1)\Big) & \text{if } \rho = [\alpha_1..\alpha_n]\langle\rho_1..\rho_n\rangle \\ \forall\gamma_0, \alpha_1 \,..\, \alpha_n.\ \neg\, Top & \text{if } \rho = \text{HALT}, \end{cases}$$

where γ_0, γ_0', α_1' through α_n', and, in the third case, α_1 through α_n are fresh variables.

8.2. Definition: Let $R = \langle\rho_1..\rho_n\rangle$ be a rowing machine. The *$F^F_\leq$-translation* of R, written $\mathcal{F}(R)$, is the $F^F_\leq$ statement

$$\vdash\ \sigma\ \leq\ \forall\gamma_0{\leq}\sigma, \gamma_1{\leq}\mathcal{F}(\rho_1) \,..\, \gamma_n{\leq}\mathcal{F}(\rho_n).\ \neg\mathcal{F}(\rho_1),$$

where

$$\sigma \equiv \forall\gamma_0, \gamma_1..\gamma_n.\ \neg(\forall\gamma_0'{\leq}\gamma_0, \gamma_1'{\leq}\gamma_1 \,..\, \gamma_n'{\leq}\gamma_n.\ \neg\gamma_0).$$

8.3. Fact: This definition is proper — i.e., $\mathcal{F}(R)$ is a well-formed $F^F_\leq$-statement for every rowing machine R.

8.4. Lemma: If $R \longrightarrow_R R'$, then $\mathcal{F}(R) \overset{*}{\longrightarrow}_F \mathcal{F}(R')$.

Proof: By the definition of the elaboration function for rowing machines,

$$R \equiv \langle \rho_1 .. \rho_n \rangle,$$

where

$$\rho_1 \equiv [\alpha_1 .. \alpha_n]\langle \rho_{11} .. \rho_{1n} \rangle,$$

and

$$R' \equiv \langle [\rho_1/\alpha_1 \,..\, \rho_n/\alpha_n]\,\rho_{11} \,..\, [\rho_1/\alpha_1 \,..\, \rho_n/\alpha_n]\,\rho_{1n} \rangle.$$

Let

$$\sigma \equiv \forall \gamma_0, \gamma_1 .. \gamma_n .\ \neg(\forall \gamma_0' {\leq} \gamma_0, \gamma_1' {\leq} \gamma_1 \,..\, \gamma_n' {\leq} \gamma_n .\ \neg \gamma_0).$$

Now calculate as follows:

$$
\begin{array}{lll}
 & & \mathcal{F}(R) \\
\equiv & \vdash & \sigma \\
 & \leq & \forall \gamma_0 {\leq} \sigma, \gamma_1 {\leq} \mathcal{F}(\rho_1) \,..\, \gamma_n {\leq} \mathcal{F}(\rho_n).\ \neg \mathcal{F}(\rho_1) \\
\equiv & \vdash & \forall \gamma_0, \gamma_1 .. \gamma_n .\ \neg(\forall \gamma_0' {\leq} \gamma_0, \gamma_1' {\leq} \gamma_1 \,..\, \gamma_n' {\leq} \gamma_n .\ \neg \gamma_0) \\
 & \leq & \forall \gamma_0 {\leq} \sigma, \gamma_1 {\leq} \mathcal{F}(\rho_1) \,..\, \gamma_n {\leq} \mathcal{F}(\rho_n).\ \neg \mathcal{F}(\rho_1) \\
\xrightarrow{*}_F & \vdash & [\sigma/\gamma_0, \mathcal{F}(\rho_1)/\gamma_1 \,..\, \mathcal{F}(\rho_n)/\gamma_n]\,\neg(\forall \gamma_0' {\leq} \gamma_0, \gamma_1' {\leq} \gamma_1 \,..\, \gamma_n' {\leq} \gamma_n .\ \neg \gamma_0) \\
 & \leq & [\sigma/\gamma_0, \mathcal{F}(\rho_1)/\gamma_1 \,..\, \mathcal{F}(\rho_n)/\gamma_n]\,\neg \mathcal{F}(\rho_1) \\
\equiv & \vdash & \neg(\forall \gamma_0' {\leq} \sigma, \gamma_1' {\leq} \mathcal{F}(\rho_1) \,..\, \gamma_n' {\leq} \mathcal{F}(\rho_n).\ \neg \sigma) \\
 & \leq & \neg \mathcal{F}(\rho_1) \\
\longrightarrow_F & \vdash & \mathcal{F}(\rho_1) \\
 & \leq & \forall \gamma_0' {\leq} \sigma, \gamma_1' {\leq} \mathcal{F}(\rho_1) \,..\, \gamma_n' {\leq} \mathcal{F}(\rho_n).\ \neg \sigma \\
\equiv & \vdash & \forall \gamma_0, \alpha_1 .. \alpha_n .\ \neg(\forall \gamma_0' {\leq} \gamma_0, \alpha_1' {\leq} \mathcal{F}(\rho_{11}) \,..\, \alpha_n' {\leq} \mathcal{F}(\rho_{1n}).\ \neg \mathcal{F}(\rho_{11})) \\
 & \leq & \forall \gamma_0' {\leq} \sigma, \gamma_1' {\leq} \mathcal{F}(\rho_1) \,..\, \gamma_n' {\leq} \mathcal{F}(\rho_n).\ \neg \sigma \\
\equiv & \vdash & \forall \gamma_0, \alpha_1 .. \alpha_n .\ \neg(\forall \gamma_0' {\leq} \gamma_0, \alpha_1' {\leq} \mathcal{F}(\rho_{11}) \,..\, \alpha_n' {\leq} \mathcal{F}(\rho_{1n}).\ \neg \mathcal{F}(\rho_{11})) \\
 & \leq & \forall \gamma_0 {\leq} \sigma, \alpha_1 {\leq} \mathcal{F}(\rho_1) \,..\, \alpha_n {\leq} \mathcal{F}(\rho_n).\ \neg \sigma \\
\xrightarrow{*}_F & \vdash & [\sigma/\gamma_0, \mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n] \\
 & & \quad \neg(\forall \gamma_0' {\leq} \gamma_0, \alpha_1' {\leq} \mathcal{F}(\rho_{11}) \,..\, \alpha_n' {\leq} \mathcal{F}(\rho_{1n}).\ \neg \mathcal{F}(\rho_{11})) \\
 & \leq & [\sigma/\gamma_0, \mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\neg \sigma \\
\equiv & \vdash & \neg(\forall\ \gamma_0' \leq \sigma, \\
 & & \quad \alpha_1' \leq ([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11})) \,.. \\
 & & \quad \alpha_n' \leq ([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{1n})). \\
 & & \quad\quad \neg([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11}))) \\
 & \leq & \neg \sigma \\
\longrightarrow_F & \vdash & \sigma \\
 & \leq & \forall\ \gamma_0' \leq \sigma, \\
 & & \quad \alpha_1' \leq ([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11})) \,.. \\
 & & \quad \alpha_n' \leq ([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{1n})). \\
 & & \quad\quad \neg([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11}))
\end{array}
$$

$$
\begin{array}{rll}
\equiv & \vdash & \sigma \\
& \leq & \forall\, \gamma_0 \leq \sigma, \\
& & \quad \gamma_1 \leq ([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11}))\ .. \\
& & \quad \gamma_n \leq ([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{1n})). \\
& & \qquad \neg([\mathcal{F}(\rho_1)/\alpha_1 \,..\, \mathcal{F}(\rho_n)/\alpha_n]\,\mathcal{F}(\rho_{11})) \\
\equiv & \mathcal{F}(R'). &
\end{array}
$$

□

8.5. Lemma: If $R \equiv \langle \text{HALT}, \rho_2..\rho_n \rangle$, then $\mathcal{F}(R)$ is derivable in $F_{\leq}^{F}$.

Proof: Let

$$\sigma \equiv \forall \gamma_0, \gamma_1..\gamma_n.\ \neg(\forall \gamma_0' {\leq} \gamma_0, \gamma_1' {\leq} \gamma_1 \,..\, \gamma_n' {\leq} \gamma_n.\ \neg\gamma_0).$$

Then

$$
\begin{array}{rll}
& \mathcal{F}(R) & \\
\equiv & \vdash & \sigma \\
& \leq & \forall \gamma_0 {\leq} \sigma, \gamma_1 {\leq} \mathcal{F}(\text{HALT}) \,..\, \gamma_n {\leq} \mathcal{F}(\rho_n).\ \neg\mathcal{F}(\text{HALT}) \\
\equiv & \vdash & \forall \gamma_0, \gamma_1..\gamma_n.\ \neg(\forall \gamma_0' {\leq} \gamma_0, \gamma_1' {\leq} \gamma_1 \,..\, \gamma_n' {\leq} \gamma_n.\ \neg\gamma_0) \\
& \leq & \forall \gamma_0 {\leq} \sigma, \gamma_1 {\leq} \mathcal{F}(\text{HALT}) \,..\, \gamma_n {\leq} \mathcal{F}(\rho_n).\ \neg\mathcal{F}(\text{HALT}) \\
\xrightarrow{*}_F & \vdash & [\sigma/\gamma_0, \mathcal{F}(\text{HALT})/\gamma_1 \,..\, \mathcal{F}(\rho_n)/\gamma_n]\,\neg(\forall \gamma_0' {\leq} \gamma_0, \gamma_1' {\leq} \gamma_1 \,..\, \gamma_n' {\leq} \gamma_n.\ \neg\gamma_0) \\
& \leq & [\sigma/\gamma_0, \mathcal{F}(\text{HALT})/\gamma_1 \,..\, \mathcal{F}(\rho_n)/\gamma_n]\,\neg\mathcal{F}(\text{HALT}) \\
\equiv & \vdash & \neg(\forall \gamma_0' {\leq} \sigma, \gamma_1' {\leq} \mathcal{F}(\text{HALT}) \,..\, \gamma_n' {\leq} \mathcal{F}(\rho_n).\ \neg\sigma) \\
& \leq & \neg\mathcal{F}(\text{HALT}) \\
\longrightarrow_F & \vdash & \mathcal{F}(\text{HALT}) \\
& \leq & \forall \gamma_0' {\leq} \sigma, \gamma_1' {\leq} \mathcal{F}(\text{HALT}) \,..\, \gamma_n' {\leq} \mathcal{F}(\rho_n).\ \neg\sigma \\
\equiv & \vdash & \forall \gamma_0, \alpha_1 \,..\, \alpha_n.\ \neg\mathit{Top} \\
& \leq & \forall \gamma_0' {\leq} \sigma, \gamma_1' {\leq} \mathcal{F}(\text{HALT}) \,..\, \gamma_n' {\leq} \mathcal{F}(\rho_n).\ \neg\sigma \\
\xrightarrow{*}_F & \vdash & \neg\mathit{Top} \\
& \leq & \neg\sigma \\
\longrightarrow_F & \vdash & \sigma \\
& \leq & \mathit{Top},
\end{array}
$$

which is an instance of FTOP. □

8.6. Corollary: The rowing machine R halts iff $\mathcal{F}(R)$ is derivable in $F_{\leq}^{F}$.

9 Two-counter Machines

This section reviews the definition of two-counter Turing machines; see, e.g., Hopcroft and Ullman [25] for more details.

9.1. Definition: A *two-counter machine* is a tuple

$$\langle PC, A, B, I_1..I_w \rangle.$$

where A and B are nonnegative numbers and PC and I_1 through I_w are instructions of the form

INCA$\Rightarrow m$
INCB$\Rightarrow m$
TSTA$\Rightarrow m/n$
TSTB$\Rightarrow m/n$
HALT

with m and n in the range 1 to w.

9.2. Definition: The *elaboration function* $\mathcal{E}$ for two-counter machines is the partial function mapping $T = \langle PC, A, B, I_1..I_w \rangle$ to

$$\mathcal{E}(T) = \begin{cases} \langle I_m, A{+}1, B, I_1..I_w \rangle & \text{if } PC \equiv \text{INCA}\Rightarrow m \\ \langle I_m, A, B{+}1, I_1..I_w \rangle & \text{if } PC \equiv \text{INCB}\Rightarrow m \\ \langle I_m, A, B, I_1..I_w \rangle & \text{if } PC \equiv \text{TSTA}\Rightarrow m/n \text{ and } A = 0 \\ \langle I_n, A{-}1, B, I_1..I_w \rangle & \text{if } PC \equiv \text{TSTA}\Rightarrow m/n \text{ and } A > 0 \\ \langle I_m, A, B, I_1..I_w \rangle & \text{if } PC \equiv \text{TSTB}\Rightarrow m/n \text{ and } B = 0 \\ \langle I_n, A, B{-}1, I_1..I_w \rangle & \text{if } PC \equiv \text{TSTB}\Rightarrow m/n \text{ and } B > 0 \\ \text{undefined} & \text{if } PC \equiv \text{HALT}. \end{cases}$$

9.3. Convention: For the following examples, it is convenient to assign alphabetic labels to the instructions of a program. By convention, the instruction with label START is used as the initial PC, and the initial value in both registers is 0.

9.4. Example: This program loads register A with the value 5, then tests the parity of register A, halting if it is even and looping forever if it is odd:

```
START   INCA⇒I1

I1      INCA⇒I2
I2      INCA⇒I3
I3      INCA⇒I4
I4      INCA⇒E

E       TSTA⇒OK/O
O       TSTA⇒LOOP/E

LOOP    INCA⇒LOOP
OK      HALT.
```

9.5. Example: This program loads 5 into register A and 3 into register B, then compares A and B for equality by repeatedly decrementing them until one or both become zero; if both do so on the same iteration, the program halts; otherwise it goes into an infinite loop.

```
START   INCA⇒I1

I1      INCA⇒I2
I2      INCA⇒I3
I3      INCA⇒I4
I4      INCA⇒J0

J0      INCB⇒J1
J1      INCB⇒J2
J2      INCB⇒LL

LL      TSTA⇒AZ/AS
AZ      TSTB⇒AZBZ/AZBS
AS      TSTB⇒ASBZ/LL
AZBZ    HALT
AZBS    INCA⇒AZBS
ASBZ    INCA⇒ASBZ.
```

9.6. Definition: A two-counter machine T *halts* if $T \xrightarrow{*}_T T'$ for some machine $T' \equiv \langle \text{HALT}, A', B', I_1..I_w \rangle$.

9.7. Fact: The halting problem for two-counter machines is undecidable.

Proof sketch: Hopcroft and Ullman [25, pp. 171–173] show that a similar formulation of two-counter machines is Turing-equivalent. (The two-counter machines in [25] have test instructions that do not change the contents of the register being tested and separate decrement instructions. It is easy to check that this formulation and the one used here are inter-encodable.) □

10 Encoding Two-counter Machines as Rowing Machines

We can now finish the proof of the undecidability of $F_{\leq}$ subtyping by showing that any two-counter machine T can be encoded as a rowing machine $\mathcal{R}(T)$ such that T halts iff $\mathcal{R}(T)$ does.

The main trick of the encoding lies in the representation of natural numbers as rows. Each number n is encoded as a *program* that, when executed, branches indirectly through one of two registers whose contents have been set beforehand to appropriate destinations for the zero and nonzero cases of a test — in other words, n encapsulates the behavior of the test instruction on a register containing n. The increment operation simply builds a new program of this sort from an existing one. The new program saves a pointer to the present contents of the register in a local variable so that it can restore the old value (i.e., one less than its own value) before executing the branch.

The encoding $\mathcal{R}(T)$ of a two-counter machine $T \equiv \langle PC, A, B, I_1..I_w \rangle$ comprises the following registers:

#1	$\mathcal{R}^w(PC)$
#2	$\mathcal{R}^w_A(A)$
#3	$\mathcal{R}^w_B(B)$
#4	address register for zero branches
#5	address register for nonzero branches
#6	$\mathcal{R}^w(I_1)$
...	
#6+w−1	$\mathcal{R}^w(I_w)$.

We use four translation functions for the various components:

1. $\mathcal{R}(T)$ is the encoding of a the two-counter machine T as a rowing machine of width $w+5$;
2. $\mathcal{R}^w(I)$ is the encoding of a two-counter instruction I as a row of width $w+5$;
3. $\mathcal{R}^w_A(n)$ is the encoding of the natural number n, when it appears as the contents of register A, as a row of width $w+5$;
4. $\mathcal{R}^w_B(n)$ is the encoding of the natural number n, when it appears as the contents of register B, as a row of width $w+5$.

10.1. Definition: The *row-encoding* (for w instructions) of a natural number n in register A, written $\mathcal{R}^w_A(n)$, is defined as follows:

$$\begin{aligned}\mathcal{R}^w_A(0) &= \langle \#4, \text{—}, \text{—}, \text{HALT}, \text{HALT}, \underbrace{\text{—}\ ..\ \text{—}}_{w \text{ times}}\rangle \\ \mathcal{R}^w_A(n{+}1) &= \langle \#5, \mathcal{R}^w_A(n), \text{—}, \text{HALT}, \text{HALT}, \underbrace{\text{—}\ ..\ \text{—}}_{w \text{ times}}\rangle.\end{aligned}$$

The row-encoding (for w instructions) of a natural number n in register B, written $\mathcal{R}^w_B(n)$, is defined as follows:

$$\begin{aligned}\mathcal{R}^w_B(0) &= \langle \#4, \text{—}, \text{—}, \text{HALT}, \text{HALT}, \underbrace{\text{—}\ ..\ \text{—}}_{w \text{ times}}\rangle \\ \mathcal{R}^w_B(n{+}1) &= \langle \#5, \text{—}, \mathcal{R}^w_B(n), \text{HALT}, \text{HALT}, \underbrace{\text{—}\ ..\ \text{—}}_{w \text{ times}}\rangle.\end{aligned}$$

10.2. Definition: The *row-encoding* (for w instructions) of an instruction I, written $\mathcal{R}^w(I)$, is defined as follows:

$$\begin{aligned}\mathcal{R}^w(\text{INCA}{\Rightarrow}m) &= \langle \#m{+}5, \langle \#5, \#\#2, \text{—}, \text{HALT}, \text{HALT}, \text{—}\ ..\ \text{—}\rangle, \text{—}, \text{HALT}, \text{HALT}, \text{—}\ ..\ \text{—}\rangle \\ \mathcal{R}^w(\text{INCB}{\Rightarrow}m) &= \langle \#m{+}5, \text{—}, \langle \#5, \text{—}, \#\#3, \text{HALT}, \text{HALT}, \text{—}\ ..\ \text{—}\rangle, \text{HALT}, \text{HALT}, \text{—}\ ..\ \text{—}\rangle \\ \mathcal{R}^w(\text{TSTA}{\Rightarrow}m/n) &= \langle \#2, \text{—}, \text{—}, \#m{+}5, \#n{+}5, \text{—}\ ..\ \text{—}\rangle \\ \mathcal{R}^w(\text{TSTB}{\Rightarrow}m/n) &= \langle \#3, \text{—}, \text{—}, \#m{+}5, \#n{+}5, \text{—}\ ..\ \text{—}\rangle \\ \mathcal{R}^w(\text{HALT}) &= \langle \text{HALT}, \text{—}, \text{—}, \text{HALT}, \text{HALT}, \text{—}\ ..\ \text{—}\rangle.\end{aligned}$$

10.3. Definition: Let $T \equiv \langle PC, A, B, I_1..I_w\rangle$ be a two-counter machine. The *row-encoding* of T, written $\mathcal{R}(T)$, is the rowing machine of width $w{+}5$ defined as follows:

$$\mathcal{R}(T) = \langle \mathcal{R}^w(PC), \mathcal{R}^w_A(A), \mathcal{R}^w_B(B), \text{HALT}, \text{HALT}, \mathcal{R}^w(I_1)\ ..\ \mathcal{R}^w(I_w)\rangle.$$

10.4. Lemma: If $T \longrightarrow_T T'$, then $\mathcal{R}(T) \xrightarrow{*}_R \mathcal{R}(T')$.

Proof: Let $T = \langle PC, A, B, I_1..I_w \rangle$. Proceed by cases on the form of PC.

Case: $PC = \text{INCA} \Rightarrow m$

Then $T' = \langle I_m, A+1, B, I_1..I_w \rangle$. Calculate as follows:

$$\begin{array}{ll} & \mathcal{R}(T) \\ \equiv & \langle\langle \#m+5, \langle \#5, \#\#2, \text{—}, \text{HALT}, \text{HALT}, \text{—} .. \text{—}\rangle, \text{—}, \text{HALT}, \text{HALT}, \text{—} .. \text{—}\rangle, \\ & \mathcal{R}^w_A(A), \mathcal{R}^w_B(B), \\ & \text{HALT}, \text{HALT}, \\ & \mathcal{R}^w(I_1) .. \mathcal{R}^w(I_w)\rangle \\ \longrightarrow_R & \langle \mathcal{R}^w(I_m), \\ & \langle \#5, \mathcal{R}^w_A(A), \text{—}, \text{HALT}, \text{HALT}, \text{—} .. \text{—}\rangle, \mathcal{R}^w_B(B), \\ & \text{HALT}, \text{HALT}, \\ & \mathcal{R}^w(I_1) .. \mathcal{R}^w(I_w)\rangle \\ \equiv & \mathcal{R}(T'). \end{array}$$

Case: $PC = \text{INCB} \Rightarrow m$

Similar.

Case: $PC = \text{TSTA} \Rightarrow m/n$

Calculate as follows:

$$\begin{array}{ll} & \mathcal{R}(T) \\ \equiv & \langle\langle \#2, \text{—}, \text{—}, \#m+5, \#n+5, \text{—} .. \text{—}\rangle, \\ & \mathcal{R}^w_A(A), \mathcal{R}^w_B(B), \\ & \text{HALT}, \text{HALT}, \\ & \mathcal{R}^w(I_1) .. \mathcal{R}^w(I_w)\rangle \\ \longrightarrow_R & \langle \mathcal{R}^w_A(A), \\ & \mathcal{R}^w_A(A), \mathcal{R}^w_B(B), \\ & \mathcal{R}^w(I_m), \mathcal{R}^w(I_n), \\ & \mathcal{R}^w(I_1) .. \mathcal{R}^w(I_w)\rangle \end{array}$$

There are two subcases to consider:

Subcase: $A = 0$

Then

$$\begin{array}{lcl} \mathcal{R}^w_A(A) & = & \langle \#4, \text{—}, \text{—}, \text{HALT}, \text{HALT}, \text{—} .. \text{—}\rangle \\ T' & = & \langle I_m, A, B, I_1..I_w \rangle. \end{array}$$

Continue calculating as follows:

$$\begin{array}{l} \langle\langle \#4, \text{—}, \text{—}, \text{HALT}, \text{HALT}, \text{—} .. \text{—}\rangle, \\ \mathcal{R}^w_A(A), \mathcal{R}^w_B(B), \\ \mathcal{R}^w(I_m), \mathcal{R}^w(I_n), \\ \mathcal{R}^w(I_1) .. \mathcal{R}^w(I_w)\rangle \end{array}$$

$$
\begin{array}{ll}
\longrightarrow_R & \langle \mathcal{R}^w(I_m), \\
& \mathcal{R}^w_A(A), \mathcal{R}^w_B(B), \\
& \text{HALT}, \text{HALT}, \\
& \mathcal{R}^w(I_1) \,..\, \mathcal{R}^w(I_w) \rangle \\
\\
\equiv & \mathcal{R}(T').
\end{array}
$$

Subcase: $A > 0$

Then

$$
\begin{array}{lcl}
\mathcal{R}^w_A(A) & = & \langle \#5, \mathcal{R}^w_A(A-1), -, \text{HALT}, \text{HALT}, - \,..\, - \rangle \\
T' & = & \langle I_n, A-1, B, I_1..I_w \rangle.
\end{array}
$$

Continue calculating as follows:

$$
\begin{array}{ll}
& \langle \langle \#5, \mathcal{R}^w_A(A-1), -, \text{HALT}, \text{HALT}, - \,..\, - \rangle, \\
& \mathcal{R}^w_A(A), \mathcal{R}^w_B(B), \\
& \mathcal{R}^w(I_m), \mathcal{R}^w(I_n), \\
& \mathcal{R}^w(I_1) \,..\, \mathcal{R}^w(I_w) \rangle \\
\\
\longrightarrow_R & \langle \mathcal{R}^w(I_n), \\
& \mathcal{R}^w_A(A-1), \mathcal{R}^w_B(B), \\
& \text{HALT}, \text{HALT}, \\
& \mathcal{R}^w(I_1) \,..\, \mathcal{R}^w(I_w) \rangle \\
\\
\equiv & \mathcal{R}(T').
\end{array}
$$

Case: $PC = \text{TSTB} \Rightarrow m/n$

Similar.

Case: $PC = \text{HALT}$

Can't happen. □

10.5. Lemma: If $T = \langle \text{HALT}, A, B, I_1..I_w \rangle$, then $\mathcal{R}(T)$ halts.

Proof: Immediate. □

10.6. Corollary: T halts iff $\mathcal{R}(T)$ does.

10.7. Theorem: The $F_{\leq}$ subtyping relation is undecidable.

Proof: Assume, for a contradiction, that we had a total-recursive procedure for testing the derivability of subtyping statements in $F_{\leq}$. Then to decide whether a two-counter machine T halts, we could use this procedure to test whether $\mathcal{F}(\mathcal{R}(T))$ is derivable, since

	T halts	
iff	$\mathcal{R}(T)$ halts	by Corollary 10.6
iff	$\mathcal{F}(\mathcal{R}(T))$ is derivable in $F^F_{\leq}$	by Corollary 8.6
iff	$\mathcal{F}(\mathcal{R}(T))$ is derivable in $F^D_{\leq}$	by Lemma 6.6
iff	$\mathcal{F}(\mathcal{R}(T))$ is derivable in $F^N_{\leq}$	by Lemma 5.4
iff	$\mathcal{F}(\mathcal{R}(T))$ is derivable in $F_{\leq}$	by Lemma 3.2.

□

11 Undecidability of Typechecking

From the undecidability of $F_\leq$ subtyping, the undecidability of typechecking follows immediately: we need only show how to write down a term that is well typed iff a given subtyping statement

$$\vdash \sigma \leq \tau$$

is derivable. One such term is:

$$\lambda f{:}\tau{\rightarrow}Top.\ \lambda a{:}\sigma.\ f\ a.$$

12 Conclusions

The undecidability of $F_\leq$ will perhaps surprise many of those who have studied, extended, and applied it since its introduction in 1985. But it may turn out that language designs and implementations based on $F_\leq$ will not be greatly affected by this discovery. Here are some reasons for optimism:

1. The algorithm has been used for several years now without any sign of misbehavior in any situation arising in practice. Indeed, constructing even the simplest nonterminating example requires a contortion that is difficult to imagine anyone performing by accident.
2. A number of useful fragments of $F_\leq$ are easily shown to be decidable. For example:
   - The prenex fragment, where all quantifiers appear at the outside and quantifiers are instantiated only at monotypes.
   - A predicative fragment where types are stratified into universes and the bound of a quantified type lives in a lower universe than the quantified type itself.
   - Cardelli and Wegner's original formulation where the bounds of two quantified types must be identical in order for one to be a subtype of the other.
3. The best (known) subtyping algorithms for these fragments are essentially identical to the algorithm $F_\leq^N$.

Acknowledgements

I am grateful for productive discussions of this material with John Reynolds, Robert Harper, Luca Cardelli, Giorgio Ghelli, Daniel Sleator, and Tim Freeman.

A Ghelli's Decidability Argument

Ghelli's Ph.D. thesis [21, pp. 80–83] argues that the algorithm $F_{\leq}^{N}$ always terminates and is therefore a decision procedure for $F_{\leq}$. This section briefly sketches Ghelli's argument and shows where it goes wrong. The problem is quite subtle: the incorrect proof was read by a number of people (including the present author) before the flaw was detected, independently, by Curien and Reynolds.

The idea, as usual, is to define a well-founded complexity metric and show that if J' is a subproblem of J, then $complexity(J')$ is strictly less than $complexity(J)$.

A.1. Definition: The function $index_\Gamma(\alpha)$ gives the index in Γ (counting from left to right) of the binding of α.

A.2. Definition: The *left depth* of a type variable α in a type τ and an environment Γ is the number of bound type variables in both τ and Γ at α's point of definition. To formalize this concept, it is convenient to assume that all binding occurrences of type variables in τ and Γ have been renamed so as to be distinct from each other (or better yet, that deBruijn indices are used instead of variable names, as in Appendix A). Now define:

$$ld(\alpha,\tau,\Gamma) = \begin{cases} len(\Gamma) + ld(\alpha,\tau) & \text{if } \alpha \text{ is bound in } \tau \\ index_\Gamma(\alpha) & \text{otherwise} \end{cases}$$

$ld(\alpha,\tau) =$ the number of instances of $\forall$ in whose scope the binding occurrence of α (in τ) falls.

Define the left depth of a type σ in a context Γ to be the maximum left depth of any type variable in σ:

$$ld(\sigma,\Gamma) = \max(\{-1\} \cup \{ld(\alpha,\sigma,\Gamma) \mid \alpha \in TV(\sigma)\})$$

Define the complexity of a subtyping statement $(\Gamma \vdash \sigma \leq \tau)$ to be the following pair:

$$complexity(\Gamma \vdash \sigma \leq \tau) = (ld(\sigma,\Gamma) + ld(\tau,\Gamma),\ size(\sigma) + size(\tau))$$

Order the range of $complexity(\Gamma,\sigma,\tau)$ lexicographically. Note that this ordering is well founded (contains no infinite descending chains).

This ordering operates as desired for all the rules of $F_{\leq}^{N}$ with the exception of NVAR. An example of its misbehavior in this case is the following. Let

$$\Gamma \equiv \alpha{\leq}(\forall\beta{\leq}Top.\ Top), \alpha'{\leq}Top.$$

Then

$$\begin{aligned} ld(\alpha,\Gamma) &= ld(\alpha,\alpha,\Gamma) \\ &= 1, \end{aligned}$$

whereas

$$\begin{aligned} ld((\forall\gamma{\leq}Top.\ Top),\Gamma) &= ld(\gamma,(\forall\gamma{\leq}Top.\ Top),\Gamma) \\ &= len(\Gamma) + ld(\gamma,(\forall\gamma{\leq}Top.\ Top)) \\ &= 2 + ld(\gamma,(\forall\gamma{\leq}Top.\ Top)) \\ &= 2. \end{aligned}$$

So the instance

$$\frac{\alpha{\leq}(\forall\beta{\leq}Top.\ Top) \vdash \forall\gamma{\leq}Top.\ Top \leq Top}{\alpha{\leq}(\forall\beta{\leq}Top.\ Top) \vdash \alpha \leq Top}$$

of NVAR has a premise of greater complexity than its conclusion.

B DeBruijn-Indexed Presentation of the Subtyping Algorithm

While developing these results, it was helpful to work with a formulation of the concrete syntax of $F_{\leq}$ where, instead of names, numeric indices are used to indicate the binder to which each free variable occurrence corresponds. For example, the pure λ-term

$$\lambda x.\ \lambda y.\ x\ y$$

is written in this notation as

$$\lambda.\ \lambda.\ 1\ 0.$$

Similarly, the $F_{\leq}$ statement

$$\alpha{\leq}\mathit{Top} \vdash \forall\beta{\leq}\alpha.\ \alpha \leq \forall\beta{\leq}\alpha.\ \beta$$

is written

$$\mathit{Top} \vdash \forall 0.\ 1 \leq \forall 0.\ 0.$$

This idea was introduced by deBruijn [20] and has been widely used by language theorists and practitioners, especially as the basis for internal data structures in compilers. For the convenience of readers who may wish to experiment with $F_{\leq}$ themselves, this section briefly reviews the notation and shows how it applies to the nonterminating example from Section 4.

B.1. Definition: The sets τ^n of types well formed in contexts of length n are defined as follows:

$$\begin{array}{lcll} \tau^n & ::= & \mathit{Top} & \\ & | & \tau^n \rightarrow \tau^n & \\ & | & \forall\tau^n.\ \tau^{n+1} & \\ & | & n-1 & (\text{when } n > 0) \\ & | & \tau^{n-1} & (\text{when } n > 0). \end{array}$$

(Arithmetic calculations like "$n-1$" take place in the metalanguage: "$n-1$" does not appear in the trees defined by this grammar.)

B.2. Definition: The sets Γ^n of well-formed contexts of length n are defined as follows

$$\begin{array}{lcl} \Gamma^0 & ::= & \mathit{empty} \\ \Gamma^{n+1} & ::= & \Gamma^n, \tau^n. \end{array}$$

B.3. Convention: We normally suppress superscripts on metavariables. As before, we assume that all statements under consideration are well formed.

B.4. Definition: The relocation operators R_n^c and R_n are defined as follows:

$$\begin{array}{lcl} R_n^c\ \mathit{Top} & = & \mathit{Top} \\ R_n^c\ m & = & \text{if } c \leq m \text{ then } m{+}n \text{ else } m \\ R_n^c\ (\tau_1 \rightarrow \tau_2) & = & R_n^c\ \tau_1 \rightarrow R_n^c\ \tau_2 \\ R_n^c\ (\forall\tau_1.\ \tau_2) & = & \forall R_n^c\ \tau_1.\ R_n^{c+1}\ \tau_2 \\ & & \\ R_n & = & R_n^0. \end{array}$$

B.5. Definition: $F_{\leq}^{N}$ is the least relation closed under the following rules:

$$\Gamma \vdash \sigma \leq \mathit{Top} \tag{NTop}$$

$$\Gamma \vdash n \leq n \tag{NRefl}$$

$$\frac{\Gamma \vdash R_{n+1}(\Gamma(n)) \leq \tau}{\Gamma \vdash n \leq \tau} \tag{NVar}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leq \tau_1 \to \tau_2} \tag{NArrow}$$

$$\frac{\Gamma \vdash \tau_1 \leq \sigma_1 \qquad \Gamma, \tau_1 \vdash \sigma_2 \leq \tau_2}{\Gamma \vdash \forall \sigma_1.\ \sigma_2 \leq \forall \tau_1.\ \tau_2} \tag{NAll}$$

In this notation, the nonterminating input discovered by Ghelli is written

$\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0) \vdash 0 \leq \forall 0.\ \neg 0$

and elaborates as follows:

$$\begin{array}{llll}
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0) & \vdash & 0 & \leq\ \forall 0.\ \neg 0 \\
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0) & \vdash & \forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0) & \leq\ \forall 0.\ \neg 0 \\
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0), 0 & \vdash & \neg(\forall 0.\ \neg 0) & \leq\ \neg 0 \\
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0), 0 & \vdash & 0 & \leq\ \forall 0.\ \neg 0 \\
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0), 0 & \vdash & 1 & \leq\ \forall 0.\ \neg 0 \\
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0), 0 & \vdash & \forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0) & \leq\ \forall 0.\ \neg 0 \\
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0), 0, 0 & \vdash & \neg(\forall 0.\ \neg 0) & \leq\ \neg 0 \\
\forall \mathit{Top}.\ \neg(\forall 0.\ \neg 0), 0, 0 & \vdash & 0 & \leq\ \forall 0.\ \neg 0
\end{array}$$

etc.

References

[1] Val Breazu-Tannen, Thierry Coquand, Carl Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93:172–221, 1991.

[2] Kim B. Bruce. The equivalence of two semantic definitions for inheritance in object-oriented languages. In *Proceedings of Mathematical Foundations of Programming Semantics*, Pittsburgh, PA, March 1991. To appear.

[3] Kim Bruce and Giuseppe Longo. A modest model of records, inheritance, and bounded quantification. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 38–50, 1988.

[4] Peter Canning, William Cook, Walter Hill, Walter Olthoff, and John Mitchell. F-bounded quantification for object-oriented programming. In *Fourth International Conference on Functional Programming Languages and Computer Architecture*, pages 273–280, September 1989.

[5] Luca Cardelli. A semantics of multiple inheritance. In G. Kahn, D. MacQueen, and G. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 51–67. Springer-Verlag, 1984.

[6] Luca Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988.

[7] Luca Cardelli. Structural subtyping and the notion of power type. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages*, pages 70–79, San Diego, CA, January 1988.

[8] Luca Cardelli. Typeful programming. Research Report 45, Digital Equipment Corporation, Systems Research Center, Palo Alto, California, February 1989.

[9] Luca Cardelli, 1991. Personal Communication.

[10] Luca Cardelli. Extensible records in a pure calculus of subtyping. To appear, 1991.

[11] Luca Cardelli and Giuseppe Longo. A semantic basis for Quest: (Extended abstract). In *ACM Conference on Lisp and Functional Programming*, pages 30–43, Nice, France, June 1990. Extended version available as DEC SRC Research Report 55, Feb. 1990.

[12] Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of system F with subtyping. In *Theoretical Aspects of Computer Science*, Sendai, Japan, 1991. To appear.

[13] Luca Cardelli and John Mitchell. Operations on records (summary). In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proceedings of Fifth International Conference on Mathematical Foundations of Programming Language Semantics*, volume 442 of *Lecture Notes in Computer Science*, pages 22–52. Tulane University, New Orleans, March 1989. Springer Verlag. To appear in Mathematical Structures in Computer Science; also available as DEC Systems Research Center Research Report #48, August, 1989.

[14] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4), December 1985.

[15] Felice Cardone. Relational semantics for recursive types and bounded quantification. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 164–178, Stresa, Italy, July 1989. Springer-Verlag.

[16] William R. Cook, Walter L. Hill, and Peter S. Canning. Inheritance is not subtyping. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 125–135, San Francisco, CA, January 1990.

[17] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and lambda calculus semantics. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 535–560. New York, 1980. Academic Press.

[18] Pierre-Louis Curien and Giorgio Ghelli. Coherence of subsumption. *Mathematical Structures in Computer Science*, 1991. To appear.

[19] Pierre-Louis Curien and Giorgio Ghelli. Subtyping + extensionality: confluence of $\beta\eta$-reductions in $F_{\leq}$. In *Theoretical Aspects of Computer Science*, Sendai, Japan, 1991. To appear.

[20] Nicolas G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.

[21] Giorgio Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, Università di Pisa, March 1990. Technical report TD–6/90, Dipartimento di Informatica, Università di Pisa.

[22] Giorgio Ghelli, 1991. Personal Communication.

[23] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[24] Carl Gunter, 1990. Personal Communication.

[25] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[26] Simone Martini. Bounded quantifiers have interval models. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, pages 174–183, Snowbird, Utah, July 1988. ACM.

[27] John C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76:211–249, 1988.

[28] Benjamin C. Pierce. Programming with intersection types and bounded polymorphism. Ph.D. thesis (in progress), 1991.

[29] John Reynolds. Towards a theory of type structure. In *Proc. Colloque sur la Programmation*, pages 408–425, New York, 1974. Springer-Verlag LNCS 19.

[30] John C. Reynolds. Preliminary design of the programming language Forsythe. Technical Report CMU-CS-88-159, Carnegie Mellon University, June 1988.