1 Introduction

Two of the most important developments in type theory were the invention, by W. W. Tait, of Tait’s Method for function types, which was later extended by J.Y. Girard to Girard’s Method for type quantification, both of which were incorporated into a general theory of logical relations for a wide range of type theories.

The problem considered by Tait was to prove that $\beta$-reduction for the simply typed $\lambda$-calculus is strongly normalizing, which is usually defined to mean that there are no infinite $\beta$-reduction sequences starting with a well-typed term: $M = M_0 \rightarrow_{\beta} M_1 \rightarrow \ldots$. A better definition, of more immediate utility, is the validity of transfinite induction on reduction, stated as follows: for any property $P$ of typed $\lambda$-terms, to show that $P$ holds for every such term, it is enough to show, for every well-typed typed term $M$, if all of its immediate $\beta$-reducts satisfy $P$, then so does $P$. More succinctly,

$$(\forall M: \tau (\forall N: \tau. M \rightarrow_{\beta} N \supset P(N))) \supset \forall M: \tau. P(M).$$

The importance of strong normalization lies exactly in the utility afforded by this principle for proving other properties. For example, using transfinite induction on reduction it is possible to prove that weak confluence (every one-step split can be reconciled) implies confluence (transitivity of the “has a common reduct” relation.) This statement is false for untyped terms; it is precisely the strong normalization property that ensures its validity for well-typed terms.

The question considered here is a related, but technically much simpler, problem, the termination of a deterministic head reduction strategy for a simply typed $\lambda$-calculus. The simplification compared to Tait’s original proof is that head reduction does not require consideration of open terms. The type system considered here has unit, product, and function types, augmented with a two-element type of observables, corresponding to the familiar “accept/reject” formulation in the study of abstract machines.

The method can be generalized in numerous ways, all of which are of central importance and unparalleled utility in type theory. In one direction it can be extended to account for open terms, using pre-sheaves of logical relations, which are also known as Kripke models. In another it can be extended to binary relations as a means of studying equality of typed terms. In another direction it can be extended to dependent types, which consider type-indexed families of types; this was chiefly developed by Per Martin-Löf. In yet another direction it can be extended to binary relations, providing an analysis of equality of typed terms.
2 Simple Types

The syntax of the language considered here is given by the following grammar:

\[ A ::= 1 \mid 2 \mid A_1 \times A_2 \mid A_1 \to A_2 \]
\[ M ::= x \mid Y \mid N \mid \ast \mid \langle M_1, M_2 \rangle \mid M \cdot 1 \mid M \cdot 2 \mid \lambda x. A. M \mid \text{ap}(M_1, M_2) \]

The statics is entirely standard, defining the typing judgment \( \Gamma \vdash M : A \), in such a way that the structural properties are admissible. Contraction and exchange are accounted for by treating the typing context \( \Gamma \) as a finite set of variable typings \( x_1 : A_1, \ldots, x_n : A_n \). Weakening is built-in by stating all rules with an ambient typing context \( \Gamma \) that goes along for the ride. Substitution is readily verified by induction on typing. See Figure 1 for the definition of typing.

The dynamics is given by a transition system \( M \mapsto M' \) between closed \( \lambda \)-terms of some type. Any closed typed term is a valid initial state. Final states are defined along with transition in Figure 2.

**Theorem 2.1** (Preservation). If \( M : A \) and \( M \mapsto M' \), then \( M' : A \).

**Proof.** By induction on transition.
3 Termination Proof

The goal is to prove termination for terms of observable type:

**Theorem 3.1 (Termination).** If $M : 2$, then either $M \rightarrow^* Y$ or $M \rightarrow^* N$.

That is, any complete program either accepts or rejects.

Given the statement of the theorem, practically the only move available is to proceed by induction on typing. Let us consider some cases.

**VAR** Does not apply to closed terms.

**YES** Immediate, as $Y$ final.

**NO** Immediate, as $N$ final.

**UNIT** Does not apply, not of type $2$.

**PAIR** Does not apply, not of type $2$.

**LFT** By induction, $um$ ....

**RHT** By induction, $um$ ....

**FUN** Does not apply, not of type $2$.

**APP** By induction applied to the first premise, $um$ ....

All cases are trivial, or completely unclear.

Well, because the subterms of a term of type $2$ need not have type $2$, it seems clear that we have to strengthen the theorem to say something about terms of any type.

**Lemma 3.2.** If $M : A$, then there exists $N$ such that $N$ final and $M \rightarrow^* N$.

The lemma suffices for the theorem because of the definition of finality for terms of type $2$. Let us consider the proof of this lemma.

**VAR** Does not apply to closed terms.

**YES** Immediate, as $Y$ final.

**NO** Immediate, as $N$ final.

**UNIT** Immediate, as $\star$ final.

**PAIR** Immediate, as $\langle M_1, M_2 \rangle$ final.

**LFT** By induction there exists $N$ such that $N$ final and $M \rightarrow^* N$. By preservation and the definition of finality $N$ must be of the form $\langle N_1, N_2 \rangle$. By the definition of transition

$$M \cdot 1 \rightarrow^* \langle N_1, N_2 \rangle \cdot 1 \rightarrow N_1.$$ 

But now what?
**RHT** Analogous, what to do with $N_2$?

**FUN** Immediate, as $\lambda x:A_1.M_2$ final.

**APP** By induction applied to the first premise there exists $N_1$ such that $N_1$ final and $M_1 \mapsto^* N_1$. By preservation and the definition of finality $N_1$ must have the form $\lambda x:A_2.M$. By the definition of transition we have

$$\text{ap}(M_1, M_2) \mapsto^* \text{ap}(\lambda x:A_2.M, M_2) \mapsto [M_2/x]M.$$

But now what?

In the projection cases the components of the pair are general terms about which we know nothing. In the application case the value of the first argument is a $\lambda$-abstraction whose body is an open term (with free variable $x$) about which we know nothing. To proceed we need to know something about the components of the pair and the body of the function. This suggests strengthening the lemma by proving a property called *hereditary termination*, which is stronger than mere termination. It should have the following characteristics in order to push through the proof of the strengthened lemma below:

1. A hereditarily terminating expression of type $1$ should be terminating, and hence transition to $\star$.
2. A hereditarily terminating expression of type $2$ should be terminating, and hence transition to either $Y$ or $N$.
3. A hereditarily terminating expression of type $A_1 \times A_2$ should terminate with a pair $\langle N_1, N_2 \rangle$ such that both $N_1$ and $N_2$ are hereditarily terminating.
4. A hereditarily terminating expression of type $A_2 \rightarrow A$ should terminate with a function $\lambda x:A_2.M$ such that if $M_2$ is hereditarily terminating of type $A_2$, then $[M_2/x]M$ should be hereditarily terminating at type $A$.

These conditions constitute a *definition* of the property $M$ is *hereditarily terminating at type* $A$, which is defined for closed $M : A$. The first two cases are given outright; the others rely on hereditary termination at constituent types of a compound type. *Thus, hereditary termination at a type is defined by induction on the structure of the type.*

**Lemma 3.3.** If $M : A$, then $M$ is hereditarily terminating at type $A$.

The proof proceeds as before by induction on typing. The problematic cases are handled by the definition of hereditary termination. The cases for the constants are immediate by the definition of hereditary termination at base type. But what about the pair and function cases?

**PAIR** By induction $M_1$ is hereditarily terminating at $A_1$ and $M_2$ is hereditarily terminating at type $A_2$; we are to show that $\langle M_1, M_2 \rangle$ is hereditarily terminating at type $A_1 \times A_2$. A pair is already a value (final state), so all we need to finish the proof is precisely what we are given by induction!

---

1 This move is precluded in the second-order case, which is what gave rise to Girard’s Method, which we do not consider here. The issue is an unavoidable circularity in any definition of hereditary termination for polymorphic types.
To show that $\lambda x:A_1. M_2$ is hereditarily terminating at $A_1 \to A_2$, we must show that whenever $M_1$ is hereditarily terminating at $A_1$, then $[M_1/x]M_2$ is hereditarily terminating at $A_2$. But what to do?

The problem now is that in the function case there is no inductive hypothesis available to give us the necessary information about the open term $M$, which has one free variable, $x$, in it. We need to strengthen the lemma once more to account for open terms, even though the desired property applies only to closed terms!

To state the required result, define $\gamma : \Gamma$ to mean that $\gamma$ is a map assigning to each variable $x$ declared in $\Gamma$ a term of the type specified in $\Gamma$. Such a map is hereditarily terminating at $\Gamma$ iff each of these terms is hereditarily terminating at its declared type.

**Lemma 3.4.** If $\Gamma \vdash M : A$ and $\gamma$ is hereditarily terminating at $\Gamma$, then $\hat{\gamma}M$ is hereditarily terminating at $A$.

We are now in a position to complete the proof. The critical case is the last one in the preceding attempt, for which the strengthened statement provides precisely what is needed to push the proof through. The other cases require a bit more care in handling the application of $\gamma$ to the terms in question, but there are no obstacles to worry about.

And we have thereby re-invented Tait’s Method!

**References**