12 *Normalization*

In this chapter, we consider another fundamental theoretical property of the pure simply typed lambda-calculus: the fact that the evaluation of a well-typed program is guaranteed to halt in a finite number of steps—i.e., every well-typed term is *normalizable*.

Unlike the type-safety properties we have considered so far, the normalization property does not extend to full-blown programming languages, because these languages nearly always extend the simply typed lambda-calculus with constructs such as general recursion (§11.11) or recursive types (Chapter 20) that can be used to write nonterminating programs. However, the issue of normalization will reappear at the level of *types* when we discuss the metatheory of System F_ω in §30-3: in this system, the language of types effectively contains a copy of the simply typed lambda-calculus, and the termination of the typechecking algorithm will hinge on the fact that a "normalization" operation on type expressions is guaranteed to terminate.

Another reason for studying normalization proofs is that they are some of the most beautiful—and mind-blowing—mathematics to be found in the type theory literature, often (as here) involving the fundamental proof technique of *logical relations*.

Some readers may prefer to skip this chapter on a first reading; doing so will not cause any problems in later chapters. (A full table of chapter dependencies appears on page xvi.)

12.1 Normalization for Simple Types

The calculus we shall consider here is the simply typed lambda-calculus over a single base type A. Normalization for this calculus is not entirely trivial to prove, since each reduction of a term can duplicate redexes in subterms.

The language studied in this chapter is the simply typed lambda-calculus (Figure 9-1) with a single base type A (11-1).

12.1.1 EXERCISE [⋆]: Where do we fail if we attempt to prove normalization by a straightforward induction on the size of a well-typed term? □

The key issue here (as in many proofs by induction) is finding a strong enough induction hypothesis. To this end, we begin by defining, for each type T, a set R_{T} of closed terms of type T. We regard these sets as predicates and write $R_{\mathsf{T}}(\mathtt{t})$ for $\mathtt{t} \in R_{\mathsf{T}}$.[1]

12.1.2 DEFINITION:

- $R_{\mathsf{A}}(\mathtt{t})$ iff $\mathtt{t}$ halts.
- $R_{\mathsf{T}_1 \to \mathsf{T}_2}(\mathtt{t})$ iff $\mathtt{t}$ halts and, whenever $R_{\mathsf{T}_1}(\mathtt{s})$, we have $R_{\mathsf{T}_2}(\mathtt{t}\ \mathtt{s})$. □

This definition gives us the strengthened induction hypothesis that we need. Our primary goal is to show that all *programs*—i.e., all closed terms of base type—halt. But closed terms of base type can contain subterms of functional type, so we need to know something about these as well. Moreover, it is not enough to know that these subterms halt, because the application of a normalized function to a normalized argument involves a substitution, which may enable more evaluation steps. So we need a stronger condition for terms of functional type: not only should they halt themselves, but, when applied to halting arguments, they should yield halting results.

The form of Definition 12.1.2 is characteristic of the *logical relations* proof technique. (Since we are just dealing with unary relations here, we should more properly say *logical predicates*.) If we want to prove some property P of all closed terms of type A, we proceed by proving, by induction on types, that all terms of type A *possess* property P, all terms of type $\mathsf{A}\to\mathsf{A}$ *preserve* property P, all terms of type $(\mathsf{A}\to\mathsf{A})\to(\mathsf{A}\to\mathsf{A})$ *preserve the property of preserving* property P, and so on. We do this by defining a family of predicates, indexed by types. For the base type A, the predicate is just P. For functional types, it says that the function should map values satisfying the predicate at the input type to values satisfying the predicate at the output type.

We use this definition to carry out the proof of normalization in two steps. First, we observe that every element of every set R_{T} is normalizable. Then we show that every well-typed term of type T is an element of R_{T}.

The first step is immediate from the definition of R_{T}:

12.1.3 LEMMA: If $R_{\mathsf{T}}(\mathtt{t})$, then $\mathtt{t}$ halts. □

The second step is broken into two lemmas. First, we remark that membership in R_{T} is invariant under evaluation.

1. The sets R_{T} are sometimes called *saturated sets* or *reducibility candidates*.

12.1.4 LEMMA: If $t : T$ and $t \longrightarrow t'$, then $R_T(t)$ iff $R_T(t')$. □

Proof: By induction on the structure of the type T. Note, first, that it is clear that t halts iff t' does. If $T = A$, there is nothing more to show. Suppose, on the other hand, that $T = T_1 \to T_2$ for some T_1 and T_2. For the "only if" direction ($\Longrightarrow$) suppose that $R_T(t)$ and that $R_{T_1}(s)$ for some arbitrary $s : T_1$. By definition we have $R_{T_2}(t\ s)$. But $t\ s \longrightarrow t'\ s$, from which the induction hypothesis for type T_2 gives us $R_{T_2}(t'\ s)$. Since this holds for an arbitrary s, the definition of R_T gives us $R_T(t')$. The argument for the "if" direction ($\Longleftarrow$) is analogous. □

Next, we want to show that every term of type T belongs to R_T. Here, the induction will be on typing derivations (it would be surprising to see a proof about well-typed terms that did not somewhere involve induction on typing derivations!). The only technical difficulty here is in dealing with the λ-abstraction case. Since we are arguing by induction, the demonstration that a term $\lambda x{:}T_1.t_2$ belongs to $R_{T_1 \to T_2}$ should involve applying the induction hypothesis to show that t_2 belongs to R_{T_2}. But R_{T_2} is defined to be a set of *closed* terms, while t_2 may contain x free, so this does not make sense.

This problem is resolved by using a standard trick to suitably generalize the induction hypothesis: instead of proving a statement involving a closed term, we generalize it to cover all closed *instances* of an open term t.

12.1.5 LEMMA: If $x_1{:}T_1, \ldots, x_n{:}T_n \vdash t : T$ and $v_1 \ldots, v_n$ are closed values of types $T_1 \ldots T_n$ with $R_{T_i}(v_i)$ for each i, then $R_T([x_1 \mapsto v_1] \cdots [x_n \mapsto v_n]t)$. □

Proof: By induction on a derivation of $x_1{:}T_1, \ldots, x_n{:}T_n \vdash t : T$. (The most interesting case is the one for abstraction.)

Case T-VAR*:* $t = x_i \qquad T = T_i$

Immediate.

Case T-ABS*:* $t = \lambda x{:}S_1.s_2 \qquad x_1{:}T_1, \ldots, x_n{:}T_n, x{:}S_1 \vdash s_2 : S_2$
$T = S_1 \to S_2$

Obviously, $[x_1 \mapsto v_1] \cdots [x_n \mapsto v_n]t$ evaluates to a value, since it is a value already. What remains to show is that $R_{S_2}(([x_1 \mapsto v_1] \cdots [x_n \mapsto v_n]t)\ s)$ for any $s : S_1$ such that $R_{S_1}(s)$. So suppose s is such a term. By Lemma 12.1.3, we have $s \longrightarrow^* v$ for some v. By Lemma 12.1.4, $R_{S_1}(v)$. Now, by the induction hypothesis, $R_{S_2}([x_1 \mapsto v_1] \cdots [x_n \mapsto v_n][x \mapsto v]s_2)$. But

$$\begin{array}{ll} & (\lambda x{:}S_1.\ [x_1 \mapsto v_1] \cdots [x_n \mapsto v_n]s_2)\ s \\ \longrightarrow^* & [x_1 \mapsto v_1] \cdots [x_n \mapsto v_n][x \mapsto v]s_2, \end{array}$$

from which Lemma 12.1.4 gives us

$$R_{S_2}((\lambda x{:}S_1.\ [x_1 \mapsto v_1] \cdots [x_n \mapsto v_n]s_2)\ s),$$

that is, $R_{S_2}(((\texttt{[}x_1 \mapsto v_1\texttt{]} \cdots \texttt{[}x_n \mapsto v_n\texttt{]}(\lambda x{:}S_1.\ s_2))\ s)$. Since s was chosen arbitrarily, the definition of $R_{S_1 \to S_2}$ gives us

$$R_{S_1 \to S_2}(\texttt{[}x_1 \mapsto v_1\texttt{]} \cdots \texttt{[}x_n \mapsto v_n\texttt{]}(\lambda x{:}S_1.\ s_2)).$$

Case T-APP*:* $t = t_1\ t_2$
$x_1{:}T_1, \ldots, x_n{:}T_n \vdash t_1 : T_{11} \to T_{12}$
$x_1{:}T_1, \ldots, x_n{:}T_n \vdash t_2 : T_{11}$
$T = T_{12}$

The induction hypothesis gives us $R_{T_{11} \to T_{12}}(\texttt{[}x_1 \mapsto v_1\texttt{]} \cdots \texttt{[}x_n \mapsto v_n\texttt{]}t_1)$ and $R_{T_{11}}(\texttt{[}x_1 \mapsto v_1\texttt{]} \cdots \texttt{[}x_n \mapsto v_n\texttt{]}t_2)$. By the definition of $R_{T_{11} \to T_{12}}$,

$$R_{T_{12}}((\texttt{[}x_1 \mapsto v_1\texttt{]} \cdots \texttt{[}x_n \mapsto v_n\texttt{]}t_1)\ (\texttt{[}x_1 \mapsto v_1\texttt{]} \cdots \texttt{[}x_n \mapsto v_n\texttt{]}t_2)),$$

i.e., $R_{T_{12}}(\texttt{[}x_1 \mapsto v_1\texttt{]} \cdots \texttt{[}x_n \mapsto v_n\texttt{]}(t_1\ t_2))$,. □

We now obtain the normalization property as a corollary, simply by taking the term t to be closed in Lemma 12.1.5 and then recalling that all the elements of R_T are normalizing, for every T.

12.1.6 THEOREM [NORMALIZATION]: If $\vdash t : T$, then t is normalizable. □

Proof: $R_T(t)$ by Lemma 12.1.5; t is therefore normalizable by Lemma 12.1.3. □

12.1.7 EXERCISE [RECOMMENDED, ⋆⋆⋆]: Extend the proof technique from this chapter to show that the simply typed lambda-calculus remains normalizing when extended with booleans (Figure 3-1) and products (Figure 11-5). □

12.2 Notes

Normalization properties are most commonly formulated in the theoretical literature as *strong normalization* for calculi with full (non-deterministic) beta-reduction. The standard proof method was invented by Tait (1967), generalized to System F (cf. Chapter 23) by Girard (1972, 1989), and later simplified by Tait (1975). The presentation used here is an adaptation of Tait's method to the call-by-value setting, due to Martin Hofmann (private communication). The classical references on the logical relations proof technique are Howard (1973), Tait (1967), Friedman (1975), Plotkin (1973, 1980), and Statman (1982, 1985a, 1985b). It is also discussed in many texts on semantics, for example those by Mitchell (1996) and Gunter (1992).

Tait's strong normalization proof corresponds exactly to an algorithm for evaluating simply typed terms, known as *normalization by evaluation* or *type-directed partial evaluation* (Berger, 1993; Danvy, 1998); also see Berger and Schwichtenberg (1991), Filinski (1999), Filinski (2001), Reynolds (1998a).