In Lecture 26, we proved that each term in the simply typed λ-calculus would never get stuck. Today, we want to show that it will actually always terminate, no matter what reduction order is used. In other words, there are no infinite reduction sequences starting from any typable term. This property is known as *strong normalization.*

Formally, we want to prove that if $\vdash e : \tau$, then any reduction sequence starting from e eventually leads to a (unique) normal form. We will prove this by induction, but we will need a fairly sophistiated argument that takes both the typing and reduction order into account. For example, even if e_1 and e_2 terminate, we cannot conclude that $(e_1\ e_2)$ does: consider $e_1 = e_2 = \lambda x.\, xx$.

1 Church vs Curry

We will prove this theorem in the pure simply-typed λ-calculus in Curry style. This differs from Church style in that the binding occurrence of a variable in a λ-abstraction is not annotated with its type.

Let $\alpha, \beta, \ldots$ denote type variables, $x, y, \ldots$ term variables, $\sigma, \tau, \ldots$ types, and $d, e, \ldots$ terms. In the Curry-style simply typed λ-calculus, terms and types are defined by

$$e \ ::= \ x \ \mid \ e_1\ e_2 \ \mid \ \lambda x.\, e \qquad\qquad \tau \ ::= \ \alpha \ \mid \ \sigma \to \tau$$

The typing rules are

$$\Gamma,\, x : \tau \vdash x : \tau \qquad\qquad \frac{\Gamma \vdash e : \sigma \to \tau \quad \Gamma \vdash d : \sigma}{\Gamma \vdash (e\ d) : \tau} \qquad\qquad \frac{\Gamma,\, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.\, e : \sigma \to \tau}$$

where Γ is a *type environment*, a partial function from type variables to types. Note that in Church style, a closed term can have at most one type, but in Curry style, if it has any type at all, then it has infinitely many. For example, $\vdash \lambda x.\, x : ((\alpha \to \beta) \to \gamma) \to ((\alpha \to \beta) \to \gamma)$. In general, if $\vdash e : \tau$, then also $\vdash e : \tau'$, where τ' is any substitution instance of τ.

A term e is *typable* if there exists a type environment Γ and a type τ such that $\Gamma \vdash e : \tau$. One can show by induction that if $\Gamma \vdash e : \tau$, then $FV(e) \subseteq \mathsf{dom}\,\Gamma$.

2 Strong Normalization

By the Church–Rosser theorem, normal forms are unique up to α-equivalence, so any two reduction strategies starting from the same term that terminate must yield the same result up to α-equivalence. However, there may be some strategies that terminate and some that do not.

A term e is *strongly normalizing* (SN) if all β-reduction sequences starting from e converge to a normal form; equivalently, if there is no infinite β-reduction sequence starting from e. Our main theorem is

Theorem 1. *All typable terms are strongly normalizing.*

Definition 2. *Define* $\Gamma \vdash_{\!\scriptscriptstyle SN} e : \tau$ *if*

(A) $\Gamma \vdash e : \tau$, *and*

(B) *if* $\tau = \sigma_1 \to \cdots \to \sigma_n \to \alpha$ *and* $\Gamma \vdash_{\!\scriptscriptstyle SN} d_i : \sigma_i$ *for* $1 \le i \le n$, *then* $(e\ d_1\ \cdots\ d_n)$ *is strongly normalizing.*

Definition 2 may seem circular, but it is not: $\Gamma \vdash_{SN} e : \tau$ is defined in terms of $\Gamma \vdash_{SN} d_i : \sigma_i$, and the σ_i are proper subexpressions of τ, so it is well-defined by structural induction on types.

Let $\delta : \mathit{Var} \rightharpoonup \{\text{terms}\}$ be a valuation of variables and $\Gamma, \Delta : \mathit{Var} \rightharpoonup \{\text{types}\}$ type environments. Define

$$\Gamma \vdash \delta : \Delta \overset{\triangle}{\iff} \forall x \in \mathsf{dom}\,\delta \;\; \Gamma \vdash \delta(x) : \Delta(x)$$
$$\Gamma \vdash_{SN} \delta : \Delta \overset{\triangle}{\iff} \forall x \in \mathsf{dom}\,\delta \;\; \Gamma \vdash_{SN} \delta(x) : \Delta(x).$$

Let $e\{\delta\}$ denote the simultaneous safe substitution of $\delta(x)$ for x in e for all $x \in \mathsf{dom}\,\delta$.

Lemma 3. *The following rule is valid:*

$$\frac{\Gamma \vdash_{SN} e_1 : \sigma \to \tau \qquad \Gamma \vdash_{SN} e_2 : \sigma}{\Gamma \vdash_{SN} (e_1\ e_2) : \tau}$$

Proof. Surely part (A) of Definition 2 holds, as without the *SN* decoration, it is just the application rule. For part (B), suppose $\tau = \sigma_1 \to \cdots \to \sigma_n \to \alpha$ and $\Gamma \vdash_{SN} d_i : \sigma_i$ for $1 \le i \le n$. We wish to show that $(e_1\ e_2\ d_1\ \cdots\ d_n)$ is strongly normalizing. But this follows directly from the two premises. □

Lemma 4. *Let $\delta : \mathit{Var} \rightharpoonup \{terms\}$ be a valuation of variables and $\Gamma, \Delta : \mathit{Var} \rightharpoonup \{types\}$ type environments with $\mathsf{dom}\,\delta = \mathsf{dom}\,\Delta$ and $\mathsf{dom}\,\Gamma \cap \mathsf{dom}\,\Delta = \varnothing$. The following rule is valid:*

$$\frac{\Gamma, \Delta \vdash e : \tau \qquad \Gamma \vdash_{SN} \delta : \Delta}{\Gamma \vdash_{SN} e\{\delta\} : \tau}$$

Proof. The proof is by induction on the structure of e.

If e is a variable $x \in \mathsf{dom}\,\delta$, the left-hand premise of the rule is $\Gamma, \Delta \vdash x : \tau$, thus $\tau = \Delta(x)$. The conclusion of the rule is then $\Gamma \vdash_{SN} \delta(x) : \Delta(x)$, which is immediate from the right-hand premise.

If e is a variable $x \notin \mathsf{dom}\,\delta$, the left-hand premise is $\Gamma, \Delta \vdash x : \tau$, thus $\tau = \Gamma(x)$. The conclusion of the rule is $\Gamma \vdash_{SN} x : \tau$. We have $\Gamma \vdash x : \tau$ from the premise, which gives part (A) of Definition 2. For part (B), suppose $\tau = \sigma_1 \to \cdots \to \sigma_n \to \alpha$ and $\Gamma \vdash_{SN} d_i : \sigma_i$ for $1 \le i \le n$. We wish to show that $(x\ d_1\ \cdots\ d_n)$ is strongly normalizing. But this must be true, because in any reduction sequence, all redexes remain inside the d_i, which are strongly normalizing by assumption.

For the case of applications $(e_1\ e_2)$, the left-hand premise is $\Gamma, \Delta \vdash (e_1\ e_2) : \tau$. This must have been derived from an application of the typing rule for applications

$$\frac{\Gamma, \Delta \vdash e_1 : \sigma \to \tau \qquad \Gamma, \Delta \vdash e_2 : \sigma}{\Gamma, \Delta \vdash (e_1\ e_2) : \tau}$$

for some type σ. By the induction hypothesis and Lemma 3, we have

$$\frac{\dfrac{\Gamma, \Delta \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash_{SN} \delta : \Delta}{\Gamma \vdash_{SN} e_1\{\delta\} : \sigma \to \tau} \qquad \dfrac{\Gamma, \Delta \vdash e_2 : \sigma \qquad \Gamma \vdash_{SN} \delta : \Delta}{\Gamma \vdash_{SN} e_2\{\delta\} : \sigma}}{\Gamma \vdash_{SN} (e_1\{\delta\}\ e_2\{\delta\}) : \tau}$$

and the conclusion is equivalent to the desired result $\Gamma \vdash_{SN} (e_1\ e_2)\{\delta\} : \tau$.

Finally, for the case of λ-abstractions $\lambda x.\, e$, assume without loss of generality that the term has been α-converted with a fresh variable x. Suppose $\tau = \sigma_1 \to \rho$ and $\rho = \sigma_2 \to \cdots \to \sigma_n \to \alpha$. It must be the case that $n \ge 1$, because the term is a λ-abstraction. The premise must have been derived from an application of the typing rule for abstractions:

$$\frac{\Gamma, \Delta, x : \sigma_1 \vdash e : \rho}{\Gamma, \Delta \vdash \lambda x.\, e : \sigma_1 \to \rho}$$

By the induction hypothesis, we have the two rules

$$\frac{\Gamma,\,\Delta,\,x:\sigma_1 \vdash e:\rho \qquad \Gamma \vdash_{SN} \delta : \Delta}{\Gamma,\,x:\sigma_1 \vdash_{SN} e\{\delta\} : \rho} \qquad \frac{\Gamma,\,\Delta,\,x:\sigma_1 \vdash e:\rho \qquad \Gamma \vdash_{SN} \delta : \Delta \qquad \Gamma \vdash_{SN} d_1 : \sigma_1}{\Gamma \vdash_{SN} e\{\delta\}\,\{d_1/x\} : \rho} \tag{1}$$

We would like to conclude that $\Gamma \vdash_{SN} \lambda x.\,e\{\delta\} : \sigma_1 \to \rho$.[1] The typing $\Gamma \vdash \lambda x.\,e\{\delta\} : \sigma_1 \to \rho$ follows from the conclusion of the left-hand rule of (1) by the abstraction rule. This gives part (A) of Definition 2.

For part (B), let $\Gamma \vdash_{SN} d_i : \sigma_i$ for $1 \le i \le n$. We wish to show that $((\lambda x.\,e\{\delta\})\ d_1\ \cdots\ d_n)$ is strongly normalizing. We know that $\lambda x.\,e\{\delta\}$ is strongly normalizing by the conclusion of the left-hand rule of (1), and the d_i are strongly normalizing by assumption. Thus a head reduction

$$(\lambda x.\,e\{\delta\}')\ d_1'\ \cdots\ d_n' \ \to\ e\{\delta\}'\,\{d_1'/x\}\ d_2'\ \cdots\ d_n'$$

must eventually be performed after any sufficiently long sequence of reductions $\lambda x.\,e\{\delta\} \overset{*}{\to} \lambda x.\,e\{\delta\}'$ and $d_i \overset{*}{\to} d_i'$, $1 \le i \le n$. But we might have done the head reduction $(\lambda x.\,e\{\delta\})\ d_1 \to e\{\delta\}\,\{d_1/x\}$ in the very first step, and $e\{\delta\}'\,\{d_1'/x\}$ is still derivable from this by reducing the same redexes as in $e\{\delta\} \overset{*}{\to} e\{\delta\}'$ and $d_1 \overset{*}{\to} d_1'$. Thus

$$(\lambda x.\,e\{\delta\})\ d_1\ \cdots\ d_n \ \to\ e\{\delta\}\,\{d_1/x\}\ d_2\ \cdots\ d_n \ \overset{*}{\to}\ e\{\delta\}'\,\{d_1'/x\}\ d_2'\ \cdots\ d_n'.$$

But $(e\{\delta\}\,\{d_1/x\}\ d_2\ \cdots\ d_n)$ is strongly normalizing by the conclusion of the right-hand rule of (1), therefore $(e\{\delta\}'\,\{d_1'/x\}\ d_2'\ \cdots\ d_n')$ must be as well. □

Proof of Theorem 1. Take Δ and δ empty in Lemma 4. □

3 Discussion

The technique uaed here is a variant of a technique called *logical relations*. This technique generalizes to more expressive languages. We will shortly see extensions of the λ-calculus that can be used to write more interesting computations, yet can be proved strongly normalizing with the same technique.

There are many situations in which it is useful to have a language in which all programs terminate. For example, operating systems and web browsers are often extended with plug-in software that is not fully trusted. Knowing that the plug-in code cannot cause an infinite loop is useful (though we probably want an even tighter bound on run time). Also, we will later see type systems with type expressions isomorphic to the λ-calculus (parameterized types). Because evaluation in the type language terminates, the type checker also terminates, which is useful to know.

[1] As $(\lambda x.\,e)\{\delta\} = \lambda x.\,(e\{\delta\})$, we simply write $\lambda x.\,e\{\delta\}$.