Rewriting and Termination in Lambda Calculus

OPLSS June 2022

1 Outline

1. Rewriting Basics
2. Lambda Calculus
3. Confluence: Church-Rosser property, local confluence
4. Normalization: strong normalization, normalization
5. Strategies: Left-outermost strategy, perpetual strategies
6. Simple types in lambda calculus: strong normalization
7. Intersection types in lambda calculus: complete characterizations of normalizations

2 Cleaning Up Yesterday

2.1 Free Variables

An important notion in lambda calculus is the notion of a free variable. Informally, we can say that a free variable is a variable not bound by an abstraction. Formally, we can define it inductively as follows:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \\ FV((\lambda x.M)) &= FV(M) \setminus \{x\} \end{aligned}$$

We say that a term M is closed if $FV(M) = \emptyset$. Finally, we define $\bigwedge^{\circ}$ as the set of all closed terms.

2.2 Substitution

Substitution is defined inductively over the language as follows

$$
\begin{aligned}
x[x := M] &\triangleq M \\
y[x := M] &\triangleq y \\
N_1 N_2[x := M] &\triangleq N_1[x := M] N_2[x := M] \\
(\lambda x.N)[x := M] &\triangleq (\lambda x.N) \\
(\lambda y.N)[x := M] &\triangleq (\lambda y.N[x := M]) \text{ if } x \notin FV(N) \\
(\lambda y.N)[x := M] &\triangleq ((\lambda z.N[y := z]))[x := M] \text{ if } x \in FV(N), z \notin FV(M, N)
\end{aligned}
$$

Note that while the language uses substitution to actually perform computation, substitution is actually a meta notion rather than something explicitly specified by the syntax. There are variants of the lambda calculus with explicit substitution that avoid the hassle of dealing with variables in the metatheory, but those are outside the scope of this lecture.

2.3 Rewrite Rules

Lambda calculus has three rules. In the last lecture we have seen the first rule which is $\alpha - conversion$. The other two two are defined as follows:

$$(\lambda x.M)N \to_\beta M[x := N] \tag{1}$$
$$(\lambda x.Mx) \to_\eta M \tag{2}$$

Where rule (1) is usually called β-reduction and rule (2) η-reduction.

3 Higher Order Rewriting Systems - Lambda Calculus

α-equivalence To capture the notion of α-conversion given yesterday, we can define the following relation:

Definition 1. *The relation $\to_\alpha$ is defined as*

$$(\lambda x.M) \to_\alpha (\lambda y.M)[x := y], y \notin FV(M) \tag{3}$$

We can again take multiple steps at a time in this relation, which we will denote as $\twoheadrightarrow_\alpha$. This relation actually characterizes an equivalence relation, which we'll denote as $=_\alpha$.

Proposition 1. *$=_\alpha$ is an equivalence relation*

Proof. This proof is left as an exercise to the reader. Symmetry is the interesting case. □

β-reduction The driving force behind function evaluation is β-reduction, which we define as follows:

Definition 2. *The relation $\rightarrow_\beta$ is defined as*

$$(\lambda x.M)N \rightarrow_\beta M[x := N] \tag{4}$$

We can also extend $\rightarrow_\beta$ to an equivalence relation by completing $\twoheadrightarrow_\beta$ with $=_\alpha$ and a symmetric closure. We call this relation $\equiv_\beta$.

For notational completeness, we call lambda terms of the form $(\lambda x.M)N$ redexes and $M[x := N]$ contractum.

Finally, we want to avoid variable capture. We do this by α-converting the offending abstraction to bind a different variable using substitution as defined in 2.2.

η-conversion We want to formalize extensionality. To do this, we define the following rewrite rule:

Definition 3. *The relation $\rightarrow_\eta$ is defined as*

$$(\lambda x.Mx) \rightarrow_\eta M$$

3.1 Combinatory Logic

Combinatory logic is a rewrite system equivalent to the lambda calculus. We define the grammar of combinatory logic as follows:

$$M, N := x \mid K \mid S \mid I \mid MN$$

Each of these combinators are defined by a single rewrite rule each:

$$\begin{aligned} KMN &\rightarrow M \\ SMNP &\rightarrow MP(NP) \\ IM &\rightarrow M \end{aligned}$$

I, which is the identity function, can be defined in terms of the other two combinators. In fact, $I = SKK$

Interestingly, each of these operators can be defined as lambda terms.

$$\begin{aligned} K &= (\lambda x.(\lambda y.x)) \\ S &= (\lambda x.(\lambda y.(\lambda z.xz(yz)))) \\ I &= (\lambda x.x) \end{aligned}$$

Over the course of the 1930's,

- Curry proved the equivalence of lambda calculus and combinatory logic,
- Kleene proved the equivalence of lambda calculus and recursive functions, and
- Turing proved the equivalence of lambda calculus and Turing machines.

3.2 Examples

We use a shorthand notation for the combination of α-conversion and many steps of β-reduction to model the evaluation of a lambda calculus term to its normal form:

$$\twoheadrightarrow \triangleq \twoheadrightarrow_\beta \cup \rightarrow_\alpha$$

$xyzx$	normal form NF
$I = \lambda x.x$	normal form NF
$K = \lambda xy.x$	normal form NF
$S = \lambda xyz.xz(yz)$	normal form NF
$KI(KII)$	strongly normalizing SN
$KI\Omega$	normalizing N
$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	head normalizing HN (solvable)
$\Omega = \Delta\Delta = (\lambda x.xx)(\lambda x.xx)$	unsolvable

In $KI\Omega$, we get to observe an interesting property that was hinted at yesterday - a non-terminating term with a normal form. This is because it depends on which order the arguments are evaluated in.

3.3 Confluence

We begin with an important theorem:

Theorem 1. *If $M \twoheadrightarrow N$ and $M \twoheadrightarrow P$, then there exists some S such that both $N \twoheadrightarrow S$ and $P \twoheadrightarrow S$.*

This is hard to prove and beyond the scope of this lecture. Try to prove it yourself or study existing proofs at your own risk.

This theorem has a litany of corollaries, which we can list here:

Corollary 1. *The following are directly implied by Theorem 1:*

1. *If $M \twoheadrightarrow N$ and $M \twoheadrightarrow P$, then $P \equiv_\beta Q$*
2. *Normal forms are unique*
3. *The order of reductions is arbitrary*
4. *Reductions can be executed in parallel*

3.4 Forms of Lambda Terms

$$(\lambda x_1 ... x_n.(((\lambda y.P))Q)Q_1 ... Q_k), n \geq 0, k \geq 0 \tag{5}$$

$$(\lambda x_1 ... x_n.xQ_1 ... Q_k), n \geq 0, k \geq 0 \tag{6}$$

The former is called the head redex form, and the latter is called the head normal form. If, in the head normal form, Q_i are normal for all i, then it is also in normal form.

4 Reduction Strategies

While it technically doesn't matter what order we do reductions in due to Church-Rosser, we do want to have some standard way of evaluating terms. This is the idea behind standardization, which fixes a reduction strategy. The main intuition is to perform head reduction while the term is in head redex form, then move through the Q terms in increasing index and recursively repeat. We call the relation characterized by this strategy $\twoheadrightarrow_s$. This strategy has a nice theorem:

Theorem 2. *If $P \twoheadrightarrow S$ then $P \twoheadrightarrow_h M \twoheadrightarrow_i S$, and if $P \twoheadrightarrow S$ then $P \twoheadrightarrow_s S$.*

where $\twoheadrightarrow_h$ is defined as head reduction and $\twoheadrightarrow_i$ is defined as internal reduction.

However, there are alternative reduction strategies, some of which we will now detail.

4.1 Perpetual Strategies

In some sense, these are adversarial strategies - if there exists a reduction strategy which will lead to non-termination, these strategies will find at least one of them.

Definition 4. *A strategy $\rightarrow_e \subseteq \rightarrow$ is perpetual if for every $M \in \bigwedge$ which is not strongly normalizing, there exists an infinite reduction path $M_1 \rightarrow_e M_2 \rightarrow_e M_3 \rightarrow_e M_4 ...$*

One such perpetual strategy is the following:

$$F_\infty(M) = \begin{cases} M & \text{if N is a normal form} \\ Otherwise & \text{if } M \equiv C[R], R \equiv ((\lambda x.P)Q) \text{the leftmost redex} \\ \begin{cases} C[P[x := Q]] & R \text{ is } I\text{-redex} \\ Otherwise & R \text{ is } K\text{-redex} \\ \begin{cases} C[P] & Q \text{ normal} \\ C[((\lambda x.P))F_\infty(Q)] & \text{otherwise} \end{cases} \end{cases} \end{cases}$$

This strategy can be characterized neatly in a theorem.

Theorem 3.
1. *If M is not strongly normalizing, then $M = M_0 \rightarrow_p M_1 \rightarrow_p M_2 \rightarrow_e ...$ where $M_{i+1} = F_\infty(M_i), i = 0, 1...$*
2. *If M is strongly normlaizing then $M = M_0 \rightarrow_p M_1 \rightarrow_p M_2 ... M_k$, where $M_{i+1} = F_\infty(M_i), i = 0, 1...k-1$ is the longest reduction starting from n*