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 Rewriting Basics

The main idea of rewriting is substitution - replacing a subexpression of a formula with another formula. Today we care about term rewriting and higher order term rewriting, but other examples are graph rewriting, string rewriting, and trace rewriting.

Example 1. *Commutative groups are defined over the signature $(A, *, e)$ as follows:*

$$\begin{aligned}(a*b)*c &\to a*(b*c)\\ e*a &\to a\\ a*a^{-1} &\to e\\ a*b &\to b*a\end{aligned}$$

where $e, a, b, c \in A$. These rules define a rewrite system.

Other examples include classical propositional logic, as well as arbitrary rewrite systems.

Definition 1. *A rewrite system is defined over the signature* $(A, \rightarrow)$ *where* $\rightarrow$: $A \rightarrow A$ *and* $\rightarrow$ *has the following properties: it is deterministic, it is composable with itself, and* $\twoheadrightarrow$ *is the reflexive transitive closure of* $\rightarrow$.

A rewrite system can have three major properties:

1. Confluence
2. Normalization
3. Termination

2.1 Confluence

Definition 2. *A rewrite system* $(A, \rightarrow)$ *is confluent if* $\twoheadleftarrow \cdot \twoheadrightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$.

Where the special arrow notations means:

$$\twoheadleftarrow \cdot \twoheadrightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow \triangleq \forall M \exists N : \{(P,Q) \mid M \twoheadrightarrow P, M \twoheadrightarrow Q\} \subseteq \{(P,Q) \mid P \twoheadrightarrow N, Q \twoheadrightarrow N\}$$

Definition 3. *A rewrite system* $(A, \rightarrow)$ *is weakly confluent if* $\leftarrow \cdot \rightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$.

Where similarly the arrow notation means:

$$\leftarrow \cdot \rightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow \triangleq \forall M \exists N : \{(P,Q) \mid M \rightarrow P, M \rightarrow Q\} \subseteq \{(P,Q) \mid P \twoheadrightarrow N, Q \twoheadrightarrow N\}$$

Example

Consider the reduction of mathematical formula $5 \cdot (3+2)$. This can be done in the following two ways

- $5 \cdot (3+2) = 5 \cdot 3 + 5 \cdot 2 = 25$
- $5 \cdot (3+2) = 5 \cdot 5 = 25$

Note that in both directions, we get the same reduced form. This is precisely what the idea of confluence is. No matter which reduction strategy we choose, we end up with the same reduced form which is unique.

2.2 Normalization and Termination

Definition 4. *A term* $n \in A$ *is normal if there exists no* $s \in A$ *such that* $n \rightarrow s$

We say that n is a normal form of t if $t \twoheadrightarrow n$ and n is normal.

Definition 5. *A rewrite system* $(A, \rightarrow)$ *is weakly normalizing if every* $t \in A$, t *has a normal form.*

Note that again, because the relation is not a function, this only implies that at least one rewrite sequence ends in a normal form. If we further restrict this condition to forcing all rewrite sequences to a normal form, we call it strongly normalizing (or terminating).

Finally, we define non-terminating.

Definition 6. *An abstract rewrite system $(A, \to)$ is non-terminating if there exists at least one $t \in A$ such that there exists a reduction sequence on t that does not end in a normal form.*

Note that a system can be both non-terminating and weakly normalizing at the same time.

We then have some nice theorems about confluence and normalization:

Theorem 1. *Confluence implies uniqueness of normal form*

Proof. Suppose a term t has two normal forms, n_1, n_2. Therefore $t \twoheadrightarrow n_1$ and $t \twoheadrightarrow n_2$. By confluence, there exists some u such that $n_1 \twoheadrightarrow u$ and $n_2 \twoheadrightarrow u$. However, this is a contradiction as n_1 and n_2 are both normal and thus cannot step and by hypothesis are not equal, so we're done. □

Theorem 2. *A terminating abstract rewriting system is confluent if and only if it is locally confluent*

Proof. Based on well-founded induction. □

The existing proofs of this are hard, and reading one is a homework.

Example 2. *We define a simple step relation as follows:*

$$\begin{aligned} f(x, x) &\to g(x) \\ f(x, g(x)) &\to b \\ h(a, x) &\to f(h(x, a), h(x, x)) \end{aligned}$$

where a, b, f, g, h are constants.

This rewrite system is normalizing, but neither confluent nor terminating.

2.3 Reduction Strategies

A reduction strategy is a subset of the reduction strategy - i.e. turning the step relation into a step function. We call such a strategy a normalization strategy if the strategy reaches a normal form if a normal form exists, and doesn't terminate otherwise.

Definition 7. *A term rewriting system (TRS) is an abstract rewriting system (ARS) whose objects are representable as an inductively defined grammar.*

We call a TRS a higher-order term rewriting system (HOR) if it allows higher order functions and bound variables.

3 Lambda Calculus

3.1 Some History

Lambda calculus was proposed by Alonzo Church in the 1930's to study computable functions and the foundations of mathematics. For a long time it was unknown it was able to encode Peano arithmetic until Stephen Kleene went to the dentist to have a tooth removed and had a revelation about the predecessor function.

In 1921, Moses Schönfinkel published his work on combinators in order to avoid having to manipulate variables. Haskell Curry rediscovered and developed the theory of combinators further. Much later, David Turner used combinators to implement a compiler for a precursor of Haskell called Miranda using combinators as intermediate representation.

3.2 Free Variables vs Functions

In mathematics, free variables are often used in questions asking the student to find an unknown value under some given constraints. Functions bind variables and represent a computation.

Functions in lambda calculus are anonymous.

For HOR, we generally want three features:

1. We want the name of the argument to be unimportant. For example, we want the function $\lambda x.x + 1$ to be equivalent to the function $\lambda y.y + 1$. We call this property α-conversion.
2. We want to be able to evaluate function applications - i.e. we want to be able to substitute terms in for bound variables in a principled manner. We call this property β-conversion (or reduction).
3. We want the names of the free variables in terms fed into functions to be irrelevant. This will (hopefully) eventually be called safe substitution.

Furthermore, functions in the lambda calculus are curried, which means that functions of multiple arguments are represented by nested lambda abstractions which are applied iteratively to each individual argument.

3.2.1 Syntax

1. $V = \{x, y, z, x_1, \dots\}$ a countable set of variables
2. $C = \{a, b, c, a, \dots\}$ a countable set of constants

Definition 8. *The set Λ of (lambda) terms is inductively defined.*

- $V \subseteq \Lambda$ *and* $C \subseteq \Lambda$ *atomic terms*
- $M, N \in \Lambda$*, then* $(M\ N) \in \Lambda$ *application*

- $M \in \Lambda, x \in V$ *then* $(\lambda x.M) \in \Lambda$

Finally, to close off the lecture, we define the syntax of the lambda calculus as follows:

$$M, N ::= x | MN | \lambda x.M \tag{1}$$

where application binds tighter than abstraction.

4 Homework

To find reductions that lead to the normal form and prove that $h(a, a)$ has an infinite derivation.