Church-Rosser Theorems

Confluence in Lambda Calculus

Amirreza Khakpour

September, 2025

A Bit of History

- Developed by Alonzo Church and his student J. Barkley Rosser in the 1930s
- Part of the foundations of computability theory and functional programming
- Lambda calculus emerged as an alternative to Turing machines for formalizing computation
- Church-Rosser theorems establish fundamental properties about reduction strategies

What is Lambda Calculus?

A formal system for expressing computation using:

- **Variables**: $x, y, z, \ldots$
- **Abstraction**: $\lambda x.M$ (function creation)
- **Application**: $M\ N$ (function application)

Examples:

- Identity function: $\lambda x.x$
- Constant function: $\lambda x.\lambda y.x$
- Function application: $(\lambda x.x)\ y$

Lambda Calculus Syntax (Formally)

$$
\begin{aligned}
M, N ::= {} & x \quad \text{(variable)} \\
\mid {} & \lambda x.M \quad \text{(abstraction)} \\
\mid {} & M\ N \quad \text{(application)}
\end{aligned}
$$

Parentheses convention: Application associates to the left

$$M\ N\ P = (M\ N)\ P$$

Example: $\lambda f.\lambda x.f\ (f\ x)$ means $\lambda f.(\lambda x.(f\ (f\ x)))$

Free and Bound Variables

- **Bound variable**: Occurs in the body of a lambda abstraction
- **Free variable**: Not bound by any enclosing lambda

Example: In $\lambda x.x\ y$

- x is bound (by λx)
- y is free

Alpha conversion: Renaming bound variables

$$\lambda x.x \equiv \lambda y.y$$

Beta Reduction (β-reduction)

The main computational rule: function application

$$(\lambda x.M)\ N \to_\beta M[x := N]$$

where $M[x := N]$ means substitute N for all free occurrences of x in M.

Example:

$$(\lambda x.x\ x)\ y \to_\beta y\ y$$

$$(\lambda x.\lambda y.x)\ z \to_\beta \lambda y.z$$

The Omega Combinator: Infinite Reduction!

Consider this fascinating term:

$$\Omega = (\lambda x.x\ x)(\lambda x.x\ x)$$

Let's reduce it:

$$\Omega \to_\beta (\lambda x.x\ x)(\lambda x.x\ x) \to_\beta (\lambda x.x\ x)(\lambda x.x\ x) \to_\beta \cdots$$

This reduces forever! No normal form exists.

Intuition: A function that applies itself to itself, creating an infinite loop.

More Non-Terminating Examples

- **Omega combinator**: $\Omega = (\lambda x.x\ x)(\lambda x.x\ x)$
- **Y combinator** (fixed-point combinator):

$$Y = \lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x))$$

- **Application to itself**: $\lambda x.x\ x$

Crucial insight: Some lambda terms have normal forms, some don't - just like some programs halt and some don't!

Eta Reduction (η-reduction)

Eliminates redundant abstractions:

$$\lambda x.M\, x \to_\eta M \quad \text{(if } x \text{ not free in } M\text{)}$$

Example:

$$\lambda x.(\lambda y.y)\, x \to_\eta \lambda y.y$$

Intuition: Two functions are extensionally equal if they produce the same output for all inputs.

Reduction Strategies

- **Normal order**: Reduce leftmost outermost redex first
- **Applicative order**: Reduce leftmost innermost redex first
- **Full reduction**: Reduce any redex in any order

Crucial difference: Normal order **will find** a normal form if one exists!

Example: $(\lambda x.\lambda y.y)(\Omega)$

- Normal order: $\rightarrow \lambda y.y$ (found normal form!)
- Applicative order: Gets stuck reducing Ω forever

The Halting Problem Connection

Theorem (Undecidability of Normal Form)
There is no algorithm that can decide, for an arbitrary lambda term M, whether M has a normal form.

This is equivalent to the Halting Problem!

Why? We can encode Turing machines as lambda terms:

- Turing machine halts $\Leftrightarrow$ corresponding lambda term has normal form
- Reduction in lambda calculus $\Leftrightarrow$ computation steps in Turing machine

Original 1936 Paper Context

”Some Properties of Conversion” by Alonzo Church and J. Barkley Rosser

- Published in Transactions of the American Mathematical Society, 1936
- Uses original terminology: **conversion** (= reduction + expansion)
- Defines: *A* conv *B* means *A* and *B* are interconvertible
- **Reduction**: Replacing a redex by its contractum
- **Expansion**: Reverse of reduction

Original Church-Rosser Theorems

The 1936 paper contains three main theorems:

1. **Theorem 1**: Conversion sequencing theorem
2. **Theorem 2**: Normal form reachability theorem
3. **Theorem 3**: Bound on reduction sequences theorem

Important: These are more nuanced than the modern ”diamond property” version!

Theorem 1: Conversion Sequencing

Theorem (Church-Rosser Theorem 1, 1936)
If A conv B, then there is a conversion from A to B in which no expansion precedes any reduction.

Modern interpretation: Any equivalence proof can be rearranged as:

$$A \to^* C \leftarrow^* B$$

where all reductions happen first, then all expansions.

Significance: Establishes that β-equivalence classes have a ”diamond” structure.

Theorem 1 Visualized

$$\begin{tikzcd}
A \arrow[rr, dashed, leftrightarrow, "\text{original conversion}"] \arrow[d, "\text{reductions only}"'] & & B \\
C \arrow[urr, "\text{expansions only}"'] & &
\end{tikzcd}$$

Example: If $A \leftarrow X \rightarrow Y \leftarrow B$, then there exists C with $A \rightarrow^* C \leftarrow^* B$.

Theorem 2: Normal Form Reachability

Theorem (Church-Rosser Theorem 2, 1936)
If B is a normal form of A, then there is a number m such that any sequence of reductions from A will lead to B (to within applications of alpha conversion) after at most m reductions.

Modern interpretation:

- If a normal form exists, it's reachable in bounded steps
- All reduction sequences of sufficient length will find it
- Accounts for alpha conversion (renaming bound variables)

Theorem 2 Example

Consider: $(\lambda x.x\ x)((\lambda y.y)\ z)$

- Normal form: $z\ z$
- Maximum needed reductions: 3
- Any reduction sequence of length ≥ 3 will reach $z\ z$ (up to alpha)

Contrast with Ω: $(\lambda x.x\ x)(\lambda x.x\ x)$ has no normal form, so no such m exists.

Theorem 3: Bound on Reduction Sequences

Theorem (Church-Rosser Theorem 3, 1936)
If A has a normal form, then there is a number m such that at most m reductions of order one can occur in a sequence of reductions on A.

Order one reduction: Reducing a redex that is not contained in any other redex.

Significance: There's a finite bound on how many "top-level" reductions can occur, regardless of reduction strategy.

Understanding ”Order One” Reductions

Order one: Outermost redexes not contained in other redexes.

Example: In $(\lambda x.x\ x)((\lambda y.y)\ z)$

- Order one: $(\lambda x.x\ x)((\lambda y.y)\ z)$ itself
- Not order one: $(\lambda y.y)\ z$ (contained in the larger redex)

Theorem 3 says: Only finitely many such top-level reductions possible if normal form exists.

Normal Order Evaluation to the Rescue!

Theorem (Standardization Theorem)
If a term has a normal form, normal order reduction will find it.

Example: $(\lambda x.\lambda y.y)(\Omega)$

- Normal order: Reduces to $\lambda y.y$ (success!)
- Applicative order: Gets stuck reducing Ω forever

Practical significance: This is why lazy evaluation (like in Haskell) can handle infinite data structures!

Relationship to Modern Church-Rosser

- **Modern version**: Usually stated as confluence: $M \to^* N_1$ and $M \to^* N_2$ implies exists P with $N_1 \to^* P$ and $N_2 \to^* P$
- **Original Theorem 1**: Stronger - deals with full conversion (reduction + expansion)
- **Original Theorems 2 & 3**: Provide quantitative bounds not in modern statements

Historical note: The "diamond property" proof came later and is often easier to work with.

Proof Sketch (High Level)

1. Define **parallel reduction**: Reduce multiple non-overlapping redexes simultaneously
2. Prove diamond property for parallel reduction
3. Show that the transitive closure of parallel reduction equals ordinary reduction
4. Conclude that ordinary reduction has the Church-Rosser property

Key insight: By allowing parallel reductions, we avoid the ”diamond problem” where reductions interfere with each other.

Significance and Applications

- **Functional programming**: Evaluation order doesn't affect final result (if it exists)
- **Compiler optimization**: Freedom to rearrange computations
- **Theorem proving**: Basis for rewriting systems and equality reasoning
- **Programming language theory**: Foundation for many type systems
- **Undecidability**: Shows fundamental limits of computation

Summary: Key Takeaways

- Lambda calculus provides a foundation for computation
- Some terms have normal forms, some don't (like Ω)
- Normal form existence is undecidable (equivalent to halting problem)
- Original Church-Rosser theorems provide deep insights about conversion
- Theorems 2 and 3 give quantitative bounds on reduction sequences
- Normal order evaluation will find a normal form if one exists

Questions?