The SKI Combinator Calculus
a universal formal system

The ceremony

alphabet The *alphabet* for the combinator calculus is the set

$$\Sigma = \{\mathbf{S}, \mathbf{K}, \mathbf{I}, (,)\}$$

terms The set T of *terms* is defined recursively as follows

1. $\mathbf{S}$, $\mathbf{K}$, and $\mathbf{I}$ are terms.
2. If τ_1 and τ_2 are terms, then $(\tau_1\tau_2)$ is also a term.
3. Nothing is a term unless required to be by rules 1 and 2.

derivations A derivation is a finite sequence of terms satisfying the following rules.

1. If Δ is a derivation ending in the term $\alpha((\mathbf{K}\beta)\gamma)\iota$, where the parentheses shown form matching pairs, then Δ followed by $\alpha\beta\iota$ is also a derivation.
2. If Δ is a derivation ending in the term $\alpha(\mathbf{I}\beta)\iota$, where the parentheses shown form a matching pair, then Δ followed by $\alpha\beta\iota$ is also a derivation.
3. If Δ is a derivation ending in the term $\alpha(((\mathbf{S}\beta)\gamma)\delta)\iota$, where the parentheses shown form matching pairs, then Δ followed by $\alpha((\beta\delta)(\gamma\delta))$ is also a derivation.

What the ceremony tells us

The ceremony above captures much of the conventional style in which logicians present the combinator calculus. But the conventional ceremony describing the combinator calculus does not match the natural structure of the formal system as well as the conventional ceremony for binary incrementing matched its underlying system.

The terms of the combinator calculus are certain finite linear sequences of symbols from its alphabet, restricted by the requirement that parentheses are well balanced, and that there are never more than 2 **S**, **K**, or **I** symbols in a row without an intervening parenthesis. The formal system can in fact be understood correctly by conceiving terms as sequences. But it is much more natural to understand the calculus as operating on binary tree-structured terms with the symbols **S**, **K**, **I** at the leaves. The parentheses are in some sense only there to indicate the tree structure, and shouldn't be regarded as part of the abstract alphabet. On the other hand, perhaps there should be a symbol to associate with the nonleaf elements in the trees, in the same way that there is an implicit symbol for multiplication in the numerical term $2\pi t$.

Even those who insist on understanding combinatory terms as linear sequences get irritated with all of the parentheses. So, they introduce conventions for leaving out parentheses on the left, and write $((\alpha\beta)\gamma)$ as $\alpha\beta\gamma$, but retain the parentheses in $\alpha(\beta\gamma)$. This is very similar to the omission of parentheses in numerical terms. If you're familiar with it, you won't need an explanation. If you're not, skip over it for now, since it's a minor side-issue. Really conventional presentations of the combinator calculus introduce the omission of parentheses as an abbreviation even before they get to the definition of derivation. Figure 1 shows an example of the same combinatory term presented with full parentheses, minimal parentheses, and as a tree diagram. The abstract form of the term is the same no matter which presentation we use, although the sameness is a bit subtle, since it depends on the power of parenthesized linear sequences to represent trees.

The view of derivations as linear sequences is natural enough, so we won't consider varying that. The rules for derivations are shown graphically in Figure 2. The English description of the rules is too long and tangled to be worth inspecting here. The pictures should be clear enough, as long as we understand that

- The system deals entirely with finite binary branching tree diagrams,

	Minimal parentheses	**Full parentheses**	**Tree diagram**
1.	**ISK**	((**IS**)**K**)	
2.	**I**(**SK**)	(**I**(**SK**))	
3.	**SKISKI**	(((((**SK**)**I**)**S**)**K**)**I**)	
4.	**S**(**KI**)(**SK**)**I**	((**S**((**KI**)(**SK**))**I**)	
5.	**S**(**K**(**I**(**S**(**KI**))))	(**S**(**K**(**I**(**S**(**KI**)))))	

In the forms with minimal parentheses, not every subsequence of symbols is well formed (i.e., has balanced implicit parentheses). In 1, notice that **SK** is *not* a balanced subterm, and does not correspond to a subtree in the tree diagram. If 3 arose in a derivation, we could not apply Rule 2 to the form **KIS**, because that is not a well-formed portion, nor could we apply Rule 1 to the form **IS**. Similarly in 4, (**KI**)(**SK**) and (**SK**)**I** are not well formed.

Figure 1: Terms with minimal parentheses, full parentheses, and as tree diagrams

Figure 2: Derivation rules for the Combinator Calculus

This derivation takes two steps with *Rule 2* to derive a tree with two **S**s. Intuitively, the leftmost two **K**s select the red **S** from the surrounding **K**s. The five-sided dashed figures are not part of the derivation: they just show where the rules are applied.

Figure 3: A derivation in the Combinator Calculus

where the end of each path is labelled with exactly one of the symbols **S**, **K**, or **I**. Such a tree diagram is called a *term*.

- You may start with any term.
- In Figure 2, the x, y, and z in dashed triangles may be replaced by any terms, as long as in each application of a rule, each of the x triangles is replaced by a copy of the same combinator, similarly for each of the y triangles and each of the z triangles.
- When a structure of the form given by the left-hand side of one of the two rules in Figure 2 appears anywhere within a term, you may replace that structure by the corresponding structure on the right-hand side of the same rule.

Compare this (I think clearer) presentation of the rules of derivation with the more ceremonial one, and convince yourself that they are two different descriptions of the same abstract notion of derivation. Notice how the metasymbols β, γ, and δ in the ceremonial version serve the same function as the metasymbols x, y, and z in the second version—they act as variables ranging over terms. The metasymbols α and ι in the ceremonial version correspond to the explanation that we may replace a structure "anywhere within a term."

In the formal system of the Combinator Calculus, we may replace a certain combination of four '**K**'s and two '**S**'s by the combination of the two '**S**'s, using the derivation in Figure 3. Writing terms as linear sequences, this derivation is described as

KKKSKS, **KSKS**, **SS**

In an interesting formal system, such as the combinator calculus, we usually get bored with doing one derivation at a time. We notice that derivations

This schematic derivation takes two steps, first with *Rule 3* and then with *Rule 2*, to reach the schematic tree at the bottom. Intuitively, the combination of one **S** and two **K**s above acts as an *identity operator* applied to the tree that fills in for a.
The five-sided dashed figures are not part of the derivation: they just show where the rules are applied.

Figure 4: A schematic derivation in the Combinator Calculus

often manipulate only certain portions of the terms in them, and other portions just come along for the ride. By carefully sorting out the manipulated portions and the inert portions, we generate *schematic derivations*, representing an infinite number of possibilites in a compact form. Figure 4 shows an interesting schematic derivation. This schematic derivation shows that **SKK** behaves like **I**. Writing terms as linear sequences, it looks like

$$\mathbf{SK}\mathbf{K}\alpha\beta,\ \mathbf{K}\alpha(\mathbf{K}\alpha)\beta,\ \alpha\beta$$

Make sure that you understand precisely why I needed parentheses in the second term, but not in the first or third. Notice that the symbols α and β are not part of any derivation. Rather, we can replace α and β by any terms that we choose, and the results are all derivations.

Here are some more derivations and schematic derivations, written with terms as minimally parenthesized linear sequences. In each term, I underlined the portion that is about to be replaced by one of the rules, and I overlined the portion that was created by application of a rule to the previous term. As an exercise, you should fill in the missing parentheses, and draw the tree diagrams.

$$\underline{\mathbf{SII}\alpha},\ \overline{\underline{\mathbf{I}\alpha}(\mathbf{I}\alpha)},\ \overline{\alpha}(\underline{\mathbf{I}\alpha}),\ \alpha\overline{\alpha} \tag{1}$$

$$\underline{\mathbf{SII}(\mathbf{SII})},\ \overline{\mathbf{I}(\mathbf{SII})(\underline{\mathbf{I}(\mathbf{SII})})},\ \underline{\mathbf{I}(\mathbf{SII})}\overline{(\mathbf{SII})},\ \overline{\mathbf{SII}}(\mathbf{SII}) \tag{2}$$

$$\begin{gathered}\underline{\mathbf{S}(\mathbf{K}(\mathbf{SI}))(\mathbf{S}(\mathbf{KK})\mathbf{I})\alpha\beta},\ \overline{\underline{\mathbf{K}(\mathbf{SI})\alpha}(\mathbf{S}(\mathbf{KK})\mathbf{I}\alpha)\beta},\ \overline{\mathbf{SI}}(\underline{\mathbf{S}(\mathbf{KK})\mathbf{I}\alpha})\beta,\\ \mathbf{SI}(\overline{\underline{\mathbf{KK}\alpha}(\mathbf{I}\alpha)})\beta,\ \mathbf{SI}(\overline{\mathbf{K}}(\underline{\mathbf{I}\alpha}))\beta,\ \underline{\mathbf{SI}(\mathbf{K}\overline{\alpha})\beta},\ \overline{\underline{\mathbf{I}\beta}(\mathbf{K}\alpha\beta)},\ \overline{\beta}(\underline{\mathbf{K}\alpha\beta}),\ \beta\overline{\alpha}\end{gathered} \tag{3}$$

Derivation 1 shows that **SII** behaves as a sort of *repeat* or *self-apply* operation. Derivation 2 shows the circularity in repeating repeat, or self-applying self-apply. Derivation 3 is rather challenging to follow. It shows that **S**(**K**(**SI**))(**S**(**KK**)**I**) behaves as a sort of reversal operation. Try to see how the **S**s serve to shuffle copies of α and β into different parts of the term. **K**(**SI**) acts as a sort of filter to throw away the α and catch the β; conversely **S**(**KK**)**I** acts as a sort of filter to catch the α and throw away the β.

Because the rules for derivations all depend on the appearance of a particular symbol at the left, we often call the form $\alpha\beta$ "α applied to β," and in general we call $\alpha\beta_1 \ldots \beta_n$ "α applied to $\beta_1, \ldots, \beta_n$."

The universal qualities of SKIing

The combinator calculus was designed precisely to be *universal* in the sense that it can accomplish every conceivable rearrangement of subterms just by means of applying terms to them. That is, given a rule for rearranging n things into the shape of a term (allowing copying and deleting of individual things), there is a term that can be applied to each choice of n things so that several derivation steps will accomplish that rearrangement. The examples of **SII** as a repeater and **S**(**K**(**SI**))(**S**(**KK**)**I**) as a reverser suggest how this works. That particular quality of a formal system is called *combinatory completeness*. Every formal system that contains something acting like **S** and something acting like **K** is combinatorily complete (**SKK** acts like **I**, so we can actually do without **I**, but interesting terms get even harder to read). Combinatory completeness can itself be defined formally in a sense that we explore further in the section on reflection.

Rearrangements arise in formal systems whenever we substitute things for variables. The combinator calculus was designed specifically to sow that substitution for variables can be reduced to more primitive looking operations.

By accident, the combinator calculus turns out to be universal in a much more powerful sense than combinatory completeness. The combinator calculus is a *universal programming system*—its derivations can accomplish everything that can be accomplished by computation. That is, terms can be understood as programs, and every program that we can write in every programming language can be written also as a term in the combinator calculus. Since formal systems are the same thing as computing systems, every formal system can be described as an interpretation of the terms in the combinator calculus. When we suggest all of the ways that formal systems can be applied to one another in the sections on mathematical formalism and on reflection, this should look pretty impressive for a system with such trivial rules.

The universality of the combinator calculus in this sense cannot be defined perfectly formally. It is essentially a nonformal observation, called the *Church-Turing thesis* (often just *Church's thesis*). For every particular computing system that anyone has conceived of so far, we have precise formal knowledge that the combinator calculus can accomplish the same computations. There are some very strong arguments, particularly by Alan Turing, that some of these computing systems have captured the ability to do everything that can conceivably be regarded as a computation. But every attempt to formalize that observation begs the question whether the formalization of formalization is complete. Nonetheless, everybody that I know who studies such things finds the thesis convincing.

Based on the primitive quality of the operations in the combinator calculus, and its ability (given the Church-Turing thesis) to describe all possible formal systems, I like to think of the combinator calculus as the machine language of mathematics.