A NOTE ON THE S COMBINATOR

by

Jon Shultis

CU-CS-304-85 July, 1985

University of Colorado, Department of Computer Science,
Boulder, Colorado.

A Note on the **S** Combinator

Jon Shultis
Department of Computer Science
University of Colorado
Boulder, Colorado 80309

Abstract

Combinatory logic is becoming important as a basis for functional language implementation. However, the combinators often seem mysterious or arbitrary, and are understood, if at all, purely as mechanical transformations. The purpose of this note is to show that the combinators, especially **S**, are neither mysterious nor arbitrary, but have a simple explanation in terms of familiar concepts from ordinary logic.

Keywords: programming languages, theory of computation

A Note on the S Combinator

Jon Shultis
Department of Computer Science
University of Colorado
Boulder, Colorado 80309

1. Combinators as Propositional Logic

The initiate to the world of combinatory logic is likely to be puzzled by the apparent arbitrariness of the combinators. For many, they eventually acquire meaning through familiarity with their peculiar internal workings, but remain at heart a mystery. In reconciling ourselves to them, it helps that some of them are at least *comprehensible*! **I** corresponds to the "identity function", **B** is "functional composition", and **K** is clearly useful for discarding unwanted information. But why are these particular things so important? Worst of all: what, oh what, is **S**?

To Schönfinkel, **S** was the "fusion function", of which he wrote [4] "Clearly, the practical use of the function **S** will be to enable us to reduce the number of occurrences of a variable - and to some extent also of a particular function - from several to a single one."† On the topic of **S** Curry and Feys wax uncharacteristically mystical [2], p. 184: "When Schönfinkel was discovered in a literature search, **K** was added to the theory at once; but **S** was regarded as a mere technicality until the development of the newer axiomatic theories in the 1940's." By way of clarifying this remark, they write on p. 237: "None of these treatments used the Schönfinkel combinator **S**, and it was not perceived how that combinator could be used to make definitions by structural induction as in the present §A." This sounds suspiciously like saying that **S** is indeed just a technicality, but oh! what a handy one it is. In any event, I never found these remarks, nor the similar ones to

† Translation by Stefan Bauer-Mengelberg, in J. van Heijenoort, ed., *From Frege to Gödel: a source book in mathematical logic, 1879-1931*, Harvard University Press, 1967.

be found throughout the literature on combinatory logic, to be very illuminating.

The purpose of this note is to show that the combinators, especially **S**, are neither mysterious nor arbitrary, but have a simple explanation in terms of familiar concepts from ordinary logic. The story begins with a pair of axioms common to numerous formulations of propositional logic (see, e.g., [1]).

$A1$: $\vdash p \to (q \to p)$

$A2$: $\vdash (p \to (q \to r)) \to ((p \to q) \to (p \to r))$

These axioms are sufficiently well-known to be blessed with names. *A1* is the *law of affirmation of the consequent*; *A2* is the *self-distributive law of implication.*

The inference rules of interest to us are *substitution* and *modus ponens.* Using these rules, we can prove many propositional theorems, though *A1* and *A2* alone are incomplete by anyone's standards. For example, here is a proof of $(p \to p)$.

1. $\vdash p \to (q \to p)$ $\quad (A1)$
2. $\vdash (p \to (q \to r)) \to ((p \to q) \to (p \to r))$ $\quad (A2)$
3. $\vdash (p \to (q \to p)) \to ((p \to q) \to (p \to p))$ $\quad (2^r_p)$
4. $\vdash (p \to q) \to (p \to p)$ $\quad (modus\ ponens\ 3,1)$
5. $\vdash (p \to (q \to p)) \to (p \to p)$ $\quad (4^q_{(q \to p)})$
6. $\vdash (p \to p)$ $\quad (modus\ ponens\ 5,1)$

The story continues with a search for effective methods corresponding to the axioms. The "effective interpretation" of *A1* is that, given p, we can produce p when given q. The method for doing this is obvious; we hang on to p when it's given us, and produce it when given q, throwing q away. In lambda-notation, this method is expressed by $\lambda x.\lambda y.x$. The connection between the lambda-expression and *A1* is captured vividly by the polymorphic type of the expression.

$(\lambda x.\lambda y.x)\colon \alpha \to (\beta \to \alpha)$

In combinators, of course, this is **K**. So **K** is just the law of affirmation of the consequent.

The effective interpretation of $A2$ is that, given 1) a way to produce a method of obtaining r from q, given p, 2) a way to produce q from p, and 3) p, we can produce r. From 1) and 3) we obtain a method for producing r from q; from 2) and 3) we obtain q. Finally, from these two things we obtain r. In lambda-notation, this is just $\lambda x.\lambda y.\lambda z.(x\ z)(y\ z)$. Again, the type of the expression makes the connection vivid.

$(\lambda x.\lambda y.\lambda z.(x\ z)(y\ z))\colon (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$

So **S** turns out to be just the self-distributive law of implication. Carrying the analogy forward, substitution is type-matching, and *modus ponens* is application. If we use unification to do type-matching automatically, then these two inference rules are always carried out together, as a single rule, which we will call *application*, and denote by juxtaposition. Hence, we can rewrite our earlier proof of $(p \to p)$ in terms of combinators as follows.

1. $\vdash \mathbf{K}\colon \alpha \to (\beta \to \alpha)$
2. $\vdash \mathbf{S}\colon (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$
3. $\vdash \mathbf{SK}\colon (\alpha \to \beta) \to (\alpha \to \gamma)$
4. $\vdash \mathbf{SKK}\colon (\alpha \to \alpha)$

SKK, of course, reduces to **I**, the "identity" on all types.

It is only fair to note that Curry and Feys carry out this and many other proofs in their discussion of the theory of functionality. However, their choice of the notation $F(\alpha,\beta)$ for $(\alpha \to \beta)$ obscures the results considerably. The functionality of **S** becomes, in this notation, $F(F(\alpha,F(\beta,\gamma)),F(F(\alpha,\beta),F(\alpha,\gamma)))$, which is

hardly recognizable as the self-distributive law of implication! In his introductory text, Hindley [3] adopts this notation for functionality wholesale, thereby perpetuating the mystique of the combinators.

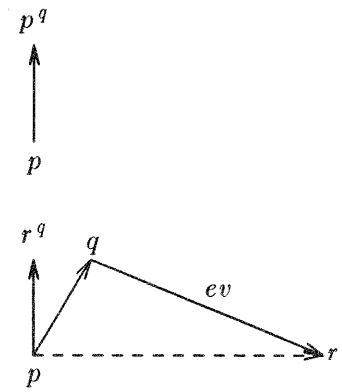
Have we succeeded in making the combinators seem less arbitrary? Perhaps. In any event, we can be comforted that they are not more mysterious than the axioms of propositional logic. Nonetheless, there is something less than completely satisfying about an "explanation" of one set of arbitrary rules in terms of another. Another possible source of dissatisfaction is the absence of any logical connective other than implication. We take up these issues in turn in the following sections.

2. A Categorial Analysis of S and K

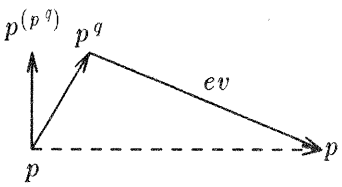
To explain the special significance of the combinators **S** and **K**, we begin with two intuitive principles which everyone will surely accept. The first is that an unconditional assertion is true under all conditions. In what follows, we shall call an unconditional assertion a *fact*, and a conditional assertion a *method* for establishing a fact under the stated conditions. With this terminology, the first principle becomes: "a method exists for establishing a fact under arbitrary conditions". The second principle is that whenever a method and its conditions coexist, it can be carried out. That a method and its conditions coexist means that there is a condition from which both derive, and that we have methods for establishing them from that condition.

Let $p, q, r, \cdots$ stand for arbitrary facts, and let p^q stand for methods of establishing p under the condition q. Finally, let $\alpha \to \beta$ denote that a method exists for establishing β from α.

Using this notation, our two principles can be diagrammed as follows.



The first of these diagrams corresponds to **K**, the second to **S**. The dashed arrow along the bottom of the **S** diagram is created by **S** from the two upward-pointing methods. For example, **S** produces the identity from p to p when these methods are both instances of **K**, as show below.



Category theorists will recognize something else in these diagrams, however.† The first diagram indicates that all objects p and q have an *exponent object*, p^q. The second is a variation on the usual diagram specifying the universal property of the exponent. *ev* is the *evaluation* map (the unit of the "Currying" adjunction).

The **S** diagram differs from the usual one in that both q and r^q are derived from a common domain, p. This peculiarity is forced on us by the fact that we don't have products. In fact, the diagram "cheats" as a result of this deficiency; to make it correct we would have to have a single object, viz. $r^q \times q$, as the domain of *ev*, instead of two disconnected objects. Because of the differences, we shall use the term "quasi-exponentials" to refer to the objects p^q. Nonetheless, the diagrams are compelling, and suggest that exponentiation is the really important part of

†The reader not versed in category theory may skip the remainder of this section without loss of continuity.

Cartesian-closedness for λ-models.

The interpretation of **S** and **K** as specifying categorial structure for λ-models raises an interesting problem when we consider reflexive domains. If the retraction $p \to p^p$ (which can always be constructed from **S** and **K** alone) is *iso*, then p is a reflexive domain. Scott has shown that there are nontrivial reflexive domains, and these can be used as models of untyped λ-calculi with η-conversion [5].

The problem is that $p^p \to p$, interpreted as a propositional formula, leads to inconsistency (in the sense of Post). From this formula and $\vdash p \to p$, we can conclude $\vdash p$ by application. Ordinarily, this would wreak havoc, because we could deduce *any* formula whatsoever via substitution. In the current situation, however, the problem vanishes because substitution, independent of *modus ponens*, is not a valid rule of inference for us! This is the ultimate reason for combining these into the rule of *application* in the first place. The effect is essentially similar to restricting the principle of comprehension in the manner of ZF.

We have already noted the lack of products, preventing us from having a full Cartesian-closed structure. We know, however, that products can be "simulated" in pure λ-calculus. The connection between these "internal" products and the "external" products of a Cartesian-closed category, and the problem of interdefinability of the logical connectives in general, is examined in the next section.

3. And, Other Connectives

We take up now the problem of defining a binary connective "∧", called "conjunction", which obeys the following laws.

$P1.\ \vdash p \wedge q \to p$

$P2.\ \vdash p \wedge q \to q$

$C.\ \vdash p \to (q \to p \wedge q)$

Abstractly, $p \wedge q$ represents the *product* of p and q. *P1* and *P2* are the first and second *projections*, respectively, and C is the *pairing* operation. Now, it happens that there is a standard way to represent products in pure λ-calculus, rendering these laws as shown below.

$p1 = \lambda x.\ x(\lambda y.\lambda z.y)\colon ((\alpha \to (\beta \to \alpha)) \to \gamma) \to \gamma$

$p2 = \lambda x.\ x(\lambda y.\lambda z.z)\colon ((\alpha \to (\beta \to \beta)) \to \gamma) \to \gamma$

$c = \lambda x.\lambda y.\lambda z.\ ((z\ x)\ y)\colon \alpha \to (\beta \to ((\alpha \to (\beta \to \gamma)) \to \gamma))$

Suppose now that the facts δ and ϵ are proved by methods d and e, respectively. We form the conjunction of these facts as follows.

1. $\vdash c\colon \alpha \to (\beta \to ((\alpha \to (\beta \to \gamma)) \to \gamma))$

2. $\vdash d\colon \delta$

3. $\vdash cd\colon \beta \to ((\delta \to (\beta \to \gamma)) \to \gamma)$

4. $\vdash e\colon \epsilon$

5. $\vdash cde\colon (\delta \to (\epsilon \to \gamma)) \to \gamma$

To select the first conjunct, we apply *p1*.

6. $\vdash p1\colon ((\alpha \to (\beta \to \alpha)) \to \gamma) \to \gamma$

7. $\vdash p1(cde)\colon \delta$

Similarly,

8. $\vdash p2(cde)\colon \epsilon$

We see that everything will work out nicely if we *define* $p \wedge q$ to be $(p \to (q \to r)) \to r$. All we need to do is to prove, as theorems, the types of the three λ-expressions corresponding to the three laws of products.

An easy way to develop any such proof is to use the abstraction algorithm to convert the pure λ-expression to a corresponding pure combinatory expression. For reference, we will use the algorithm in the following form.

```
let rec abstract(x,e) =  case e of
                           x: (S K K)
                           y: (K y)
                           (e1 e2): S(abstract(x,e1) (abstract(x,e2))
                           (λy.e1): abstract(x,abstract(y,e1))
                         esac
```

To convert c to a combinatory expression, we compute $abstract(x,\ \lambda y.\lambda z.(z\ x)\ y)$, which results in the following combination.

```
((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S))))
  ((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K))))))
  ((S ((S (K S)) ((S (K K)) (K K)))) ((S (K K)) (S K K)))))))
  ((S ((S (K S)) ((S (K K)) (K K)))) (K (S K K))))
```

A syntactic derivation of this expression follows.

1. K
2. S
3. S K
4. S K K
5. K (S K K)
6. K K
7. S (K K)
8. (S (K K)) (K K)
9. K S
10. S (K S)
11. (S (K S)) ((S (K K)) (K K))
12. (S ((S (K S)) ((S (K K)) (K K))))
13. ((S ((S (K S)) ((S (K K)) (K K)))) (K (S K K)))
14. (S (K K)) (S K K)
15. ((S ((S (K S)) ((S (K K)) (K K)))) ((S (K K)) (S K K)))
16. (S (K K)) (K (S K K))
17. (S (K K)) (K S)
18. (S (K S)) ((S (K K)) (K S))
19. (S ((S (K S)) ((S (K K)) (K S))))
20. ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K))))
21. (S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K))))
22. (S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K))))))
23. ((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K)))))) ((S ((S (K S)) ((S (K K)) (K K)))) ((S (K K)) (S K K))))
24. ((S ((S (K S)) ((S (K K)) (K S))))
((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K)))))) ((S ((S (K S)) ((S (K K)) (K K)))) ((S (K K)) (S K K)))))
25. ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S))))
((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K)))))) ((S ((S (K S)) ((S (K K)) (K K)))) ((S (K K)) (S K K))))))
26. (S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S))))
((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K)))))) ((S ((S (K S)) ((S (K K)) (K K)))) ((S (K K)) (S K K)))))))
27. ((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S))))
((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K (S K K)))))) ((S ((S (K S)) ((S (K K)) (K K)))) ((S (K K)) (S K K)))))))
((S ((S (K S)) ((S (K K)) (K K)))) (K (S K K))))

By writing down the corresponding sequence of types, we obtain a more conventional proof.

1. A1: $\alpha \to (\beta \to \alpha)$
2. A2: $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$
3. 2 1: $(\alpha \to \beta) \to (\alpha \to \alpha)$
4. 3 1: $\alpha \to \alpha$
5. 1 4: $\alpha \to (\beta \to \beta)$
6. 1 1: $\alpha \to (\beta \to (\gamma \to \beta))$
7. 2 6: $(\alpha \to \beta) \to (\alpha \to (\gamma \to \beta))$
8. 7 6: $\alpha \to (\beta \to (\gamma \to (\delta \to \gamma)))$
9. 1 2: $\alpha \to ((\beta \to (\gamma \to \delta)) \to ((\beta \to \gamma) \to (\beta \to \delta)))$
10. 2 9: $(\alpha \to (\beta \to (\gamma \to \delta))) \to (\alpha \to ((\beta \to \gamma) \to (\beta \to \delta)))$
11. 10 8: $\alpha \to ((\beta \to \gamma) \to (\beta \to (\delta \to \gamma)))$
12. 2 11: $(\alpha \to (\beta \to \gamma)) \to (\alpha \to (\beta \to (\delta \to \gamma)))$
13. 12 5: $\alpha \to (\beta \to (\gamma \to \beta))$
14. 7 4: $\alpha \to (\beta \to \alpha)$
15. 12 14: $\alpha \to (\beta \to (\gamma \to \alpha))$
16. 7 5: $\alpha \to (\beta \to (\gamma \to \gamma))$
17. 7 9: $\alpha \to (\beta \to ((\gamma \to (\delta \to \epsilon)) \to ((\gamma \to \delta) \to (\gamma \to \epsilon))))$
18. 10 17: $\alpha \to ((\beta \to (\gamma \to (\delta \to \epsilon))) \to (\beta \to ((\gamma \to \delta) \to (\gamma \to \epsilon))))$
19. 2 18: $(\alpha \to (\beta \to (\gamma \to (\delta \to \epsilon)))) \to (\alpha \to (\beta \to ((\gamma \to \delta) \to (\gamma \to \epsilon))))$
20. 19 16: $\alpha \to (\beta \to (((\gamma \to \delta) \to \gamma) \to ((\gamma \to \delta) \to \delta)))$
21. 10 20: $\alpha \to ((\beta \to ((\gamma \to \delta) \to \gamma)) \to (\beta \to ((\gamma \to \delta) \to \delta)))$
22. 2 21: $(\alpha \to (\beta \to ((\gamma \to \delta) \to \gamma))) \to (\alpha \to (\beta \to ((\gamma \to \delta) \to \delta)))$
23. 22 15: $\alpha \to (\beta \to ((\alpha \to \gamma) \to \gamma))$
24. 19 23: $\alpha \to (\beta \to (((\alpha \to (\gamma \to \delta)) \to \gamma) \to ((\alpha \to (\gamma \to \delta)) \to \delta)))$
25. 10 24: $\alpha \to ((\beta \to ((\alpha \to (\gamma \to \delta)) \to \gamma)) \to (\beta \to ((\alpha \to (\gamma \to \delta)) \to \delta)))$
26. 2 25: $(\alpha \to (\beta \to ((\alpha \to (\gamma \to \delta)) \to \gamma))) \to (\alpha \to (\beta \to ((\alpha \to (\gamma \to \delta)) \to \delta)))$
27. 26 13: $\alpha \to (\beta \to ((\alpha \to (\beta \to \gamma)) \to \gamma))$

The preceding pair of proofs correspond to a bottom-up "parse", or "recognition" of the proof. We can also do a top-down "derivation" of the proof, starting with the goal and working backwards, which is how the abstraction algorithm itself works. Along the way, we will introduce two "optimizations", at steps 13 and 14, which were revealed by inspecting the types at those steps in the preceeding proof.

27. 26 13: (26 13)
26. 2 25: ((2 25) 13)
25. 10 24: ((2 (10 24)) 13)
24. 19 23: ((2 (10 (19 23))) 13)
23. 22 15: ((2 (10 (19 (22 15)))) 13)
22. 2 21: ((2 (10 (19 ((2 21) 15)))) 13)
21. 10 20: ((2 (10 (19 ((2 (10 20)) 15)))) 13)
20. 19 16: ((2 (10 (19 ((2 (10 (19 16))) 15)))) 13)
19. 2 18: ((2 (10 ((2 18) ((2 (10 ((2 18) 16))) 15)))) 13)
18. 10 17: ((2 (10 ((2 (10 17)) ((2 (10 ((2 (10 17)) 16))) 15)))) 13)
17. 7 9: ((2 (10 ((2 (10 (7 9))) ((2 (10 ((2 (10 (7 9))) 16))) 15)))) 13)
16. 7 5: ((2 (10 ((2 (10 (7 9))) ((2 (10 ((2 (10 (7 9))) (7 5)))) 15)))) 13)
15. 12 14: ((2 (10 ((2 (10 (7 9))) ((2 (10 ((2 (10 (7 9))) (7 5)))) (12 14))))) 13)
14. A1: ((2 (10 ((2 (10 (7 9))) ((2 (10 ((2 (10 (7 9))) (7 5)))) (12 K))))) 13)
13. 6: ((2 (10 ((2 (10 (7 9))) ((2 (10 ((2 (10 (7 9))) (7 5)))) (12 K))))) 6)
12. 2 11: ((2 (10 ((2 (10 (7 9))) ((2 (10 ((2 (10 (7 9))) (7 5)))) ((2 11) K))))) 6)
11. 10 8: ((2 (10 ((2 (10 (7 9))) ((2 (10 ((2 (10 (7 9))) (7 5)))) ((2 (10 8)) K))))) 6)
10. 2 9: ((2 ((2 9) ((2 ((2 9) (7 9))) ((2 ((2 9) ((2 ((2 9) (7 9)))
(7 5)))) ((2 ((2 9) 8)) K))))) 6)
9. 1 2: ((2 ((2 (1 2)) ((2 ((2 (1 2)) (7 (1 2)))) ((2 ((2 (1 2))
((2 ((2 (1 2)) (7 (1 2)))) (7 5))))
((2 ((2 (1 2)) 8)) K))))) 6)
8. 7 6: ((2 ((2 (1 2)) ((2 ((2 (1 2)) (7 (1 2)))) ((2 ((2 (1 2))
((2 ((2 (1 2)) (7 (1 2)))) (7 5))))
((2 ((2 (1 2)) (7 6))) K))))) 6)
7. 2 6: ((2 ((2 (1 2)) ((2 ((2 (1 2)) ((2 6) (1 2)))) ((2 ((2 (1 2))
((2 ((2 (1 2)) ((2 6) (1 2)))) ((2 6) 5))))
((2 ((2 (1 2)) ((2 6) 6))) K))))) 6)
6. 1 1: ((2 ((2 (1 2)) ((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 ((2 (1 2))
((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 (1 1)) 5))))
((2 ((2 (1 2)) ((2 (1 1)) (1 1)))) K))))) (1 1))
5. 1 4: ((2 ((2 (1 2)) ((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 ((2 (1 2))
((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 (1 1)) (1 4)))))
((2 ((2 (1 2)) ((2 (1 1)) (1 1)))) K))))) (1 1))
4. 3 1: ((2 ((2 (1 2)) ((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 ((2 (1 2))
((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 (1 1)) (1 (3 1))))))
((2 ((2 (1 2)) ((2 (1 1)) (1 1)))) K))))) (1 1))
3. 2 1: ((2 ((2 (1 2)) ((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 ((2 (1 2))
((2 ((2 (1 2)) ((2 (1 1)) (1 2)))) ((2 (1 1)) (1 ((2 1) 1))))))
((2 ((2 (1 2)) ((2 (1 1)) (1 1)))) K))))) (1 1))
2. S: ((S ((S (1 S)) ((S ((S (1 S)) ((S (1 1)) (1 S)))) ((S ((S (1 S))
((S ((S (1 S)) ((S (1 1)) (1 S)))) ((S (1 1)) (1 ((S 1) 1))))))
((S ((S (1 S)) ((S (1 1)) (1 1)))) K))))) (1 1))
1. K: ((S ((S (K S)) ((S ((S (K S)) ((S (K K)) (K S)))) ((S ((S (K S))
((S ((S (K S)) ((S (K K)) (K S)))) ((S (K K)) (K ((S K) K))))))
((S ((S (K S)) ((S (K K)) (K K)))) K))))) (K K))

The "optimized" combination eliminates 16 symbols out of 61, for a 26% reduction.

Both combinations work equally well for pairing. A similar process leads to proofs

of *p1* and *p2*. Other connectives can, of course, be defined in a similar manner, using well-known techniques from the study of pure λ-calculi.

4. Inside, Out

An interesting question to ask about our definition of $\wedge$ is whether it is "robust" in the sense that it remains a product for any "reasonable" extension of the logic. One "reasonable" extension is Heyting's formulation of intuitionistic propositional logic (IPL). The axioms of IPL are as follows.

*I*1 $\vdash \alpha \to (\alpha \wedge \alpha)$

*I*2 $\vdash (\alpha \wedge \beta) \to (\beta \wedge \alpha)$

*I*3 $\vdash (\alpha \to \beta) \to ((\alpha \wedge \gamma) \to (\beta \wedge \gamma))$

*I*4 $\vdash ((\alpha \to \beta) \wedge (\beta \to \gamma)) \to (\alpha \to \gamma)$

*I*5 $\vdash \alpha \to (\beta \to \alpha)$

*I*6 $\vdash (\alpha \wedge (\alpha \to \beta)) \to \beta$

*I*7 $\vdash \alpha \to (\alpha \vee \beta)$

*I*8 $\vdash (\alpha \vee \beta) \to (\beta \vee \alpha)$

*I*9 $\vdash ((\alpha \to \gamma) \wedge (\beta \to \gamma)) \to ((\alpha \vee \beta) \to \gamma)$

*I*10 $\vdash \neg\alpha \to (\alpha \to \beta)$

*I*11 $\vdash ((\alpha \to \beta) \wedge (\alpha \to \neg\beta)) \to \neg\alpha$

Although the self-distributive law of implication does not appear explicitly in this list, it is easily derived as a theorem. Hence the theorems corresponding to *c*, *p1*, and *p2* can also be derived as before. In order for our definition of $\wedge$ in terms of $\to$ to continue to play the role of conjunction, however, we must show that all of the new axioms involving $\wedge$ (namely I1, I2, I3, I4, I6, I9, and I11) can be proved as theorems from the remaining axioms (namely I5, I7, I8, and I10). This is impossible on the face of it, because of the four remaining axioms only I10 even mentions

the connective $\neg$, and it appears only on the left of $\rightarrow$, whereas we need to derive I11, in which $\neg\alpha$ is the conclusion. In fact, none of the connectives of IL are interdefinable.

This result is somewhat disturbing, because clearly any model of IL is also a model of SK, but $\wedge$ as defined in SK is *not* the product in any such model! How can this be? The definition of $\wedge$ in SK is an *internal* construction. No counterexample to its being a product can be constructed inside SK. From outside, however, it may or may not look like a product. That is, in the system where both SK and its model are constructed, the constructions of the model are available, in addition to those of SK, in which case counterexamples can be constructed if they exist.

From a computational standpoint, only the internal constructions of any system are relevant. This does not mean that any theorem about a computational system is useless to the computer scientist unless its proof corresponds to some construction within the system. For example, it may be useful to know that some construction is or is not "robust" with respect to reasonable extensions. However, we are only concerned with non-robustness if it can be demonstrated in *some* computational system. If this cannot be done, then the purported lack of robustness can never impair the operation of our program. These observations suggest that the metamathematics of computer science should be carried out in some "universal" computational system. For more on this point, see [6].

The correspondence between types, logical formulas, and realizations (i.e., programs) is only hinted at here. It has been part of the folklore of the field for quite some time, but most of the discussions of it in print are buried in technically intimidating works on such things as the semantics of higher-order intuitionistic logic. I hope the current essay has helped to make the essence of this important

insight more accessible, and that it encourages the reader to explore the subject further.

[1] Church, A.: Introduction to Mathematical Logic. Princeton 1956

[2] Curry, H. B., Feys, R.: Combinatory Logic. North-Holland 1968

[3] Hindley, R.: Introduction to Combinatory Logic. Cambridge 1972

[4] Schönfinkel, M.: Uber die Bausteine der mathematischen Logik. Mathematische Annalen **92**, pp. 305-316 1924

[5] Scott, D.: Data Types as Lattices. SIAM J. Comput. **5**, pp. 522-587 Sept. 1976

[6] Shultis, J.: What is a Model? A Consumer's Perspective on Semantic Theory. In: Proc. Conference on Mathematical Foundations of Programming Semantics, Springer-Verlag 1985