CS263—Spring 2008
Topic 1: The Lambda Calculus
Section 1.1: Combinators

Dana S. Scott
Hillman University Professor (Emeritus)
School of Computer Science
Carnegie Mellon University
====
Visiting Professor EECS
Visiting Scientist
Logic & Methodology Program
University of California, Berkeley

Last edited 25 January 2008

Some History

The Problem

Is it possible to have a type-free theory of functions,

where *no difference* is made between *operators* and *arguments* ?

And if so, what *use* is it?

Moses Iljitsch Schönfinkel

Born: 9 September 1886 in Jekaterinoslav, Russia

Died: ~ 1942 in Moscow (?)

Moses Schönfinkel was a student of Hilbert's in Göttingen. In December 1920, he presented a report to the Mathematical Society in Göttingen on a new type of formal logic based on the concept of a generalized function whose argument is also a function (Schönfinkel 1924). This mathematical discipline was subsequently termed **combinatory logic** by H. B. Curry and **λ-conversion** or **λ-calculus** by A. Church. Combinators can be used in the study of algebra, topology, and category theory, and have found application in the study of programs in algorithmic languages.

Reference: Moses Schönfinkel, "*Ueber die Bausteine der mathematischen Logik*", Mathematische Annalen, vol. 92, 1924. An English translation can be found in, "*On the building blocks of mathematical logic*", which appears in "From Frege to Gödel", Jean van Heijenoort (editor), Harvard University Press, Cambridge, 1967.

Haskell Brooks Curry

Born: 12 September 1900 in Millis, Massachusetts, USA

Died: 1 September 1982 in State College, Pennsylvania , USA

Haskell Curry was educated at Harvard and received a doctorate from Göttingen in 1930 for a thesis, supervised by **Hilbert**, entitled *Grundlagen der kombinatorischen Logik.*

He taught at Harvard, Princeton, then for 35 years at Pennsylvania State University. During World War II Curry did research in applied physics at Johns Hopkins University. In 1966, after retirement from Penn State, he accepted a chair of mathematics at Amsterdam for a four-year period.

Curry's main work was in mathematical logic with particular interest in the theory of formal systems and processes. He formulated a logical calculus using inferential rules. His works include *Combinatory Logic* (1958) (with Robert Feys) and *Foundations of Mathematical Logic* (1963).

From an article by: *J J O'Connor* and *E F Robertson*

Curry had started reading Whitehead and Russell (1910-1913) while he was still an undergraduate, and he noticed that in the very first chapter, which deals with propositional logic, there are two rules of inference; the first, modus ponens, which says that from p and $p \supset q$ we may deduce q, is very

simple in its formal structure, but the second rule, which allows the *substitution* of any formula for a propositional variable, is much more complicated. In the mid-1920s, Curry decided to try to break down this rule of substitution into simpler rules. In 1926, he realized that he could do this with essentially the same formalism as Schöfinkel had introduced; Curry called it **combinatory logic**. He did not discover Schöfinkel's paper until November 1927. Curry was later to become so completely identified with combinatory logic that the method of treating functions of more than one argument in terms of functions of one argument, which Frege used and which Curry took from Schöfinkel, has become known as **currying**.

From notes on Lambda-Calculus and Functional Programming: *Jonathan P. Seldin*

Alonzo Church

Born: 14 June 1903 in Washington, D.C., USA

Died: 11 August 1995 in Hudson, Ohio, USA

Alonso Church was a student at Princeton receiving his first degree in 1924, then his doctorate three years later. His doctoral work was supervised by **Veblen**, and he was awarded his doctorate for his dissertation entitled *Alternatives to Zermelo's Assumption.*

Church spent a year at Harvard University then half a year at Göttingen and half a year at Amsterdam where he worked with **Brouwer**. He returned to the USA becoming professor of mathematics at Princeton in 1929, a post he held until 1967 when he became professor of mathematics and philosophy at UCLA.

His work is of major importance in mathematical logic, recursion theory and in theoretical computer science. He created the λ-calculus in the 1930s which today is an invaluable tool for computer scientists. He is best remembered for Church's Theorem (1936), which says that there is no decision procedure for the full predicate calculus. It appears in *An unsolvable problem in elementary number theory* published in the American Journal of Mathematics. His work extended that of **Gödel**.

Church was one of the founders of the *Journal of Symbolic Logic* in 1936 and remained an editor of the section on reviews until 1979. He wrote the book *Introduction to Mathematical Logic* in 1956. He had 31 doctoral students including **Turing**, **Kleene**, **Rosser**, **Kemeny, Rabin, Scott** and **Smullyan**.

Adapted from an article by: *J J O'Connor* and *E F Robertson*

■ Church's Ph.D. students

1931 Alfred L. Foster

1934 J. Barkley Rosser

1934 Stephen C. Kleene

1938 Alan M. Turing

1944 Enrique Bustamente-Llaca

1947 Leon Henkin

1949 John G. Kemeny

1950 Martin D. Davis

1951 Maurice L'Abb'e

1952 Hartley Rogers, Jr.

1952 William W. Boone

1955 Norman Shapiro

1956 T. Thacher Robinson

1957 Michael O. Rabin

1958 Dana Scott

1959 Aubert Daigneault

1959 Raymond Smullyan

1959 Simon Kochen

1960 Robert W. Ritchie

1961 James R. Guard

1962 James Bennett

1962 Robert O. Winder

1963 Gustav B. Hensel

1963 Wayne H. Richter

1964 Peter Andrews

1964 William B. Easton

1965 Joel W. Robbin

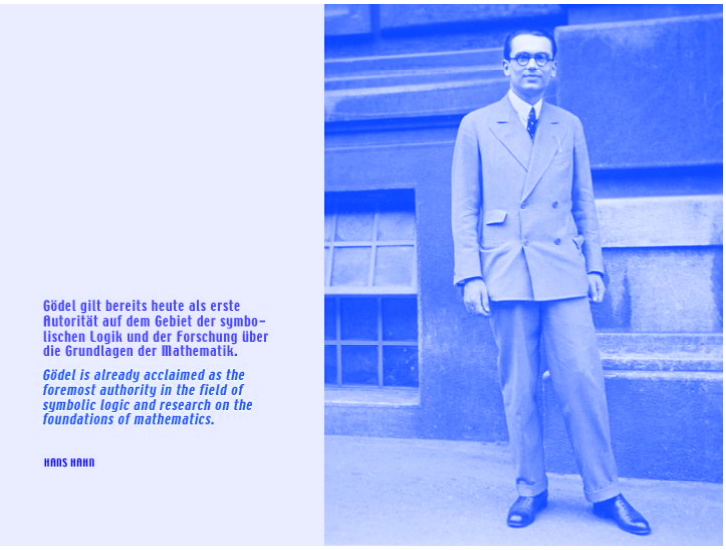
1967 Donald J. Collins

1976 Richard J. (Isaac) Malitz

1977 C. Anthony Anderson

1985 Gary R. Mar

Kurt Gödel



http://www.goedelexhibition.at/gallery/gdelgilt.jpg

Born: 28 April 1906 in Brünn, Austria-Hungary

Died: 14 January 1978 Princeton, New Jersey, USA

Time Magazine ranked him among the hundred most important persons of the twentieth century. Harvard University made him a honorary doctor "for the discovery of the most significant mathematical truth of the century." He is generally viewed as the greatest logician since Aristotle. His friend Einstein liked to say that he only went to the institute to have the privilege of walking back home with Kurt Gödel. And John von Neumann, one of the fathers of the computer, wrote: "Indeed Gödel is absolutely irreplaceable. He is the only mathematician about whom I dare make this assertion."

Kurt Gödel was born in Brno, and studied in Vienna during the `twenties. At the age of twenty-four, he revolutionized not only mathematics, but also the way we see it. During the `thirties, he commuted between Vienna (where he earned, as a private lec-

turer, 2.90 shillings per semester) and Princeton (he was one of the first guest members of the newly founded Institute for Advanced Study). In early 1940, although not racially persecuted by the Nazi regime, he emigrated to the USA via the Soviet Union and Japan. He never returned to Vienna, and he wrote to his mother that he was plagued by nightmares about being trapped in Vienna again.

But Gödel had spent his best and most productive years in Vienna. He belongs to the Vienna between the wars, just like Sigmund Freud, Ludwig Wittgenstein, Karl Popper, Konrad Lorenz, Robert Musil or Arnold Schönberg, and he may well one day be considered as the best-known representative of this uniquely rich "golden autumn". Vienna honored him in 2006, belatedly but gratefully, by a large congress held at the University of Vienna, sponsored by the Templeton Foundation, and an exhibition under the patronage of the Austrian president, financed by the state and the city.

Gödel proved that every mathematical theory rich enough to allow for counting, adding and multiplying contains true statements which cannot be proved, except if the theory harbours a contradiction. Worse yet, if one can prove that the theory is consistent, then it is not. As Hans Magnus Enzensberger wrote in his Hommage à Gödel: "You can describe your own language in your own language: but not completely". This statement seems reasonable enough, but Gödel managed to translate it into a mathematical proposi tion. He succeeded in turning a philosophical sentence into a mathematical theorem. In this sense, what Gödel has done for philosophy is similar to what Newton has done for physics.

The two other big discoveries of Gödel are of a similar breath-taking temerity. He made a fundamental contribution to set theory, the science of infinity, a field that has been called "the theology of mathematicians". Gödel succeeded in solving one half of the so-called continuum hypothesis, the number one on the hit-list of mathematical problems of his century. And he proved that Einstein's theory of relativity permits, on principle, to travel into one's own past, something that cosmologists, to this day, have not properly digested. Gödel remarks, in an aside, that the direction of time, after landing in one's own past, is the same as before. Hence time does not run in the opposite sense, like a film spooling backward.

Gödel, who analysed proofs for the existence of God, who believed in metempsychosis and who wanted to uncover a conspiracy against Leibniz seems hardly to fit into the twentieth century. But he spent his life right in the center of the avant garde of its time. Both the thinkers of the Vienna Circle and the scientists in Princeton belonged to the most modern minds the twentieth century had to offer. Thus for instance, the development of the computer by Alan Turing and John von Neumann is based on mathematical

logic and formal systems, fields whose undisputed champion was Gödel, in those years. The incompleteness theorem, discovered by Gödel well before the time of programmable computers, is a theorem on the limitations of computer programs, and since the success of "Gödel-Escher-Bach", Gödel ranks as an icon of the computer age.

From an introduction by Prof. Karl Sigmund

Alan Mathison Turing

Born: 23 June 1912 in London, England

Died: 7 June 1954 in Wilmslow, Cheshire, England

Alan Mathison Turing, OBE, FRS was an English mathematician, logician, and cryptographer. Turing is often considered to be the father of modern computer science. Turing provided an influential formalization of the concept of the algorithm and computation with the Turing machine, formulating the now widely accepted "Turing" version of the Church–Turing thesis, namely that any practical computing model has either the equivalent or a subset of the capabilities of a Turing machine.

With the Turing test, he made a significant and characteristically provocative contribution to the debate regarding artificial intelligence: whether it will ever be possible to say that a machine is conscious and can think.

He later worked at the National Physical Laboratory, creating one of the first designs for a stored-program computer, although it was never actually built. In 1948 he moved

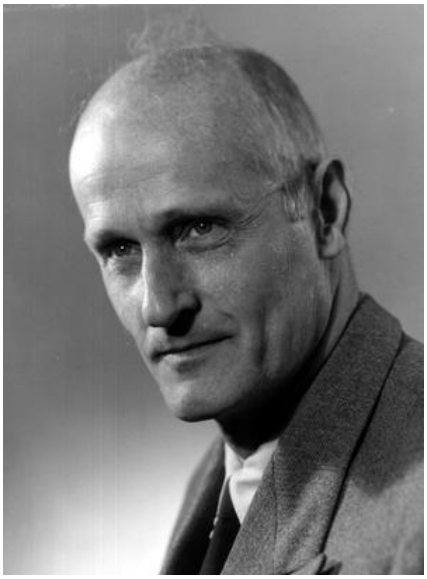
to the University of Manchester to work on the Manchester Mark I, then emerging as one of the world's earliest true computers.

During the Second World War Turing worked at Bletchley Park, Britain's codebreaking centre, and was for a time head of Hut 8, the section responsible for German naval cryptanalysis. He devised a number of techniques for breaking German ciphers, including the method of the bomb, an electromechanical machine that could find settings for the Enigma machine.

In 1954 Turing died after eating an apple laced with cyanide. His death was ruled a suicide.

Adapted from an article in the Wikipedia.

Stephen Cole Kleene



Born: 5 January 1909 in Hartford, Connecticut, USA

Died: 25 January 1994 in Madison, Wisconsin, USA

Stephen C Kleene studied for his first degree at Amherst College. He went on to receive a doctorate from Princeton University in 1934, supervised by Church, for a thesis entitled *A Theory of Positive Integers in Formal Logic*. He taught at Princeton until he joined the University of Wisconsin at Madison in 1935. He became a full professor at the University of Wisconsin at Madison in 1948 and remained on the staff there until he retired in 1979.

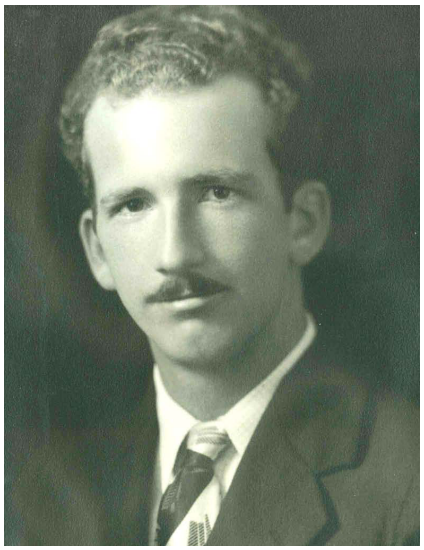
Kleene's research was on the theory of algorithms and recursive functions. He developed the field of recursion theory with Church, Gödel, Turing and others. He contributed to mathematical Intuitionism which had been founded by Brouwer. His work on recursion theory helped to provide the foundations of theoretical computer science. By providing methods of determining which problems are soluble, Kleene's work led to the study of which functions can be computed.

From 1930's on Kleene more than any other mathematician developed the notions of computability and effective process in all their forms both abstract and concrete, both mathematical and philosophical. He tended to lay the foundations for an area and then move on to the next, as each successive one blossomed into a major research area in his wake.

Kleene's best known books are Introduction to Metamathematics (1952) and Mathematical Logic (1967).

From an article by J J O'Connor and E F Robertson.

J. Barkley Rosser



First picture: 1930 (imagine bright red hair) Second picture: about 1965

Born: 6 December 1907 in Jacksonville, Florida, USA

Died: 5 September 1989 in Madison, Wisconsin, USA

J. Barkley Rosser earned both his Bachelor of Science (1929) and his Master of Science (1931) from the University of Florida. He obtained his Ph.D. from Princeton University in 1934. After getting his Ph.D., Rosser taught at Princeton, Harvard, and Cornell and spent the latter part of his career at the University of Wisconsin-Madison. He continued to lecture well into his late 70s.

Rosser contributed to many committees and professional associations in addition to his teaching: he served as president of the Association for Symbolic Logic and the Society of Industrial and Applied Mathematics; was a member of the space vehicle panel for the advisory committee of the Apollo project; was an early contributor to computer science theory; and helped develop the Polaris missile. While at the University of Wisconsin-Madison, he served as the director of the U. S. Army Mathematics Research Center. Rosser is known for his part in the Church-Rosser theorem, in lambda calculus, and he developed the Rosser sieve. His areas of expertise include symbolic logic, ballistics, rocket development, and numerical analysis.

Rosser began work on his own version of combinatory logic, which led to his dissertation (Rosser 1935). In 1934, he and Kleene discovered something important about both the system of logic of Church and the combinatory logic of Curry: both systems are inconsistent in that they admit Richard's paradox (Kleene and Rosser 1935). Curry then began a thorough study of their paradox in detail; this led first to a new exposition of the paradox (Curry 1941 "Paradox") and then to a new, simpler contradiction (Curry 1942 "Inconsistency"), which has since come to be known as **Curry's paradox** .

Partly from notes on "Lambda-Calculus and Functional Programming" by Jonathan P. Seldin

Some Computations

The Basic Rules

■ The combinators J, S and K

To begin with, we single out three ***combinators***, from which we will generate all others.

Initially, by a ***combination*** we understand either

a **J**, an **S** or a **K**, forming the basic ***constants***, or

a ***letter*** set aside to be a ***variable***, or

a ***compound expression*** of the form **A[B]**, where

A and **B** are previously obtained combinations,

A combination ***without variables*** is also called a ***combinator***. Intuitively, a combinator is some kind of ***function*** **F** which when applied to arguments, as in $\mathbf{F[x_1][x_2][x_3]..[x_n]}$, affects a ***transformation***. To give some kind of exact ***"meaning"*** to the combinators we use ***replacement rules***.

```
crules = {J[x_] → x, S[x_][y_][z_] → x[z][y[z]], K[x_][y_] → x};
```

We have to do some ***examples***, however, to see what these rules ***accomplish*** in giving meaning to all combinations.

Note: As an aid to memory, we might ***nickname*** the basic combinators as follows:

J is the ***Joker***;

S is the ***Slider***; and

K is the ***Killer***.

Warning: The combinitor **J** is usually written as **I**.

But *Mathematica* has a special role for **I** which does not concern the current discussion.

■ The identity combinator

The ***identity combinator*** **J** can be defined — ***in effect*** — by the others.

In general many combinations are ***equivalent*** to one another, and we shall have to sort out how this comes about. But for some technical reasons for giving proofs about our rules it is easier to keep **J** as a ***primitive***.

```
comb = J;
test = comb[A]
crules
```
```
J[A]
```
```
{J[x_] → x, S[x_][y_][z_] → x[z][y[z]], K[x_][y_] → x}
```
```
test = test /. crules
```
```
A
```
```
comb = S[K][K];
test = comb[A]
```
```
S[K][K][A]
```
```
test = test //. crules
```
```
A
```

In other words, **J** and **S[K][K]** are ***equivalent*** in action when applied to an argument. What about **S[K][S]** ?

■ The composition combinator

Here is a bigger combination to try out.

```
comb = S[K[S]][K];
test = comb[x][y]
```
```
S[K[S]][K][x][y]
```
```
test = test /. crules
```
```
K[S][x][K[x]][y]
```
```
test = S[K[x]][y][z]
```
```
S[K[x]][y][z]
```
```
test = test /. crules
```
```
K[x][z][y[z]]
```
```
test = test /. crules
```
```
x[y[z]]
```

Thus, if we think of two "functions" **F** and **G**, then **S[K[F]][G]** represents their ***composition***.

■ The next question

But, how do we *find* other useful combinations?

Eliminating Variables

■ Abstraction

Given a ***list of variables*** and a ***combination***, we create a combinator by ***removing variables one at a time***, starting with the right-most variable.

```
ToC[vars_, comb_] := Fold[rm, comb, Reverse[vars]];

rm[v_, v_] := J;
rm[f_[v_], v_] /; FreeQ[f, v] := f;
rm[h_, v_] /; FreeQ[h, v] := K[h];
rm[f_[g_], v_] := S[rm[f, v]][rm[g, v]];
```

Warning: In *Mathematica*, **FreeQ** means "to be free of". Do not confuse this with "free and bound variables".

Note: In ***traditional*** notation **ToC[{x, y, z}, A]** is written as **λ x λ y λ z.A**.

■ Example 1: The identity and diagonal combinators

In these tests we only have to remove ***one*** variable.

```
rm[x, x]
J
```

```
rm[x[x], x]
S[J][J]
```

```
test = S[J][J][x]
S[J][J][x]
```

```
test = test /. crules
J[x][J[x]]
```

```
test = test /. crules
x[x]
```

Note: In other words, if you calculate **B = λ x.A** ***to remove*** **x**, then

you should (pretty much) expect to get **A** ***back again*** by simplifying

B[x] by several applications of the **crules**.

We will have to ***prove*** this in general later.

Example 2: Reordering variables

Here is the ***composition*** combinator we looked at earlier.

```
comb = ToC[{x, y, z}, x[y[z]]]
S[K[S]][K]
```

If the variables are given in a ***different order***, there is a different ***result***.

```
comb = ToC[{y, x, z}, x[y[z]]]
S[K[S[S[K[S]][K]]]][K]
```

```
test = comb[x][y][z]
S[K[S[S[K[S]][K]]]][K][x][y][z]
```

```
test = test /. crules
K[S[S[K[S]][K]]][x][K[x]][y][z]
```

```
test = test /. crules
S[F][K[x]][y][z]
```

```
test = test /. crules
F[y][K[x][y]][z]
```

```
test = test /. crules
F[y][x][z]
```

In general, `ToC[{y, x, z}, F[x][y][z]]` reorders the arguments of **F**.

```
ToC[{y, x, z}, F[x][y][z]]
S[K[S[F]]][K]
```

```
test = S[K[S[F]]][K][x][y][z]
S[K[S[F]]][K][x][y][z]
```

```
test = test /. crules
K[S[F]][x][K[x]][y][z]
```

```
test = test /. crules
F[y][x][z]
```

■ Example 3: Longer compositions

```
comb = ToC[{x, y, z, w}, x[y[z[w]]]]
S[S[K[S]][S[K[K]][S[K[S]][S[K[K]][S[K[S]][K]]]]]][
 K[S[K[S]][K]]]
```

```
test = comb[x][y][z]
S[S[K[S]][S[K[K]][S[K[S]][S[K[K]][S[K[S]][K]]]]]][
    K[S[K[S]][K]]][x][y][z]
```

```
test = test //. crules
S[K[x]][S[K[y]][z]]
```

```
test = S[K[x]][S[K[y]][z]][w]
S[K[x]][S[K[y]][z]][w]
```

```
test = test //. crules
x[y[z[w]]]
```

■ **Example 4: Reverse application**

```
comb = ToC[{x, y}, y[x]]
S[K[S[J]]][K]
```

```
test = comb[x][y]
S[K[S[J]]][K][x][y]
```

```
test = test //. crules
y[x]
```

■ **Example 5: Using self-application**

```
comb = ToC[{x}, F[x[x]]]
S[K[F]][S[J][J]]
```

```
test = comb[x]
S[K[F]][S[J][J]][x]
```

```
test = test //. crules
F[x[x]]
```

That combination was "well behaved" when simplified ***all by itself***.

But if we put it in combination ***with itself*** . . . .

```
test = comb[comb]
S[K[F]][S[J][J]][S[K[F]][S[J][J]]]
```

```
test = test /. crules
K[F][S[K[F]][S[J][J]]][S[J][J][S[K[F]][S[J][J]]]]
```

```
test = test /. crules
F[J[S[K[F]][S[J][J]]][J[S[K[F]][S[J][J]]]]]
```

```
test = test /. crules
F[S[K[F]][S[J][J]][S[K[F]][S[J][J]]]]
```

In other words, we have "reduced" to **F[comb[comb]]**. So actually there has been ***an expansion and a self-replication***.

A comment and a warning!

This calculation shows that

every function has a fixed point!

This means that given **F**, we can find a **P** such that **P → F[P]** by the **crules**.

And, moreover, we see that

the reduction of a combinator need not stop!

The problem here is trying to know when reductions ***will*** stop.

This also shows that the notion of *function*

embodied in combinators is *not* the

same as is familiar in mathematical usage.

Example 6: Combinations of Ks

Theorem. If a combination consists entirely of **K**s, then by the **crules** it reduces to an irreducible combination of the form: **K[K[K[... K[K[K]] ...]]]**.

Proof. Note that we include among the irreducible forms a single **K**.

Now, if a combination is not of the form of the theorem, it must contain a subexpression of the form $\mathbf{K[A_1][A_2][\ ...][A_n]}$, where $\mathbf{n \geq 2}$. This reduces at once to the form $\mathbf{A_1[\ ...][A_n]}$, which is shorter, as $\mathbf{A_2}$ has been eliminated.

Indeed, every application of the **K**-rule produces a shorter expresion.

Thus, the reduction sequence is always finite. **Q.E.D.**

```
test = K[K[K[K]]][x][y][z]
K[K[K[K]]][x][y][z]
```

```
test = test //. crules
K
```

More generally:

```
test = K[K[K[a]]][x][y][z]
K[K[K[a]]][x][y][z]
```

```
test = test //. crules
a
```

Or, the other way 'round:

```
ToC[{x, y, z}, a]
K[K[K[a]]]
```

■ Example 7: Combinations of Ss

Mathematica allows us to make ***recursive definitions***, so that a sequence of combinators can be investigated.

```
SS[0] := S;
SS[n_] := SS[n - 1][S];
```

```
test = SS[13]
S[S][S][S][S][S][S][S][S][S][S][S][S][S]
```

```
test = test //. crules
S[S][S[S][S[S][S[S][S[S][S[S][S[S]]]]]]]
```

Such combinators seem to ***stabilize*** (also called ***normalize***). Let's try a longer one.

```
SS[50] //. crules
S[S][S[S][S[S][
    S[S][S[S][S[S][S[S][S[S][S[S][S[S][S[S][S[S][S[S][S[S][
                  S[S][S[S][S[S][S[S][S[S][S[S][S[S][S[S][S[S][
                              S[S][S[S][S]]]]]]]]]]]]]]]]]]]]]]]]]
```

Can this be proved ***in general*** ?

Now try another kind of combination.

```
test = SS[2][SS[2]]
S[S][S][S[S][S]]
```

```
test = test //. crules
S[S[S][S]][S[S[S][S]]]
```

That was easy. Next?

```
test = SS[3][SS[3]]
S[S][S][S][S[S][S][S]]
```

```
test = test //. crules
S[S[S][S[S]]][S[S][S[S][S[S]]]]
```

Next?

```
test = SS[4][SS[4]]
S[S][S][S][S][S[S][S][S][S]]
```

```
test = test //. crules
S[S[S][S[S][S]]][S[S[S][S[S][S]]][S[S[S][S[S][S]]]]]
```

```
test = SS[5][SS[5]]
S[S][S][S][S][S][S[S][S][S][S][S]]
```

```
test = test //. crules
S[S[S][S[S][S[S]]]][
 S[S[S][S[S][S[S]]]][S[S][S[S][S[S][S[S]]]]]]
```

Are all these **SS[n][SS[n]]** safe?   Try another.

```
SS[10][SS[10]] //. crules
S[S[S][S[S][S[S][S[S][S[S][S]]]]]][
 S[S[S][S[S][S[S][S[S][S[S][S]]]]]][
  S[S[S][S[S][S[S][S[S][S[S][S]]]]]][
   S[S[S][S[S][S[S][S[S][S[S][S]]]]]][
    S[S[S][S[S][S[S][S[S][S[S][S]]]]]][
     S[S[S][S[S][S[S][S[S][S[S][S]]]]]]]]]]]
```

OK. What about this?

```
test = SS[2][SS[2]][SS[2]]
S[S][S][S[S][S]][S[S][S]]
```

```
Do[test = test /. crules, {15}];
test
```

```
S[S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]][
            S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]][
          S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]][
            S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]][
        S[S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]][
            S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]][
          S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]][
            S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]]][
      S[S][S][S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]][
```

```
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
        S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]][
      S[S][S][S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]][
        S[S][S][S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]][
    S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
     S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
        S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
       S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
        S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
      S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
       S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
        S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]]][
    S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
       S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]][
     S[S[S][S]][S[S[S][S]][S[S][S]]]][
        S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
        S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]]][
      S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
         S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]][
       S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
        S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]]][
  S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
     S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]][
    S[S[S][S]][S[S[S][S]][S[S][S]]]][
       S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
        S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]]][
      S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
         S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]][
       S[S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
        S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
          S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]]]]][
 S[S[S[S][S]][S[S[S][S]][S[S][S]]]]][
  S[
    S[S[S][S]][S[S[S][S]][S[S][S]]]]][
   S[S[S[S][S]][S[S[S][S]][S[S][S]]]]]]][
S[S[S[S][S]][S[S][S]]][S[
   S[S][S]][
  S[S[S][S]][
   S[S][S]]]]]
```

Not so good! This combinator probably ***does not*** normalize. Proof?

■ **Example 8: Even more Ss**

```
test = S[A][A][S[A][A]] /. A → S[S]
```

```
S[S[S]][S[S]][S[S[S]][S[S]]]
```

Safe or not?

```
Do[test = test /. crules, {30}];
test
```

```
S[S[S][S[S[S]][S[S]]]][
 S[S[S][S[S][S[S[S]][S[S]]]]][S[S[S][S[S][S[S[S]][S[S]]]]][
   S[S[S][S[S][S[S][S[S[S]][S[S]]]]]][
    S[S[S][S[S][S[S][S[S[S]][S[S]]]]]][
     S[S[S][S[S][S[S][S[S[S]][S[S]]]]]][
      S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
       S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
        S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
         S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
          S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
           S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
            S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
             S[S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
              S[S[S][S[S][S[S][S[S][S[
                     S[S][S[S[S]][S[S]]]][S[S[S][S[S][
                  S[S][S[S][S[S][S[S][S[S]][S[S]]]]]][
                S[S[S][S[S][S[S][S[S][S[S][S[S]][
                        S[S]]]]]]]][S[S[S][S[S][S[S][
                     S[S][S[S][S[S][S[S]][S[S]]]]]]][
                   S[S[S][S[S][S[S][S[S][S[S][S[S]][
                           S[S]]]]]]]][S[S[S][S[S][S[
                      S[S][S[S][S[S][S[S]][S[S]]]]]]]][
                    S[S[S][S[S][S[S][S[S][S[S][S[
                            S[S][S[S[S]][S[S]]]]]]]]][
                      S[S[S][S[S][S[S][S[S][S[S][
                              S[S][S[S[S]][S[S]]]]]]]]][
                       S[S[S][S[S][S[S][S[S][S[
                               S[S][S[S[S]][S[S]]]]]]]]][
                         S[S[S][S[S][S[S][S[S][
                                 S[S][S[S[S]][S[S]]]]]]]]][
                          S[S[S][S[S][S[S][S[S][S[
                                  S[S][S[S[S]][S[S]]]]]]]]][
                            S[S[S][S[S][S[S][S[S][
                                    S[S][S[S[S]][S[S]]]]]]]]][
                              S[S[S][S[S][S[S][S[S][S[
                                      S[S][S[S[S]][S[S]]]]]]]]][
                                S[S[S][S[S][S[S][S[S][
                                        S][S[S][S[S[S]][S[S]]]]]]]]][
```

```
S[S[S] [S[S] [S[S] [S[S] [S[S] [S[
  S] [S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] [
 S[S[S] [S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] [S[S[
  S] [S[S] [S[S] [S[S] [S[S] [S[S] [S[
  S] [S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] [
 S[S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S[S] ] [S[
  S] ] ] ] ] ] ] ] ] ] ] [S[S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] [S[S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [S[
  S] [S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] [
  S[S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S[S] ] [S[
  S] ] ] ] ] ] ] ] ] ] ] [S[S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] [S[S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [S[
  S] [S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] [
  S[S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S[S] ] [S[
  S] ] ] ] ] ] ] ] ] ] ] [S[S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] ] [S[
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S[S] ] [S[
  S] ] ] ] ] ] ] ] ] ] ] ] [S[S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] ] [S[
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S[S] ] [S[
  S] ] ] ] ] ] ] ] ] ] ] ] [S[S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] ] [S[
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S[S] ] [S[
  S] ] ] ] ] ] ] ] ] ] ] ] [S[S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S[S] ] [S[S] ] ] ] ] ] ] ] ] ] ] ] [S[
  S[S] [S[S] [S[S] [S[S] [S[S] [
  S[S] [S[S] [S[S] [S[S] [S[S[S] ] [S[
  S] ] ] ] ] ] ] ] ] ] ] ] [S[S[S] [S[S] [S[S] [
```

```
                    S[S] [S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S[S]] [S[S]]]]]]]]]]]] [
                    S[S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S[S]] [S[S]]]]]]]]]]]]] [
                    S[S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S[S]] [S[S]]]]]]]]]]]]] [
                    S[S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S[S]] [S[S]]]]]]]]]]]]] [
                    S[S[S] [S[S] [S[S] [S[S] [S[
                    S] [S[S] [S[S] [S[S] [S[S] [S[S] [S[
                    S[S]] [S[S]]]]]]]]]]]]]] [S[S[S] [
                    S[S] [S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S] [S[S] [S[S] [S[S[S]] [S[
                    S]]]]]]]]]]]]] [S[S] [S[S] [S[S] [
                    S[S] [S[S] [S[S[S]] [S[S]]]]]]]] [
                    S[S] [S[S] [S[S] [S[S] [S[S] [S[S] [
                    S[S] [S[S] [S[S] [S[S] [S[S[S]] [S[
                    S]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
                    ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
  ]]
```

Another bad one. ***So, some are good and some are bad.***

■ A known theorem

It has actually been ***proved*** in the paper:

Johannes Waldmann, *"The Combinator S"*,

Information and Computation, vol. 159 (2000), pp. 2-21.

that it is possible to ***decide*** by an algorithm whether an **S**-expression ***normalizes*** or ***not***.

It is not so much ***fun*** to do this, however!