Kissy Kiosk

by Zefram, London MathsJam, March 2023

Definitions

A *kiss* is a finite sequence of symbols, of the form "`S`", "`K`", or "`(`xy`)`" where x and y are themselves kisses. Two kisses are equal (written "=") if and only if they are identical symbol sequences.

Most kisses have a *suksessor* which is also a kiss. Where the suksessor of x is y, we write that "$x \overset{1}{\mapsto} y$". Suksessors are determined by these rules:

form of kiss	**suksessor**
`(((S`x`)`y`)`z`)`	`((`xz`)(`yz`))`
`((K`x`)`y`)`	x
`(`xy`)` where $x \overset{1}{\mapsto} x'$	`(`$x'y$`)`
otherwise	no suksessor

We are interested in iterating the suksessor operation, that is, finding the suksessor of the suksessor, and so on. A kiss x *sukkumbs to* a kiss y (written "$x \mapsto y$") if and only if a finite number of suksessor operations (zero or more) applied to x yields y. For example, `(((S(SS))K)K)` $\mapsto$ `((S(KK))(K(KK)))`.

Some kisses represent truth values. A kiss t is *kikky* (representing truth) if and only if, for all kisses x and y, `((`tx`)`y`)` $\mapsto x$, and a kiss f is *sukky* (representing falsity) if and only if, for all kisses x and y, `((`fx`)`y`)` $\mapsto y$. A kiss is *sassy* if and only if it is either kikky or sukky.

Problems

Each of these problems has infinitely many solutions.

Problem 1: find a kiss I such that, for all kisses x, `(`Ix`)` $\mapsto x$.

Problem 2: find a kikky kiss T and a sukky kiss F.

Problem 3: find a kiss N performing the logical "not" operation, such that, for all sassy kisses x, `(`Nx`)` is sukky if x is kikky, and is kikky otherwise.

Problem 4: find a kiss A performing the logical "and" operation, such that, for all sassy kisses x and y, `((`Ax`)`y`)` is kikky if x and y are both kikky, and is sukky otherwise. Do the equivalent for a kiss O performing logical "or" and a kiss X performing logical "xor".

Problem 5: find a kiss U that does not sukkumb to any kiss that has no suksessor, i.e., its chain of suksessors has no end.

Solutions and discussion overleaf

Solutions

Problem 1: one solution is $I = \texttt{((SK)K)}$. Observe that, for any kiss x,
$\texttt{(}Ix\texttt{)} = \texttt{(((SK)K)}x\texttt{)} \overset{1}{\mapsto} \texttt{((K}x\texttt{)(K}x\texttt{))} \overset{1}{\mapsto} x$

Problem 2: one pair of sassy kisses is $T = \texttt{K}$ and $F = \texttt{(SK)}$. The correctness of this value for T follows directly from the suksessor rule concerning K. For F, observe that, for any kisses x and y,
$\texttt{((}Fx\texttt{)}y\texttt{)} = \texttt{(((SK)}x\texttt{)}y\texttt{)} \overset{1}{\mapsto} \texttt{((K}y\texttt{)(}xy\texttt{))} \overset{1}{\mapsto} y$

Problem 3: one solution is $N = \texttt{((S(S(K((S}I\texttt{)(K}T\texttt{)))))(K}F\texttt{))}$. The way this works is that, for any kiss x,

$$\texttt{(}Nx\texttt{)} \mapsto \texttt{((}xf\texttt{)}t\texttt{)},$$

where $f = \texttt{((K}F\texttt{)}x\texttt{)} \overset{1}{\mapsto} F$ is sukky
and $t = \texttt{((K}T\texttt{)(}xf\texttt{))} \overset{1}{\mapsto} T$ is kikky.

Problem 4: The method used in the solution above to problem 3 can be used to make $\texttt{(}Ax\texttt{)}$ sukkumb to one of two kisses that then combine usefully with y. The same approach works for O and X.

$A = \texttt{((S(S(K((S}I\texttt{)(K(K}F\texttt{))))))(K}I\texttt{))}$
$O = \texttt{((S(S(K((S}I\texttt{)(K}I\texttt{)))))(K(K}T\texttt{)))}$
$X = \texttt{((S(S(K((S}I\texttt{)(K}I\texttt{)))))(K}N\texttt{))}$

Problem 5: one solution is $U = \texttt{(((S}I\texttt{)}I\texttt{)((S}I\texttt{)}I\texttt{))}$.

Discussion

The system of kisses is a version of *SK calculus*, which is a surprising way of formulating the calculus of functions. A kiss is actually an expression that describes a function by means of applying functions to functions: $\texttt{(}xy\texttt{)}$ represents the application of the function x to the argument y.

Problems 2 to 4 demonstrate that this system is capable of arbitrary computation on truth values, and problem 5 demonstrates that evaluation of an expression isn't limited to any bounded number of computational steps. These features hint at the important result that the SK calculus is a Turing-complete system of computation. The surprising aspect of the SK calculus is that the primitives S and K and function application are sufficient to describe any possible function in this domain of pure unary functions.

To learn more, start at the Wikipedia article on SK calculus. For the more usual formulation of the calculus of functions see *lambda calculus*. The representation of truth values used in problems 2 to 4 is known as *Church Booleans*, and there are also *Church numerals* which represent natural numbers. If you'd like to actually execute programs written in (a different version of) SK calculus, see the esoteric programming language *Unlambda*.