Under consideration for publication in J. Functional Programming

A Turing Machine For SKI Combinators

Greg Michaelson
School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, EH14 4AS, Scotland
andJoe Davidson
School of Computing Science, University of Glasgow, Glasgow, G12 8RZ, Scotland
(*e-mail:* joseph.davidson8@gmail.com)

Abstract

A Turing complete system is thought to fully capture the notion of algorithm. A demonstration of Turing completeness involves establishing that the expressive power of a candidate system is equivalent to that of a system known to be Turing complete, classically Turing machines or λ calculus.

It has been long established that Curry's combinatory logic is Turing complete, demonstrated through the mutual equivalence of combinators and λ calculus. Here, we discuss a demonstration through the construction of a Turing machine to directly reduce combinatory expressions. We also assess the computational characteristics of this TM against those for other models of computability, namely Random Access Stored Programs (RASP), and two Universal Turing Machines.

1 Overview

A Turing complete system is thought to fully capture the notion of algorithm. A demonstration of Turing completeness involves establishing that the expressive power of a candidate system is equivalent to that of a system known to be Turing complete, classically Turing machines or λ calculus.

It has been long established that Curry's combinatory logic is Turing complete, demonstrated through the mutual equivalence of combinators and λ calculus. Here, we discuss a demonstration through the construction of a Turing machine to directly reduce combinatory expressions. We also assess the computational characteristics of this TM against those for other models of computability, namely Random Access Stored Programs (RASP), and Minsky's and Neary's Universal Turing Machines.

2 Introduction

In 1921, Hilbert elaborated his foundational programme for mathematical logic (Hilbert, 1998) which sought to establish whether there was a:

- complete and consistent formalisation of number theoretic predicate calculus (NTPC);
- terminating algorithm to determine if an arbitrary NTPC formula was a theorem.

$$
\begin{array}{lll}
e & \rightarrow id \mid & \text{(variable)} \\
 & \lambda\ id.e \mid & \text{(function)} \\
 & e_1\ e_2 \mid & \text{(application)} \\
 & (\ e\ ) & \text{(brackets)}
\end{array}
$$

Fig. 1. λ calculus abstract syntax.

The decidability of this latter *entschiedungsproblem* became an important locus of 1930s research after Gödels 1931 incompleteness proofs (Gödel, 1962) established that consistency and completeness for NTPC were mutually contradictory.

Two leading approaches were Church's λ calculus (Church, 1936) and Turing's machines (TMs) (Turing, 1937b), which we will explore in more detail below. In 1936, both Turing and Church used their formalisms to demonstrate independently of each other that the *entschiedungsproblem* is undecidable. However, both had very different notions of "algorithm". For Turing, an algorithm was something that was *computable* by a machine, whereas for Church an algorithm was something that was *effectively calculable* by normalisation.

Both subsequently agreed that their notions were equivalent, leading to what is now termed the *Church-Turing thesis* (CTT), that all characterisations of algorithm are equivalent. The CTT has been repeatedly challenged by proponents of super-Turing- or hypercomputation (Cockshott & Michaelson, 2007), who claim that there are models of computing which are more powerful than Turing machines. Nonetheless, all extant models which do not require trans-finite constructions have been shown to satisfy the CTT. We will consider this further once we have briefly explored λ calculus and TMs, and introduced combinatory logic.

2.1 λ calculus

Figure 1 shows an abstract syntax for λ calculus.

A *redex*, consisting of an application whose first expression is a function, may be *reduced* to *normal form* by β reduction:

$$(\lambda id.e_1)\ \ e_2\ \ \rightarrow_\beta\ \ e_1[id/e_2]$$

That is, all free occurrences of the bound variable *id* in the body e_1 are replaced by the argument e_2.

2.2 Turing machines

Turing machines(TMs) are based on a bounded but extensible *tape* of *cells* each of which may hold a single *symbol*. There is a read/write *head* positioned over the *current* cell. The head may move left or right over the tape. When either end of the tape is reached, a new empty cell is appended. See Figure 2.

A TM is controlled by a set of transition rules conventionally written as a quintuplet in the form:

$$(state, symbol, state', symbol', direction)$$

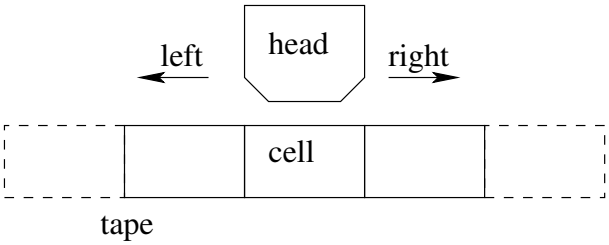

Fig. 2. Turing machine.

$\texttt{I}\ x \rightarrow_I x$	(identity)
$\texttt{K}\ x\ y \rightarrow_K x$	(constant)
$\texttt{S}\ x\ y\ z \rightarrow_S x\ z\ (y\ z)$	(application)

Fig. 3. Basic combinators.

These are interpreted as specifying that if the TM is in the initial state (*state*) with the current cell containing the initial symbol (*symbol*) then it changes to the new state (*state*′), writes the new symbol (*symbol*′) onto the current cell and moves the head in the specified *direction*.

2.3 Combinatory logic

Combinatory logic (CL) precedes both λ calculus and TMs. It was first developed by Schönfinkel in 1924 (Schönfinkel, 1967) and independently rediscovered by Curry (Curry, 1930) in 1927.

Figure 3 shows three basic combinators.

Each combinator specifies some unitary operation on its arguments and may be considered as a rewrite rule. Thus:

- `I` returns whatever it is applied to;
- `(K` x`)` applied to anything returns x. Alternatively, `K` may be regarded as returning the first of two arguments;
- `S` generalises function application.

In principle, only `S` and `K` are required, as `I` may be defined in terms of them. In practice, to enable the succinct expression of richer constructs, additional combinators are defined on the `S`/`K`/`I` base. We will not consider this further here.

Figure 4 shows an abstract syntax for CL expressions.

As discussed in considerably more detail below, CL expressions are reduced by systematically applying a left most combinator in an expression or bracketed sub-expression to the subsequent combinators and sub-expressions. For example:

$$\texttt{S K I I} \rightarrow_S \texttt{K I (K I)} \rightarrow_K \texttt{I}$$

$$\begin{array}{l} e \rightarrow \texttt{I} \mid \texttt{K} \mid \texttt{S} \mid \texttt{(}\ es\ \texttt{)} \\ es \rightarrow e \mid e\ es \end{array}$$

Fig. 4. Combinatory logic abstract syntax.

2.4 Comparison

There are a number of fundamental differences between TMs, λ calculus and CL.

First of all, both the λ calculus and CL require some external agency capable of applying reduction to arbitrary expressions. In contrast, a set of TM quintuplets characterises an autonomous *machine* for the corresponding algorithm.

TMs correspond to a Harvard architecture with separate code and data spaces. At run-time, the code directs modification of the data but the code itself may not change. In contrast, λ calculus and CL may be thought of as having substitution models of execution, with no distinct notions of code and data[1].

Finally, unlike λ calculus, TMs and CL have no notion of a variable as an abstraction. In CL, the `S` combinator may be used to get the effect of an abstraction, to substitute sub-expressions at specific loci. For TMs, substitution is achieved by marking sections of tape appropriately, distinguishing amongst different marked sections and then copying to required locations.

3 Turing completeness

A model of computation is termed *Turing complete* (TC) if it can be used to compute anything that a TM can compute. In general, to demonstrate that some formalism X is TC, given a formalism Y which is known to be TC, it is necessary to show that everything that can be expressed in X can also be expressed in Y, and vice versa. This may be accomplished by:

1. defining a translation from instances of X to provably equivalent instances of Y , that is a compiler, or a semantics preserving algorithm in Y to reduce instances of X to normal form, that is an interpreter;
2. defining a translation from instances of Y to provably equivalent instances of X, or a semantics preserving algorithm in Y to reduce instances of X to normal form.

Kleene had shown that λ calculus and recursive function theory, on which NTPC is based, were equivalent in 1936 (Kleene, 1936). In 1937, Turing (Turing, 1937a) constructed a TM to reduce λ expressions and also sketched how to convert TMs to recursive functions. Thus, as recursive functions may be converted to λ calculus, so may TMs. Kleene subsequently demonstrated directly that TMs and recursive function theory were equivalent (Kleene, 1952).

Today it is common practice to construct compilers to λ calculus or interpreters in λ calculus when prototyping new languages, albeit to/in rich functional languages, like Standard ML or Haskell, which are equivalent to λ calculus. It is also not uncommon in teaching settings to construct an interpreter for TMs in an arbitrary language, including

[1] In practise, λ calculus and functional language implementations are based on maintaining static code spaces and dynamic stack memories of bound variable/value bindings.

Combinatory logic	λ calculus
$\mathtt{I}\ x \rightarrow_I x$	$\lambda x.x$
$\mathtt{K}\ x\ y \rightarrow_K x$	$\lambda x.\lambda y.x$
$\mathtt{S}\ x\ y\ z \rightarrow_S x\ z\ (y\ z)$	$\lambda x.\lambda y.\lambda z.x\ z\ (y\ z)$

Fig. 5. Basic combinators as λ functions.

functional languages. However, since Turing, it is very rare for an interpreting TM or a compiler to TMs to be constructed for an arbitrary language.

It has long been known that CL logic is Turing complete. It is straightforward to translate CL to λ calculus by simple definition - see Figure 5. In turn, λ calculus may be translated to CL by bracket abstraction (Turner, 1979a).

It is also straightforward to write a CL interpreter in λ calculus, at least in a pure functional language through abstract syntax tree (AST) manipulation. It is certainly conceptually possible, if markedly more complicated, to construct a λ calculus interpreter in CL.

Nonetheless, such demonstrations that CL is TC, while conclusive, remain indirect. In contrast, as part of a wider empirical study of the elegance and expressiveness of computability formalisms (Davidson, 2016), including RASP machines as well as TMs, λ calculus and CL, we have built a TM to directly reduce combinator expressions.

While closing a tiny lacuna in exemplifying the universality of the Church-Turing thesis is of minor consequence, constructing a substantially larger TM than is generally common provides insights into the practical properties and limitations of Turing's foundational formalism. It also provides a basis for comparing the computational characteristics of SKI with computability formalisms for which there are already TMs, in particular Random Access Stored Program machines and the TM itself.

4 Reducing combinator expressions by Turing machine

4.1 Combinator expressions

Our TM will start at the left end of a tape holding a combinator expression, with each symbol on successive tape cells, and systematically locate and reduce sub-expressions for the basic combinators `S`, `K` and `I`. Thus, we may consider the reduction of combinator expressions as an exercise in linear symbol string rewriting.

We will use:

- α, β, γ etc to represent arbitrary symbol sequences;
- e_1, e_2, e_3 etc to represent arbitrary expressions;
- ... to indicate erased symbol sequences.

4.2 I combinator

For the `I` combinator, given: α `I` e β

1. shift e β left over `I`: α e β

4.3 K combinator

For the K combinator, given: α K e_1 e_2 β

1. shift β left over e_2: α K e_1 β
2. shift e_1 β left over K: α e_1 β

4.4 S combinator

For the S combinator, given: α S e_1 e_2 e_3 β

1. copy $(e_2\ e_3)$ right after β, erasing e_2: α S e_1 ... e_3 β $(e_2\ e_3)$
2. copy β right after $(e_1\ e_3)$: α S e_1 ... e_3 β $(e_2\ e_3)$ β
3. shift $(e_2\ e_3)$ β left over β: α S e_1 ... e_3 $(e_2\ e_3)$ β
4. shift e_3 $(e_2\ e_3)$ β left to close up the gap: α S e_1 e_3 $(e_2\ e_3)$ β
5. shift e_1 e_3 $(e_2\ e_3)$ β left over S: α e_1 e_3 $(e_2\ e_3)$ β

4.5 Strategy

We now need to dig deeper and consider how the above accounts may be realised as TM quintuples which can only locate and modify single symbols.

Our TM will proceed by repeatedly finding and reducing CL redexes, that is bracketed sub-expressions consisting of a combinator followed by at least the appropriate number of arguments. We systematically reduce *leftmost-outermost* redexes within what we term a *redex region*, a bracketed sub-expression which may or may not start with a redex.

For example, consider reducing:

```
(S(IKI)(IK))
```

We start with the whole expression as the redex region:

```
{S(IKIK)(IK)}
```

This is not a redex as S lacks a third argument, so we move right to find the next region:

```
(S{IKIK}(IK))
```

This starts with a redex IKIK which we reduce:

```
(S{KIK}(IK))
```

The region still starts with a redex KIK which we reduce:

```
(S{I}(IK))
```

The region no longer starts with a redex so we move right to the next region:

```
(S(I){IK})
```

This new region starts with a redex which we reduce:

```
(S(I){K})
```

There are no more redexes and no more regions so we halt with:

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

originalarg	1arg	2
S	s	5
K	k	4
I	i	1
(	[	\
)	]	/

Fig. 6. Argument rewrites.

4.6 Shifting

All our combinator reduction schemes have phases requiring the leftwards shifting of rightmost symbol sequences β. In some phases, this sequence may be preceded by one or more arguments which must also be shifted left. In this context, we treat such arguments as additional unstructured symbols which extend the left end of the rightmost sequence. To keep track of where we are in the sequence being copied, and the location to which it is being copied, we use distinct symbols to mark the wave fronts.

4.7 Bracketed expressions

The most complicated aspect of the linear reduction of combinator expressions is identifying bracketed expressions. Unlike a typical compiler or interpreter manipulating rich intermediate representations such as abstract syntax trees, we are processing unstructured symbol sequences. Thus, we have to repeatedly parse symbol sequences to locate sub-expression.

Where a sub-expression is a single combinator, this is trivial. For a bracketed expression, however, it is necessary to keep track of matching brackets. Here, we use the leftmost end of the tape, beyond the left expression delimiter, as a stack. We push opening brackets onto the stack and pop them for closing brackets. When the stack is empty we have found a bracketed sub-expression. Turing appears to have originated this technique in his TM for reducing λ expressions (Turing, 1937a). We discuss our approach in more detail below.

When an argument is not erased, it is necessary to keep track of which constituent symbols have already been treated. We do so by systematically replacing symbols according to the table in Figure 6.

Additionally, the combinator at the start of the current redex is rewritten `Z` for `S`, `C` for `K` and `Y` for `I`.

4.8 Extraneous brackets

The `S` combinator introduces additional brackets. Thus, as a result of subsequent reduction steps, nested brackets may accumulate. For example, consider:

`(S I I I)` $\rightarrow_S$
`(I I (I I))` $\rightarrow_I$
`(I (I I))` $\rightarrow_I$
`((I I))`

Hence, at the start of each reduction cycle, any leftmost nested brackets are removed. Once again, this requires bracket matching via a stack.

Note that we do not remove brackets round singleton combinators: to do so complicates the TM to no good purpose.

4.9 Abstracting over bracketed expressions

From the above discussion, bracketed sub-expressions must be identified in contexts involving:

- finding the redex region;
- deleting nested brackets;
- finding a 1st argument - all combinators;
- deleting a 2nd argument - `K`;
- copying and deleting a 2nd argument - `S`
- copying a 3rd argument - `S`

Rather than crafting repetitive bracket matching sequences of quintuplets for each context, we would prefer to construct generic `(`/push and `)`/pop sequences. However, the TM notation lacks any notion of abstracted re-entrant quintuplet sequences analogous to sub-programs. Instead, to get the effect of a sub-program, we have to explicitly remember the context in which a generic sequence is initiated and behave appropriately for that context after the sequence has finished. To do so, we leave unique context-specific symbols on the tape.

Thus, for a `(`/push, our TM will:

- replace the `(` with a context specific symbol;
- move to the first blank cell to the left of the start of expression marker `<`;
- write a `(`;
- move right to the context specific symbol;
- change state depending on the symbol.

And for a `)`/pop, out TM will:

- replace the `)` with a context specific symbol;
- move to the first blank cell to the left of the start of expression marker `<`;
- move right, erase the `(` and move right;
- if the current cell holds a `(` then the whole of the bracketed expression has yet to be found, so:
  - — move right to the context specific symbol;
  - — change state depending on the symbol to find the rest of the expression in the context;
- if the current cell is the start of expression marker `<` then the whole bracketed expression has been found, so:
  - — move right to the context specific symbol;
  - — change state depending on the symbol to initiate the appropriate activity after finding an expression in the context.

4.10 Summary

To summarise our design, given a tape holding a combinator expression, the TM will :

- start at the leftmost expression delimiter, with the whole expression as the redex region;
- repeatedly:
  - remove nested brackets;
  - repeatedly:
    - if the current redex region starts with a redex:
      - reduce that redex;
  - find the next bracketed sub-expression to the right as the next redex region.

5 TM overview

The full TM is given in Appendix A. It has:

- 33 symbols;
- 80 states;
- 687 quintuplets.

Rather than documenting each state in detail, Figures 7 and 8 give an overview of the TM's main phases. The numbers above boxes indicate quintuplets by entry state.

The 33 symbols comprise:

- 4 - boundries of the expression and current redex region;
- 5 - SKI expressions;
- 3 - rewriting combinators at the heads of redexes;
- 10 - rewriting 1st and 2nd arguments;
- 8 - marking push and pop return points;
- 3 - marking gaps and copy/erase return points;

Around a third of the TM is comprised of the re-entrant quintuplets that we factored out. Thus, push and pop for parsing bracketed expressions require 132 quintuplets (19%) for states 66 to 73. Furthermore, copying requires 98 quintuplets (14%) for states 74 to 80.

A full trace of the reduction of `SKII` on a TM interpreter is shown in Appendix B. We now highlight salient points from the trace. On the TM tape:

- the symbol under the head is bracketed `<<...>>`;
- the initial expression is delimited by `<...>` and bracketed `(...)`;
- the current redex region is delimited by `{...}`;
- an empty cell is denoted as _.

For the first reduction of `(SKII)`:
a) initial tape
`<<<>>(SKII)>`
b) found region
`__<SKI<<I>>> - [@4,'I',@4,'I',L]`

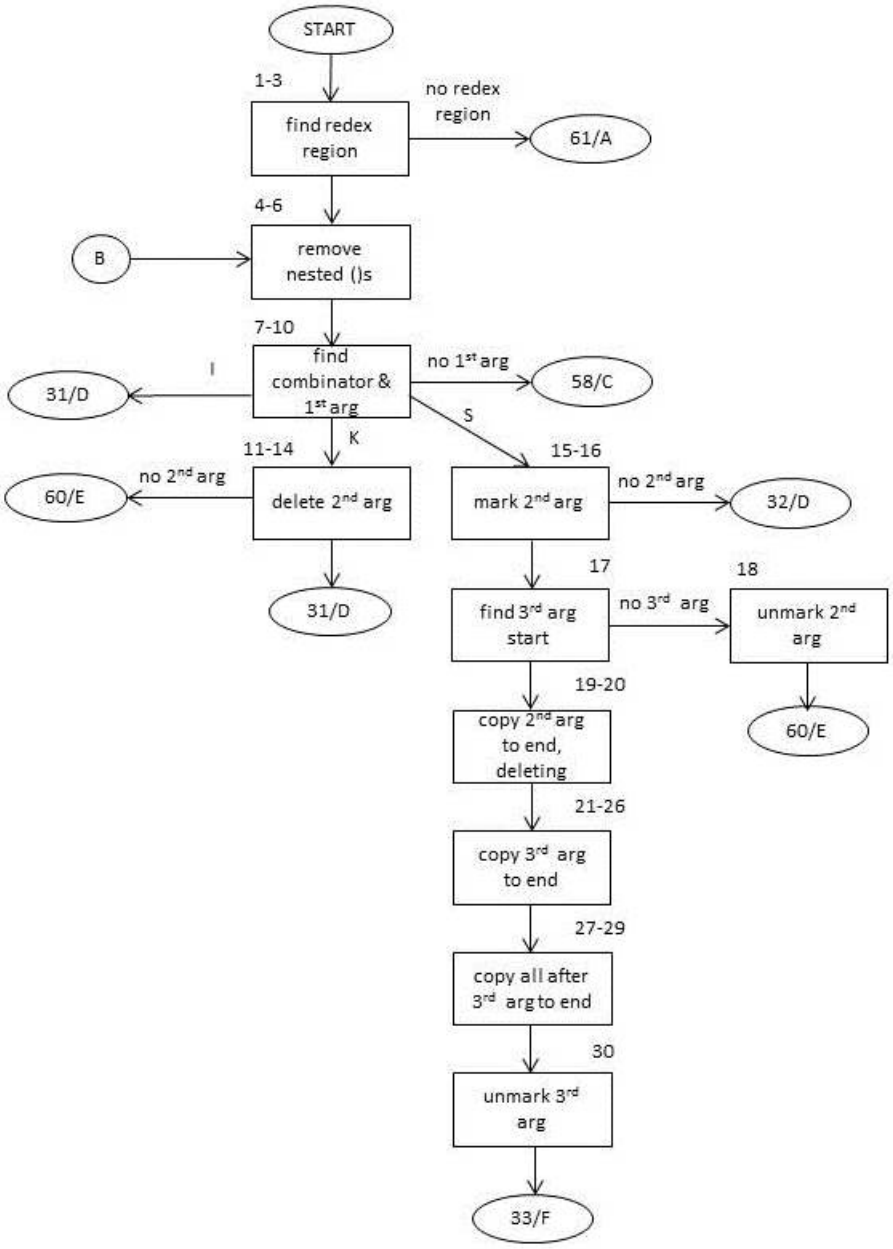

Fig. 7. Combinator reduction TM main phases (a).

c) found S

```
__<Z<<K>>II> - [@7,'K',@9,'k',L]
```

d) marked 1st argument

```
__<Zk<<I>>I> - [@15,'I',@17,'1',R]
```

e) marked 2nd argument

```
__<Zk1<<I>>> - [@17,'I',@19,'I',L]
```

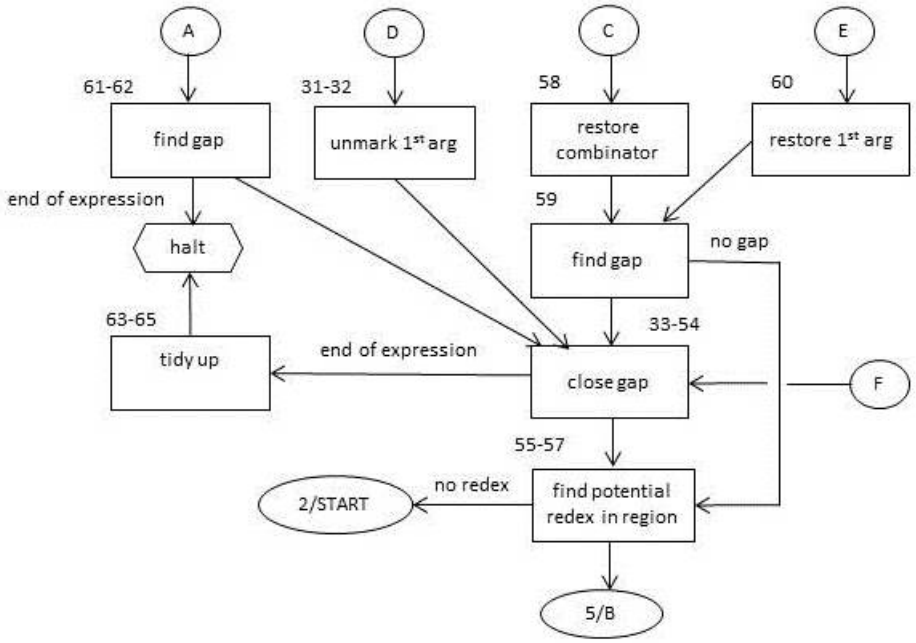

Fig. 8. Combinator reduction TM main phases (b).

f) copied 2nd argument left; 2nd argument deleted
`__<Zk-<<I>>>I - [@20,'I',@23,'i',R]`
g) copied 3rd argument left
`__<Zk-i>II<<_>> - [@26,'_',@27,')',L]`
h) place) after 3rd argument copy
`__<Zk-i>I<<I>>) - [@27,'I',@27,'I',L]`
i) copied all after 3rd argument left;
placed (before 2nd argument copy
`__<Zk-i-(<<I>>I) - [@29,'I',@29,'I',R]`
j) placed >
`__<Zk-i-(II)<<>>> - [@30,'}',@30,'}',L]`
k) rewritten 1st and 2nd arguments
`__<<<Z>>K-I-(II)> - [@30,'Z',@33,'-',R]`
l) erased S
`__<-<<K>>-I-(II)> - [@33,'K',@37,'-',L]`
m) closed gaps
`__<KI(II)<<_>>___ - [@54,'_',@55,'>',L]`

For the reduction of `KI(II)`:
n) found K
`__<C<<I>>(II)>___ - [@7,'I',@9,'i',L]`
o) marked 1st argument
`__<<<C>>i(II)>___ - [@9,'C',@10,'C',R]`

p) erased 2nd argument
```
__<<<C>>i---->___ - [@13,'C',@14,'-',R]
```
q) removed `K`
```
__<-<<i>>---->___ - [@14,'i',@31,'i',R]
```
r) rewritten 1st argument
```
__<<<->>I---->___ - [@32,'-',@33,'-',R]
```
s) closed gaps
```
__<I<<>>>_______ - [@55,'}',@55,'}',L]
```

And for the reduction of `I`:
t) found `I`; no 1st argument
```
__<Y<<>>>_______ - [@7,'}',@57,'}',L]
```
u) no redex in region
```
__<(I)<<>>>_______ - [@2,'>',@60,'*',L]
```
v) no new region so Halt
```
__<(I)<<>>>_______ - [@62,'*',@62,'>',H]
```

6 Characterising Turing machines

It would be interesting to have some way of assessing the quality of a Turing machine. A raw count of quintuplets gives a crude measure of size, and the number of states and symbols indicates the trade off between processing and representation.

If we consider the basic form of a quintuplet:

$$(state, symbol, state', symbol', direction)$$

we can see that, ignoring the direction, there are four possibilities:

- *skip*: initial and new state are the same; initial and new symbol are the same;
- *replace*: initial and new state are the same; initial and new symbol are different;
- *skip and branch*: initial and new state are different; initial and new symbol are the same;
- *replace and branch*: initial and new state are different; initial and final symbol are different.

Thus, we might use counts of the number of occurrences of each form as a further characterising measure, given that:

- skips are used to stay in some state to skip over some designated sequence
- replaces are used to stay in some state while overwriting some designated sequence;
- branches are used to change state, either with or without changing the current symbol.

Table 1. *Comparing TMs.*

	SKI		Minsky UTM		Neary UTM		RASP	
symbols	33		8		11		11	
states	80		24		3		208	
quintuplets	687		113		32		1111	
	quins	quins%	quins	quins%	quins	quins%	quins	quins%
skip	408	59%	68	60%	2	6%	767	69%
skip-branch	120	17%	13	11%	2	6%	191	17%
replace	41	5%	6	5%	20	62%	29	2%
rep-branch	119	17%	26	23%	8	25%	124	11%
skip+skipb	528	76%	81	71%	4	12%	958	86%
rep+repb	160	22%	32	28%	28	87%	153	13%
skip+rep	449	64%	74	65%	22	68%	796	71%
skipb+repb	239	34%	39	34%	10	31%	315	28%

7 Comparing models of computability

Table 1 shows all of these measures for our SKI TM, and for Minsky's Universal TM (UTM) (Minsky, 1972), Neary's (11,3) UTM (Neary, 2008) and Davidson's TM for the RASP (Davidson, 2016).

The Universal Turing machine is a TM that will execute arbitrary TMs against arbitrary tapes, given a suitable representation of both. Turing elaborated the first UTM in 1936 (Turing, 1937b). Subsequently, numerous UTMs have been built.

The Random Access Stored Program (RASP) (Elgot & Robinson, 1964) model is an extension of the RAM model, itself a highly simplified von Neumann machine model. RASP is TC: for details of a UTM implemented as a RASP, and a universal RASP implemented as a TM, see Davidson's PhD (Davidson, 2016).

We can see that the SKI, Minsky's UTM and the RASP all have low ratios of symbols to states (33/80, 8/24,11/208), where Neary's UTM has a high ratio (11/3). We can also see that Neary's UTM has by far the smallest number of quintuplets (32 against SKI 687, Minsky 113 and RASP 1111). Comparing Neary with Minsky suggests a classic representation/computation trade off.

We can also see that the SKI, Minsky's UTM and the RASP all have high ratios of total skip to total replace quintuplets (76/22, 71/28, 86/13). In contrast, Neary's UTM has a low ratio (12/87) which is close to the inverse of the other three TMs. This suggests that Neary's machine makes far more use of tape in situ, taking advantage of a wider range of symbols to discriminate amongst configurations.

However, all models have higher, and comparable, proportions of non-branching compared with branching quintuplets (64/34, 65/34, 68/31, 71/28). This suggests very similar programming styles, with all states doing comparable amounts of work.

The RASP is by far the largest TM. But then the RASP is the most expressive model, and must use tape to represent address/value pairs, with associated searching for and matching of addresses to simulate random access.

The SKI is the next largest TM. Arguably, CL is the least expressive model. Nonetheless, as we have noted above, reducing CL expressions involves their repeated re-parsing, which in turn requires explicit stack manipulation.

8 Related work

Curry et al (Curry *et al.*, 1958; Curry *et al.*, 1972) contain a comprehensive account of the formal properties of combinatory logic, including a proof of the equivalence of CL and λ calculus. Hindley et al (Hindley *et al.*, 1972) is an accessible introduction. Fitch (Fitch, 1974) has detail on realising arithmetic, recursive functions and combinatorial circuits.

At the end of the 20^{th} century, under the impetus of international 5^{th} Generation research programs, CL proved seductive as a basis for parallel implementations of *functional* languages, grounded in λ calculus, on specialised *graph reduction machines* (Clarke *et al.*, 1980; Darlington & Reeve, 1981; Hankin *et al.*, 1985; Peyton Jones *et al.*, 1987; Sale, 1989; Nöcker *et al.*, 1991). The perceived advantages of CL included the absence of name/value bindings, giving very high degrees of implicit parallelism, and the elaboration of simple algorithms to convert between pure functional languages and specialised combinators, through λ lifting (Johnsson, 1985) and super-combinator lifting (Hughes, 1982).

Turner's DPhil (Turner, 1981) provides a foundational account of implementing functional languages through combinators, in particular SASL (Turner, 1979b). Peyton Jones (Peyton Jones, 1987) provides integrated coverage of graph reduction, λ lifting and super-combinator lifting.

Graph reduction proved markedly less efficient than the predominant von Neumann architecture and has been largely abandoned. Nonetheless, remnants of graph reduction may still be found deep inside intermediate stages of implementations of the lazy language Haskell.

Most analysis of Turing machines is in terms of complexity, rather than characteristics of construction. Neary (Neary, 2008) gives a systematic account of the complexities of universal Turing machines, exploring in considerable detail state and symbol trade offs.

9 Conclusions

We have demonstrated what we believe is the first example of a TM that reduces CL expressions represented directly on the TM's tape.

It is interesting to consider the formal status of our demonstration. Turing was elaborating TMs in a world before computers, as mathematical constructs. It is apparent that a convincing demonstration that constructing a TM was plausible was regarded as tantamount to a rigorous demonstration of some property. For example, Naur(Naur, 1993)

found numerous infelicities in Turing's canonical account of the UTM[2]. For example, in Kleene's account (Kleene, 1952) of the equivalence of TMs and recursive function theory, he explicitly leaves fine detail to the reader.

Of course Turing and his contemporaries were not in a position to subject their constructions to the full rigour of machine checking of static properties, or to exhaustively test them, let alone mechanically verify their formal properties. In the longer term, it would be interesting to rationally reconstruct as quintuplets Turing's UTM (Turing, 1937b) and his TM to reduce λ expressions (Turing, 1937a), from his rather convoluted formulation based on function abstraction and substitution.

Acknowledgements

Joe Davidson has been supported by the EU FP7 'Release' Project and the Engineering and Physical Science Research Council project EP/J001058/1 'The Integration and Interaction of Multiple Mathematical Reasoning Processes'.

We wish to thank:

- Roger Hindley of the University of Swansea for useful discussions about the history and properties of combinatory logic;
- Fairouz Kamareddine of Heriot-Watt University for helpful comments on the first draft of this paper.

[2] Alternatively, this may be evidence of laxer standards of refereeing 80 years ago.

Appendix A: TM for SKI combinator reduction

Notation:

- % - comment
- quintuplets have the form: [*oldstate*,*oldsymbol*, *newstate*,*newsymbol*,*direction*]
- @*N* - state *N*

```
%START
% find next region
% find next (
[@1,'<',@2,'<',R]
% find 1st (
[@2,'-',@2,'-',R]
[@2,'(',@66,'!',R]
[@2,'<',@2,'<',R]
[@2,'S',@2,'S',R]
[@2,'K',@2,'K',R]
[@2,'I',@2,'I',R]
[@2,')',@2,')',R]
% if no region then close & halt
[@2,'>',@61,'*',L]
% find ( or )
[@3,'S',@3,'S',R]
[@3,'K',@3,'K',R]
[@3,'I',@3,'I',R]
[@3,'[',@3,'[',R]
[@3,']',@3,']',R]
[@3,'#',@3,'#',R]
[@3,'(',@66,'#',L]
[@3,')',@68,'#',L]
% found region
% left to {
[@4,'-',@4,'-',L]
[@4,'S',@4,'S',L]
[@4,'K',@4,'K',L]
[@4,'I',@4,'I',L]
[@4,'(',@4,'(',L]
[@4,')',@4,')',L]
[@4,'[',@4,'(',L]
[@4,']',@4,')',L]
[@4,'{',@5,'{',R]
% look for nested (...)
% find (
[@5,'-',@5,'-',R]
[@5,'[',@5,'[',R]
[@5,']',@5,']',R]
[@5,'(',@66,'+',L]
[@5,')',@68,'+',R]
% found (...), mark combinator
[@5,'S',@7,'Z',R]
[@5,'K',@7,'C',R]
[@5,'I',@7,'Y',R]
% find rest of (...)
[@6,'-',@6,'-',R]
[@6,'S',@6,'S',R]
[@6,'K',@6,'K',R]
[@6,'I',@6,'I',R]
[@6,'(',@66,'$',L]
[@6,')',@68,'+',L]
[@6,'[',@6,'[',R]
[@6,']',@6,']',R]
% mark 1st arg
[@7,'-',@7,'-',R]
[@7,'S',@9,'s',L]
[@7,'K',@9,'k',L]
[@7,'I',@9,'i',L]
[@7,'(',@66,':',L]
[@7,')',@58,')',L]
[@7,'}',@58,'}',L]
% look for next in 1st arg
[@8,'-',@8,'-',R]
[@8,'S',@8,'s',R]
[@8,'K',@8,'k',R]
[@8,'I',@8,'i',R]
[@8,'(',@66,':',L]
[@8,')',@68,':',L]
% identify combinator
[@9,'[',@9,'[',L]
[@9,']',@9,']',L]
[@9,'s',@9,'s',L]
[@9,'k',@9,'k',L]
[@9,'i',@9,'i',L]
[@9,'-',@9,'-',L]
[@9,'Y',@10,'-',R]
[@9,'C',@11,'C',R]
[@9,'Z',@15,'Z',R]
% I
% right to arg
[@10,'-',@10,'-',R]
[@10,'[',@31,'[',R]
[@10,']',@31,']',R]
[@10,'s',@31,'s',R]
[@10,'k',@31,'k',R]
[@10,'i',@31,'i',R]
% K
% find 2nd arg
[@11,'[',@11,'[',R]
[@11,'s',@11,'s',R]
[@11,'k',@11,'k',R]
[@11,'i',@11,'i',R]
[@11,']',@11,']',R]
[@11,'-',@11,'-',R]
[@11,'S',@13,'-',L]
```

```
[@11,'K',@13,'-',L]
[@11,'I',@13,'-',L]
[@11,'(',@66,'=',L]
% - no 2nd arg
[@11,')',@60,'-',L]
[@11,'}',@60,'}',L]
% delete (...) 2nd arg
[@12,'S',@12,'-',R]
[@12,'K',@12,'-',R]
[@12,'I',@12,'-',R]
[@12,'(',@66,'=',L]
[@12,')',@68,'=',L]
% find next in 2nd arg
[@13,'s',@13,'s',L]
[@13,'k',@13,'k',L]
[@13,'i',@13,'i',L]
[@13,'[',@13,'[',L]
[@13,']',@13,']',L]
[@13,'-',@13,'-',L]
[@13,'C',@14,'-',R]
% go to end of 1st arg
[@14,'-',@14,'-',R]
[@14,'s',@31,'s',R]
[@14,'k',@31,'k',R]
[@14,'i',@31,'i',R]
[@14,'[',@31,'[',R]
% S
% looking for 2nd arg
[@15,'-',@15,'-',R]
[@15,'[',@15,'[',R]
[@15,'s',@15,'s',R]
[@15,'k',@15,'k',R]
[@15,'i',@15,'i',R]
[@15,']',@15,']',R]
[@15,'S',@17,'5',R]
[@15,'K',@17,'4',R]
[@15,'I',@17,'1',R]
[@15,'(',@66,';',L]
[@15,'}',@32,'}',L]
% mark (...) 2nd arg
[@16,'S',@16,'5',R]
[@16,'K',@16,'4',R]
[@16,'I',@16,'1',R]
[@16,'(',@66,';',L]
[@16,')',@68,';',L]
% right to 3rd arg
[@17,'-',@17,'-',R]
[@17,'S',@19,'S',L]
[@17,'K',@19,'K',L]
[@17,'I',@19,'I',L]
[@17,'(',@19,'(',L]
[@17,')',@18,')',L]
[@17,'}',@18,'}',L]
% no 3rd arg
% unmark 2nd & 1st args
[@18,'-',@18,'-',L]
[@18,'\',@18,'(',L]
[@18,'5',@18,'S',L]
[@18,'4',@18,'K',L]
[@18,'1',@18,'I',L]
[@18,'/',@18,')',L]
[@18,'s',@60,'S',L]
[@18,'k',@60,'K',L]
[@18,'i',@60,'I',L]
[@18,']',@60,')',L]
% left to 1st arg
[@19,'/',@19,'/',L]
[@19,'5',@19,'5',L]
[@19,'4',@19,'4',L]
[@19,'1',@19,'1',L]
[@19,'\',@19,'\',L]
[@19,'(',@19,'(',L]
[@19,']',@20,']',R]
[@19,'s',@20,'s',R]
[@19,'k',@20,'k',R]
[@19,'i',@20,'i',R]
[@19,'-',@19,'-',L]
% copy 2nd arg to end,
% deleting
[@20,'\',@78,'*',R]
[@20,'5',@75,'*',R]
[@20,'4',@76,'*',R]
[@20,'1',@77,'*',R]
[@20,'/',@79,'*',R]
[@20,'S',@21,'s',R]
[@20,'K',@22,'k',R]
[@20,'I',@23,'i',R]
[@20,'(',@66,'',L]
[@20,')',@68,'',L]
% copy 3rd arg right
% copy single S
[@21,'S',@21,'S',R]
[@21,'K',@21,'K',R]
[@21,'I',@21,'I',R]
[@21,'(',@21,'(',R]
[@21,')',@21,')',R]
[@21,'}',@21,'}',R]
[@21,'>',@21,'>',R]
[@21,'-',@21,'-',R]
[@21,'_',@26,'S',R]
% copy single K
[@22,'S',@22,'S',R]
[@22,'K',@22,'K',R]
[@22,'I',@22,'I',R]
[@22,'(',@22,'(',R]
[@22,')',@22,')',R]
```

```
[@22,'}',@22,'{',R]
[@22,'>',@22,'>',R]
[@22,'-',@22,'-',R]
[@22,'_',@26,'K',R]
% copy single I
[@23,'S',@23,'S',R]
[@23,'K',@23,'K',R]
[@23,'I',@23,'I',R]
[@23,'(',@23,'(',R]
[@23,')',@23,')',R]
[@23,'}',@23,'}',R]
[@23,'>',@23,'>',R]
[@23,'-',@23,'-',R]
[@23,'_',@26,'I',R]
% copy (...) 3rd arg right
[@24,'-',@24,'-',R]
[@24,'S',@75,'s',R]
[@24,'K',@76,'k',R]
[@24,'I',@77,'i',R]
[@24,'(',@66,'',L]
[@24,')',@68,'',L]
% copy final ) right
[@25,'-',@25,'-',R]
[@25,'}',@25,'}',R]
[@25,'>',@25,'>',R]
[@25,'(',@25,'(',R]
[@25,')',@25,')',R]
[@25,'S',@25,'S',R]
[@25,'K',@25,'K',R]
[@25,'I',@25,'I',R]
[@25,'_',@26,')',R]
% place ) after 3rd arg copy
[@26,'_',@27,')',L]
% left to 3rd arg
[@27,'S',@27,'S',L]
[@27,'K',@27,'K',L]
[@27,'I',@27,'I',L]
[@27,'(',@27,'(',L]
[@27,')',@27,')',L]
[@27,'}',@27,'}',L]
[@27,'>',@27,'>',L]
[@27,'-',@27,'-',L]
[@27,']',@28,']',R]
[@27,'s',@28,'s',R]
[@27,'k',@28,'k',R]
[@27,'i',@28,'i',R]
[@27,'{',@28,'{',R]
% copy everything after 3rd
% arg right, erasing
[@28,'-',@28,'-',R]
[@28,'S',@75,'+',R]
[@28,'K',@76,'+',R]
[@28,'I',@77,'+',R]
[@28,'(',@78,'+',R]
[@28,')',@79,'+',R]
[@28,'}',@80,'+',R]
[@28,'>',@29,'(',R]
% copy > right
[@29,'S',@29,'S',R]
[@29,'K',@29,'K',R]
[@29,'I',@29,'I',R]
[@29,'(',@29,'(',R]
[@29,')',@29,')',R]
[@29,'}',@29,'}',R]
[@29,'>',@29,'>',R]
[@29,'_',@30,'>',L]
% left rewriting 3rd arg
[@30,'S',@30,'S',L]
[@30,'K',@30,'K',L]
[@30,'I',@30,'I',L]
[@30,'(',@30,'(',L]
[@30,')',@30,')',L]
[@30,'}',@30,'}',L]
[@30,'-',@30,'-',L]
[@30,'s',@30,'S',L]
[@30,'k',@30,'K',L]
[@30,'i',@30,'I',L]
[@30,'[',@30,'(',L]
[@30,']',@30,')',L]
[@30,'Z',@33,'-',R]
% CLOSE
% unmark 1st arg
% right over 1st arg
[@31,'[',@31,'[',R]
[@31,']',@31,']',R]
[@31,'s',@31,'s',R]
[@31,'k',@31,'k',R]
[@31,'i',@31,'i',R]
[@31,'(',@32,'(',L]
[@31,')',@32,')',L]
[@31,'S',@32,'S',L]
[@31,'K',@32,'K',L]
[@31,'I',@32,'I',L]
[@31,'}',@32,'}',L]
[@31,'-',@32,'-',L]
% rewrite 1st arg
[@32,']',@32,')',L]
[@32,'[',@32,'(',L]
[@32,'s',@32,'S',L]
[@32,'k',@32,'K',L]
[@32,'i',@32,'I',L]
[@32,'Z',@58,'S',L]
[@32,'C',@58,'K',L]
[@32,'Y',@58,'I',L]
[@32,'-',@33,'-',R]
%  close gaps
```

```
[@33,'-',@33,'-',R]
[@33,'S',@34,'-',L]
[@33,'K',@37,'-',L]
[@33,'I',@40,'-',L]
[@33,'(',@43,'-',L]
[@33,')',@46,'-',L]
[@33,'}',@49,'-',L]
[@33,'>',@52,'_',L]
[@33,'*',@63,'_',L]
% copy S left
[@34,'-',@35,'-',L]
[@34,'S',@34,'S',L]
[@34,'K',@34,'K',L]
[@34,'I',@34,'I',L]
[@34,'(',@34,'(',L]
[@34,')',@34,')',L]
[@34,'}',@34,'}',L]
[@35,'-',@35,'-',L]
[@35,'S',@36,'S',R]
[@35,'K',@36,'K',R]
[@35,'I',@36,'I',R]
[@35,'(',@36,'(',R]
[@35,')',@36,')',R]
[@35,'{',@36,'{',R]
[@35,'*',@36,'*',R]
[@35,'}',@36,'}',R]
[@36,'-',@33,'S',R]
% copy K left
[@37,'-',@38,'-',L]
[@37,'S',@37,'S',L]
[@37,'K',@37,'K',L]
[@37,'I',@37,'I',L]
[@37,'(',@37,'(',L]
[@37,')',@37,')',L]
[@37,'}',@37,'}',L]
[@38,'-',@38,'-',L]
[@38,'S',@39,'S',R]
[@38,'K',@39,'K',R]
[@38,'I',@39,'I',R]
[@38,'(',@39,'(',R]
[@38,')',@39,')',R]
[@38,'{',@39,'{',R]
[@38,'*',@39,'*',R]
[@38,'}',@39,'}',R]
[@39,'-',@33,'K',R]
% copy I left
[@40,'-',@41,'-',L]
[@40,'S',@40,'S',L]
[@40,'K',@40,'K',L]
[@40,'I',@40,'I',L]
[@40,'(',@40,'(',L]
[@40,')',@40,')',L]
[@40,'}',@40,'}',L]
[@41,'-',@41,'-',L]
[@41,'S',@42,'S',R]
[@41,'K',@42,'K',R]
[@41,'I',@42,'I',R]
[@41,'(',@42,'(',R]
[@41,')',@42,')',R]
[@41,'{',@42,'{',R]
[@41,'*',@42,'*',R]
[@41,'}',@42,'}',R]
[@42,'-',@33,'I',R]
% copy ( left
[@43,'-',@44,'-',L]
[@43,'S',@43,'S',L]
[@43,'K',@43,'K',L]
[@43,'I',@43,'I',L]
[@43,'(',@43,'(',L]
[@43,')',@43,')',L]
[@43,'}',@43,'}',L]
[@44,'-',@44,'-',L]
[@44,'S',@45,'S',R]
[@44,'K',@45,'K',R]
[@44,'I',@45,'I',R]
[@44,'(',@45,'(',R]
[@44,')',@45,')',R]
[@44,'{',@45,'{',R]
[@44,'*',@45,'*',R]
[@44,'}',@45,'}',R]
[@45,'-',@33,'(',R]
% copy ) left
[@46,'-',@47,'-',L]
[@46,'S',@46,'S',L]
[@46,'K',@46,'K',L]
[@46,'I',@46,'I',L]
[@46,'(',@46,'(',L]
[@46,')',@46,')',L]
[@46,'}',@46,'}',L]
[@47,'-',@47,'-',L]
[@47,'S',@48,'S',R]
[@47,'K',@48,'K',R]
[@47,'I',@48,'I',R]
[@47,'(',@48,'(',R]
[@47,')',@48,')',R]
[@47,'{',@48,'{',R]
[@47,'*',@48,'*',R]
[@47,'}',@48,'}',R]
[@48,'-',@33,')',R]
% copy } left
[@49,'-',@50,'-',L]
[@49,'S',@49,'S',L]
[@49,'K',@49,'K',L]
[@49,'I',@49,'I',L]
[@49,'(',@49,'(',L]
[@49,')',@49,')',L]
```

```
[@49,'}',@49,'}',L]
[@50,'-',@50,'-',L]
[@50,'S',@51,'S',R]
[@50,'K',@51,'K',R]
[@50,'I',@51,'I',R]
[@50,')',@51,')',R]
[@50,'{',@51,'{',R]
[@50,'*',@51,'*',R]
[@50,'}',@51,'}',R]
[@51,'-',@33,'}',R]
% copy > left
[@52,'-',@53,'_',L]
[@52,'S',@52,'S',L]
[@52,'K',@52,'K',L]
[@52,'I',@52,'I',L]
[@52,'(',@52,'(',L]
[@52,')',@52,')',L]
[@52,'}',@52,'}',L]
[@53,'-',@53,'_',L]
[@53,'S',@54,'S',R]
[@53,'K',@54,'K',R]
[@53,'I',@54,'I',R]
[@53,'(',@54,'(',R]
[@53,')',@54,')',R]
[@53,'{',@54,'{',R]
[@53,'*',@54,'*',R]
[@53,'}',@54,'}',R]
[@54,'_',@55,'>',L]
% left to start of region
[@55,'-',@55,'-',L]
[@55,'S',@55,'S',L]
[@55,'K',@55,'K',L]
[@55,'I',@55,'I',L]
[@55,'(',@55,'(',L]
[@55,')',@55,')',L]
[@55,'}',@55,'}',L]
[@55,'*',@56,'*',R]
[@55,'{',@5,'{',R]
% no redex - right to end of region
[@56,'-',@56,'-',R]
[@56,'S',@56,'S',R]
[@56,'K',@56,'K',R]
[@56,'I',@56,'I',R]
[@56,'(',@56,'(',R]
[@56,')',@56,')',R]
[@56,'}',@57,')',L]
% left to start of region
[@57,'-',@57,'-',L]
[@57,'S',@57,'S',L]
[@57,'K',@57,'K',L]
[@57,'I',@57,'I',L]
[@57,'(',@57,'(',L]
[@57,')',@57,')',L]
[@57,'*',@2,'(',R]
% no 1st arg
% unmark combinator
[@58,'Z',@58,'S',L]
[@58,'C',@58,'K',L]
[@58,'Y',@58,'I',L]
[@58,'-',@58,'-',L]
[@58,'{',@59,'*',R]
% look for gap
[@59,'S',@59,'S',R]
[@59,'K',@59,'K',R]
[@59,'I',@59,'I',R]
[@59,'(',@59,'(',R]
[@59,')',@59,')',R]
[@59,'}',@59,'}',R]
[@59,'>',@55,'>',L]
[@59,'-',@33,'-',R]
% rewrite 1st arg
[@60,'[',@60,'(',L]
[@60,']',@60,')',L]
[@60,'s',@60,'S',L]
[@60,'k',@60,'K',L]
[@60,'i',@60,'I',L]
[@60,'Z',@60,'S',L]
[@60,'C',@60,'K',L]
[@60,'-',@60,'-',L]
[@60,'{',@59,'*',R]
% left to <
[@61,'-',@61,'-',L]
[@61,'S',@61,'S',L]
[@61,'K',@61,'K',L]
[@61,'I',@61,'I',L]
[@61,'(',@61,'(',L]
[@61,')',@61,')',L]
[@61,'<',@62,'<',R]
% HALT
[@62,'S',@62,'S',R]
[@62,'K',@62,'K',R]
[@62,'I',@62,'I',R]
[@62,'(',@62,'(',R]
[@62,')',@62,')',R]
[@62,'*',@62,'>',H]
[@62,'-',@33,'-',R]
% left rewriting gap
[@63,'-',@64,'_',L]
[@63,'S',@63,'S',L]
[@63,'K',@63,'K',L]
[@63,'I',@63,'I',L]
[@63,'(',@63,'(',L]
[@63,')',@63,')',L]
% left to symbol
[@64,'-',@64,'_',L]
[@64,'S',@65,'S',R]
```

```
[@64,'K',@65,'K',R]
[@64,'I',@65,'I',R]
[@64,')',@65,')',R]
% HALT
[@65,'_',@65,'>',H]
% push
% - left to stack
[@66,'-',@66,'-',L]
[@66,'S',@66,'S',L]
[@66,'K',@66,'K',L]
[@66,'I',@66,'I',L]
[@66,'(',@66,'(',L]
[@66,')',@66,')',L]
[@66,'<',@66,'<',L]
[@66,'{',@66,'{',L]
[@66,'s',@66,'s',L]
[@66,'k',@66,'k',L]
[@66,'i',@66,'i',L]
[@66,'[',@66,'[',L]
[@66,']',@66,']',L]
[@66,'5',@66,'5',L]
[@66,'4',@66,'4',L]
[@66,'1',@66,'1',L]
[@66,'\',@66,'\',L]
[@66,'/',@66,'/',L]
[@66,'Z',@66,'Z',L]
[@66,'C',@66,'C',L]
[@66,'Y',@66,'Y',L]
[@66,'!',@66,'!',L]
[@66,'_',@67,'(',R]
% - right to ( source
[@67,'-',@67,'-',R]
[@67,'S',@67,'S',R]
[@67,'K',@67,'K',R]
[@67,'I',@67,'I',R]
[@67,'(',@67,'(',R]
[@67,')',@67,')',R]
[@67,'<',@67,'<',R]
[@67,'{',@67,'{',R]
[@67,'s',@67,'s',R]
[@67,'k',@67,'k',R]
[@67,'i',@67,'i',R]
[@67,'[',@67,'[',R]
[@67,']',@67,']',R]
[@67,'5',@67,'5',R]
[@67,'4',@67,'4',R]
[@67,'1',@67,'1',R]
[@67,'\',@67,'\',R]
[@67,'/',@67,'/',R]
[@67,'Z',@67,'Z',R]
[@67,'C',@67,'C',R]
[@67,'Y',@67,'Y',R]
[@67,'!',@3,'{',R]
[@67,'#',@3,'[',R]
[@67,'+',@6,'-',R]
[@67,'\$',@6,'[',R]
[@67,':',@8,'[',R]
[@67,'=',@12,'-',R]
[@67,';',@16,'\',R]
[@67,'',@78,'[',R]
% pop
% - left to stack
[@68,'-',@68,'-',L]
[@68,'S',@68,'S',L]
[@68,'K',@68,'K',L]
[@68,'I',@68,'I',L]
[@68,'(',@68,'(',L]
[@68,')',@68,')',L]
[@68,'{',@68,'{',L]
[@68,'s',@68,'s',L]
[@68,'k',@68,'k',L]
[@68,'i',@68,'i',L]
[@68,'[',@68,'[',L]
[@68,']',@68,']',L]
[@68,'5',@68,'5',L]
[@68,'4',@68,'4',L]
[@68,'1',@68,'1',L]
[@68,'\',@68,'\',L]
[@68,'/',@68,'/',L]
[@68,'Z',@68,'Z',L]
[@68,'C',@68,'C',L]
[@68,'Y',@68,'Y',L]
[@68,'<',@69,'<',L]
% found stack
[@69,'(',@69,'(',L]
[@69,'_',@70,'_',R]
% popped
[@70,'(',@71,'_',R]
[@71,'(',@72,'(',R]
[@71,'<',@73,'<',R]
% not end of (...)
[@72,'-',@72,'-',R]
[@72,'S',@72,'S',R]
[@72,'K',@72,'K',R]
[@72,'I',@72,'I',R]
[@72,'(',@72,'(',R]
[@72,')',@72,')',R]
[@72,'<',@72,'<',R]
[@72,'{',@72,'{',R]
[@72,'s',@72,'s',R]
[@72,'k',@72,'k',R]
[@72,'i',@72,'i',R]
[@72,'[',@72,'[',R]
[@72,']',@72,']',R]
[@72,'5',@72,'5',R]
[@72,'4',@72,'4',R]
```

```
[@72,'1',@72,'1',R]
[@72,'\',@72,'\',R]
[@72,'/',@72,'/',R]
[@72,'Z',@72,'Z',R]
[@72,'C',@72,'C',R]
[@72,'Y',@72,'Y',R]
[@72,'#',@3,']',L]
[@72,'+',@6,']',L]
[@72,':',@8,']',R]
[@72,'=',@12,'-',R]
[@72,';',@16,'/',R]
[@72,'',@79,']',R]
% end of (...)
[@73,'-',@73,'-',R]
[@73,'S',@73,'S',R]
[@73,'K',@73,'K',R]
[@73,'I',@73,'I',R]
[@73,'(',@73,'(',R]
[@73,')',@73,')',R]
[@73,'<',@73,'<',R]
[@73,'{',@73,'{',R]
[@73,'s',@73,'s',R]
[@73,'k',@73,'k',R]
[@73,'i',@73,'i',R]
[@73,'[',@73,'[',R]
[@73,']',@73,']',R]
[@73,'5',@73,'5',R]
[@73,'4',@73,'4',R]
[@73,'1',@73,'1',R]
[@73,'\',@73,'\',R]
[@73,'/',@73,'/',R]
[@73,'Z',@73,'Z',R]
[@73,'C',@73,'C',R]
[@73,'Y',@73,'Y',R]
[@73,'#',@4,'}',L]
[@73,'+',@4,'-',L]
[@73,':',@9,']',L]
[@73,'=',@13,'-',L]
[@73,';',@17,'/',R]
[@73,'',@25,']',R]
% copy right
% left to copy mark
[@74,'(',@74,'(',L]
[@74,'S',@74,'S',L]
[@74,'K',@74,'K',L]
[@74,'I',@74,'I',L]
[@74,')',@74,')',L]
[@74,'>',@74,'>',L]
[@74,'}',@74,'}',L]
[@74,'\',@74,'\',L]
[@74,'5',@74,'5',L]
[@74,'4',@74,'4',L]
[@74,'1',@74,'1',L]
[@74,'/',@74,'/',L]
[@74,'-',@74,'-',L]
[@74,'*',@20,'-',R]
[@74,'+',@28,'-',R]
[@74,'[',@24,'[',R]
[@74,'s',@24,'s',R]
[@74,'k',@24,'k',R]
[@74,'i',@24,'i',R]
[@74,']',@24,']',R]
% copy S
[@75,'-',@75,'-',R]
[@75,'(',@75,'(',R]
[@75,'5',@75,'5',R]
[@75,'4',@75,'4',R]
[@75,'1',@75,'1',R]
[@75,'/',@75,'/',R]
[@75,'}',@75,'}',R]
[@75,'S',@75,'S',R]
[@75,'K',@75,'K',R]
[@75,'I',@75,'I',R]
[@75,'(',@75,'(',R]
[@75,')',@75,')',R]
[@75,'>',@75,'>',R]
[@75,'_',@74,'S',L]
%copy K
[@76,'-',@76,'-',R]
[@76,'(',@76,'(',R]
[@76,'5',@76,'5',R]
[@76,'4',@76,'4',R]
[@76,'1',@76,'1',R]
[@76,'/',@76,'/',R]
[@76,'}',@76,'}',R]
[@76,'S',@76,'S',R]
[@76,'K',@76,'K',R]
[@76,'I',@76,'I',R]
[@76,'(',@76,'(',R]
[@76,')',@76,')',R]
[@76,'>',@76,'>',R]
[@76,'_',@74,'K',L]
% copy I
[@77,'-',@77,'-',R]
[@77,'(',@77,'(',R]
[@77,'5',@77,'5',R]
[@77,'4',@77,'4',R]
[@77,'1',@77,'1',R]
[@77,'/',@77,'/',R]
[@77,'}',@77,'}',R]
[@77,'S',@77,'S',R]
[@77,'K',@77,'K',R]
[@77,'I',@77,'I',R]
[@77,'(',@77,'(',R]
[@77,')',@77,')',R]
[@77,'>',@77,'>',R]
```

```
[@77,'_',@74,'I',L]
% copy (
[@78,'-',@78,'-',R]
[@78,'\',@78,'\',R]
[@78,'5',@78,'5',R]
[@78,'4',@78,'4',R]
[@78,'1',@78,'1',R]
[@78,'/',@78,'/',R]
[@78,'}',@78,'}',R]
[@78,'S',@78,'S',R]
[@78,'K',@78,'K',R]
[@78,'I',@78,'I',R]
[@78,'(',@78,'(',R]
[@78,')',@78,')',R]
[@78,'>',@78,'>',R]
[@78,'_',@74,'(',L]
% copy )
[@79,'-',@79,'-',R]
[@79,'(',@79,'(',R]
[@79,'5',@79,'5',R]
[@79,'4',@79,'4',R]
[@79,'1',@79,'1',R]
[@79,'/',@79,'/',R]
[@79,'}',@79,'}',R]
[@79,'S',@79,'S',R]
[@79,'K',@79,'K',R]
[@79,'I',@79,'I',R]
[@79,'(',@79,'(',R]
[@79,')',@79,')',R]
[@79,'>',@79,'>',R]
[@79,'_',@74,')',L]
% copy }
[@80,'S',@80,'S',R]
[@80,'K',@80,'K',R]
[@80,'I',@80,'I',R]
[@80,'(',@80,'(',R]
[@80,')',@80,')',R]
[@80,'}',@80,'}',R]
[@80,'>',@80,'>',R]
[@80,'_',@74,'}',L]
```

Appendix B: reduction of `SKII`

The following shows a full trace of the reduction of `SKII` by the machine in Appendix A, on a TM simulator written in Haskell and available, with a test suite, from the first author. The trace format is:

- current state;
- tape with head marked by `<<...>>`;
- matched quintuplet.

A blank cell is indicated by _.

```
<<<>>(SKII)>
@1: <<<>>(SKII)> - [@1,'<',@2,'<',R]
@2: <<<(>>SKII)> - [@2,'(',@84,'!',R]
@84: <!<<S>>KII)> - [@84,'S',@84,'S',L]
@84: <<<!>>SKII)> - [@84,'!',@84,'!',L]
@84: <<<>>!SKII)> - [@84,'<',@84,'<',L]
@84: <<_>><!SKII)> - [@84,'_',@85,'(',R]
@85: (<<<>>!SKII)> - [@85,'<',@85,'<',R]
@85: (<<<!>>SKII)> - [@85,'!',@3,'{',R]
@3: (<{<<S>>KII)> - [@3,'S',@3,'S',R]
@3: (<{S<<K>>II)> - [@3,'K',@3,'K',R]
@3: (<{SK<<I>>I)> - [@3,'I',@3,'I',R]
@3: (<{SKI<<I>>)> - [@3,'I',@3,'I',R]
@3: (<{SKII<<)>>> - [@3,')',@86,'#',L]
@86: (<{SKI<<I>>#> - [@86,'I',@86,'I',L]
@86: (<{SK<<I>>I#> - [@86,'I',@86,'I',L]
@86: (<{S<<K>>II#> - [@86,'K',@86,'K',L]
@86: (<{<<S>>KII#> - [@86,'S',@86,'S',L]
@86: (<<<{>>SKII#> - [@86,'{',@86,'{',L]
@86: (<<<>>{SKII#> - [@86,'<',@87,'<',L]
@87: <<(>><{SKII#> - [@87,'(',@87,'(',L]
@87: <<_>>(<{SKII#> - [@87,'_',@88,'_',R]
@88: _<<(>><{SKII#> - [@88,'(',@89,'_',R]
@89: __<<<>>{SKII#> - [@89,'<',@91,'<',R]
@91: __<<<{>>SKII#> - [@91,'{',@91,'{',R]
@91: __<{<<S>>KII#> - [@91,'S',@91,'S',R]
@91: __<{S<<K>>II#> - [@91,'K',@91,'K',R]
@91: __<{SK<<I>>I#> - [@91,'I',@91,'I',R]
@91: __<{SKI<<I>>#> - [@91,'I',@91,'I',R]
@91: __<{SKII<<#>>> - [@91,'#',@4,'}',L]
@4: __<{SKI<<I>>}> - [@4,'I',@4,'I',L]
@4: __<{SK<<I>>I}> - [@4,'I',@4,'I',L]
@4: __<{S<<K>>II}> - [@4,'K',@4,'K',L]
@4: __<{<<S>>KII}> - [@4,'S',@4,'S',L]
@4: __<<<{>>SKII}> - [@4,'{',@5,'{',R]
@5: __<{<<S>>KII}> - [@5,'S',@7,'Z',R]
@7: __<{Z<<K>>II}> - [@7,'K',@9,'k',L]
@9: __<{<<Z>>kII}> - [@9,'Z',@14,'Z',R]
@14: __<{Z<<k>>II}> - [@14,'k',@14,'k',R]
@14: __<{Zk<<I>>I}> - [@14,'I',@16,'1',R]
@16: __<{Zk1<<I>>}> - [@16,'I',@18,'I',L]
@18: __<{Zk<<1>>I}> - [@18,'1',@18,'1',L]
@18: __<{Z<<k>>1I}> - [@18,'k',@19,'k',R]
@19: __<{Zk<<1>>I}> - [@19,'1',@24,'*',R]
```

```
@24: __<{Zk*<<I>>}> - [@24,'I',@24,'I',R]
@24: __<{Zk*I<<}>>> - [@24,'}',@24,'}',R]
@24: __<{Zk*I}<<>>> - [@24,'>',@24,'>',R]
@24: __<{Zk*I}><<_>> - [@24,'_',@21,'I',L]
@21: __<{Zk*I}<<>>>I - [@21,'>',@21,'>',L]
@21: __<{Zk*I<<}>>>I - [@21,'}',@21,'}',L]
@21: __<{Zk*<<I>>}>I - [@21,'I',@21,'I',L]
@21: __<{Zk<<*>>I}>I - [@21,'*',@19,'-',R]
@19: __<{Zk-<<I>>}>I - [@19,'I',@28,'i',R]
@28: __<{Zk-i<<}>>>I - [@28,'}',@28,'}',R]
@28: __<{Zk-i}<<>>>I - [@28,'>',@28,'>',R]
@28: __<{Zk-i}><<I>> - [@28,'I',@28,'I',R]
@28: __<{Zk-i}>I<<_>> - [@28,'_',@37,'I',R]
@37: __<{Zk-i}>II<<_>> - [@37,'_',@38,')',L]
@38: __<{Zk-i}>I<<I>>) - [@38,'I',@38,'I',L]
@38: __<{Zk-i}><<I>>I) - [@38,'I',@38,'I',L]
@38: __<{Zk-i}<<>>>II) - [@38,'>',@38,'>',L]
@38: __<{Zk-i<<}>>>II) - [@38,'}',@38,'}',L]
@38: __<{Zk-<<i>>}>II) - [@38,'i',@39,'i',R]
@39: __<{Zk-i<<}>>>II) - [@39,'}',@45,'-',R]
@45: __<{Zk-i-<<>>>II) - [@45,'>',@45,'>',R]
@45: __<{Zk-i-><<I>>I) - [@45,'I',@45,'I',R]
@45: __<{Zk-i->I<<I>>) - [@45,'I',@45,'I',R]
@45: __<{Zk-i->II<<)>> - [@45,')',@45,')',R]
@45: __<{Zk-i->II)<<_>> - [@45,'_',@47,'}',L]
@47: __<{Zk-i->II<<)>>} - [@47,')',@47,')',L]
@47: __<{Zk-i->I<<I>>)} - [@47,'I',@47,'I',L]
@47: __<{Zk-i-><<I>>I)} - [@47,'I',@47,'I',L]
@47: __<{Zk-i-<<>>>II)} - [@47,'>',@47,'>',L]
@47: __<{Zk-i<<->>>II)} - [@47,'-',@39,'-',R]
@39: __<{Zk-i-<<>>>II)} - [@39,'>',@46,'(',R]
@46: __<{Zk-i-(<<I>>I)} - [@46,'I',@46,'I',R]
@46: __<{Zk-i-(I<<I>>)} - [@46,'I',@46,'I',R]
@46: __<{Zk-i-(II<<)>>} - [@46,')',@46,')',R]
@46: __<{Zk-i-(II)<<}>> - [@46,'}',@46,'}',R]
@46: __<{Zk-i-(II)}<<_>> - [@46,'_',@48,'>',L]
@48: __<{Zk-i-(II)<<}>>> - [@48,'}',@48,'}',L]
@48: __<{Zk-i-(II<<)>>}> - [@48,')',@48,')',L]
@48: __<{Zk-i-(I<<I>>)}> - [@48,'I',@48,'I',L]
@48: __<{Zk-i-(<<I>>I)}> - [@48,'I',@48,'I',L]
@48: __<{Zk-i-<<(>>II)}> - [@48,'(',@48,'(',L]
@48: __<{Zk-i<<->>(II)}> - [@48,'-',@48,'-',L]
@48: __<{Zk-<<i>>-(II)}> - [@48,'i',@48,'I',L]
@48: __<{Zk<<->>I-(II)}> - [@48,'-',@48,'-',L]
@48: __<{Z<<k>>-I-(II)}> - [@48,'k',@48,'K',L]
@48: __<{<<Z>>K-I-(II)}> - [@48,'Z',@51,'-',R]
@51: __<{-<<K>>-I-(II)}> - [@51,'K',@55,'-',L]
@55: __<{<<->>--I-(II)}> - [@55,'-',@56,'-',L]
@56: __<<<{>>---I-(II)}> - [@56,'{',@57,'{',R]
@57: __<{<<->>--I-(II)}> - [@57,'-',@51,'K',R]
@51: __<{K<<->>-I-(II)}> - [@51,'-',@51,'-',R]
@51: __<{K-<<->>I-(II)}> - [@51,'-',@51,'-',R]
@51: __<{K--<<I>>-(II)}> - [@51,'I',@58,'-',L]
```

```
@58: __<{K-<<->>--(II)}> - [@58,'-',@59,'-',L]
@59: __<{K<<->>---(II)}> - [@59,'-',@59,'-',L]
@59: __<{<<K>>----(II)}> - [@59,'K',@60,'K',R]
@60: __<{K<<->>---(II)}> - [@60,'-',@51,'I',R]
@51: __<{KI<<->>--(II)}> - [@51,'-',@51,'-',R]
@51: __<{KI-<<->>-(II)}> - [@51,'-',@51,'-',R]
@51: __<{KI--<<->>(II)}> - [@51,'-',@51,'-',R]
@51: __<{KI---<<(>>II)}> - [@51,'(',@61,'-',L]
@61: __<{KI--<<->>-II)}> - [@61,'-',@62,'-',L]
@62: __<{KI-<<->>--II)}> - [@62,'-',@62,'-',L]
@62: __<{KI<<->>---II)}> - [@62,'-',@62,'-',L]
@62: __<{K<<I>>----II)}> - [@62,'I',@63,'I',R]
@63: __<{KI<<->>---II)}> - [@63,'-',@51,'(',R]
@51: __<{KI(<<->>--II)}> - [@51,'-',@51,'-',R]
@51: __<{KI(-<<->>-II)}> - [@51,'-',@51,'-',R]
@51: __<{KI(--<<->>II)}> - [@51,'-',@51,'-',R]
@51: __<{KI(---<<I>>I)}> - [@51,'I',@58,'-',L]
@58: __<{KI(--<<->>-I)}> - [@58,'-',@59,'-',L]
@59: __<{KI(-<<->>--I)}> - [@59,'-',@59,'-',L]
@59: __<{KI(<<->>---I)}> - [@59,'-',@59,'-',L]
@59: __<{KI<<(>>----I)}> - [@59,'(',@60,'(',R]
@60: __<{KI(<<->>---I)}> - [@60,'-',@51,'I',R]
@51: __<{KI(I<<->>--I)}> - [@51,'-',@51,'-',R]
@51: __<{KI(I-<<->>-I)}> - [@51,'-',@51,'-',R]
@51: __<{KI(I--<<->>I)}> - [@51,'-',@51,'-',R]
@51: __<{KI(I---<<I>>)}> - [@51,'I',@58,'-',L]
@58: __<{KI(I--<<->>-)}> - [@58,'-',@59,'-',L]
@59: __<{KI(I-<<->>--)}> - [@59,'-',@59,'-',L]
@59: __<{KI(I<<->>---)}> - [@59,'-',@59,'-',L]
@59: __<{KI(<<I>>----)}> - [@59,'I',@60,'I',R]
@60: __<{KI(I<<->>---)}> - [@60,'-',@51,'I',R]
@51: __<{KI(II<<->>--)}> - [@51,'-',@51,'-',R]
@51: __<{KI(II-<<->>-)}> - [@51,'-',@51,'-',R]
@51: __<{KI(II--<<->>)}> - [@51,'-',@51,'-',R]
@51: __<{KI(II---<<)>>}> - [@51,')',@64,'-',L]
@64: __<{KI(II--<<->>-}> - [@64,'-',@65,'-',L]
@65: __<{KI(II-<<->>--}> - [@65,'-',@65,'-',L]
@65: __<{KI(II<<->>---}> - [@65,'-',@65,'-',L]
@65: __<{KI(I<<I>>----}> - [@65,'I',@66,'I',R]
@66: __<{KI(II<<->>---}> - [@66,'-',@51,')',R]
@51: __<{KI(II)<<->>--}> - [@51,'-',@51,'-',R]
@51: __<{KI(II)-<<->>-}> - [@51,'-',@51,'-',R]
@51: __<{KI(II)--<<->>}> - [@51,'-',@51,'-',R]
@51: __<{KI(II)---<<}>>> - [@51,'}',@67,'-',L]
@67: __<{KI(II)--<<->>-> - [@67,'-',@68,'-',L]
@68: __<{KI(II)-<<->>--> - [@68,'-',@68,'-',L]
@68: __<{KI(II)<<->>---> - [@68,'-',@68,'-',L]
@68: __<{KI(II<<)>>----> - [@68,')',@69,')',R]
@69: __<{KI(II)<<->>---> - [@69,'-',@51,'}',R]
@51: __<{KI(II)}<<->>--> - [@51,'-',@51,'-',R]
@51: __<{KI(II)}-<<->>-> - [@51,'-',@51,'-',R]
@51: __<{KI(II)}--<<->>> - [@51,'-',@51,'-',R]
@51: __<{KI(II)}---<<>>> - [@51,'>',@70,'_',L]
```

```
@70: __<{KI(II)}--<<->>_ - [@70,'-',@71,'_',L]
@71: __<{KI(II)}-<<->>__ - [@71,'-',@71,'_',L]
@71: __<{KI(II)}<<->>___ - [@71,'-',@71,'_',L]
@71: __<{KI(II)<<}>>____ - [@71,'}',@72,'}',R]
@72: __<{KI(II)}<<_>>___ - [@72,'_',@73,'>',L]
@73: __<{KI(II)<<}>>>___ - [@73,'}',@73,'}',L]
@73: __<{KI(II<<)>>}>___ - [@73,')',@73,')',L]
@73: __<{KI(I<<I>>)}>___ - [@73,'I',@73,'I',L]
@73: __<{KI(<<I>>I)}>___ - [@73,'I',@73,'I',L]
@73: __<{KI<<(>>II)}>___ - [@73,'(',@73,'(',L]
@73: __<{K<<I>>(II)}>___ - [@73,'I',@73,'I',L]
@73: __<{<<K>>I(II)}>___ - [@73,'K',@73,'K',L]
@73: __<<<{>>KI(II)}>___ - [@73,'{',@5,'{',R]
@5: __<{<<K>>I(II)}>___ - [@5,'K',@7,'C',R]
@7: __<{C<<I>>(II)}>___ - [@7,'I',@9,'i',L]
@9: __<{<<C>>i(II)}>___ - [@9,'C',@10,'C',R]
@10: __<{C<<i>>(II)}>___ - [@10,'i',@10,'i',R]
@10: __<{Ci<<(>>II)}>___ - [@10,'(',@84,'=',L]
@84: __<{C<<i>>=II)}>___ - [@84,'i',@84,'i',L]
@84: __<{<<C>>i=II)}>___ - [@84,'C',@84,'C',L]
@84: __<<<{>>Ci=II)}>___ - [@84,'{',@84,'{',L]
@84: __<<<>>{Ci=II)}>___ - [@84,'<',@84,'<',L]
@84: _<<_>><{Ci=II)}>___ - [@84,'_',@85,'(',R]
@85: _(<<<>>{Ci=II)}>___ - [@85,'<',@85,'<',R]
@85: _(<<<{>>Ci=II)}>___ - [@85,'{',@85,'{',R]
@85: _(<{<<C>>i=II)}>___ - [@85,'C',@85,'C',R]
@85: _(<{C<<i>>=II)}>___ - [@85,'i',@85,'i',R]
@85: _(<{Ci<<=>>II)}>___ - [@85,'=',@11,'-',R]
@11: _(<{Ci-<<I>>I)}>___ - [@11,'I',@11,'-',R]
@11: _(<{Ci--<<I>>)}>___ - [@11,'I',@11,'-',R]
@11: _(<{Ci---<<)>>}>___ - [@11,')',@86,'=',L]
@86: _(<{Ci--<<->>=}>___ - [@86,'-',@86,'-',L]
@86: _(<{Ci-<<->>-=}>___ - [@86,'-',@86,'-',L]
@86: _(<{Ci<<->>--=}>___ - [@86,'-',@86,'-',L]
@86: _(<{C<<i>>---=}>___ - [@86,'i',@86,'i',L]
@86: _(<{<<C>>i---=}>___ - [@86,'C',@86,'C',L]
@86: _(<<<{>>Ci---=}>___ - [@86,'{',@86,'{',L]
@86: _(<<<>>{Ci---=}>___ - [@86,'<',@87,'<',L]
@87: _<<(>><{Ci---=}>___ - [@87,'(',@87,'(',L]
@87: <<_>>(<{Ci---=}>___ - [@87,'_',@88,'_',R]
@88: _<<(>><{Ci---=}>___ - [@88,'(',@89,'_',R]
@89: __<<<>>{Ci---=}>___ - [@89,'<',@91,'<',R]
@91: __<<<{>>Ci---=}>___ - [@91,'{',@91,'{',R]
@91: __<{<<C>>i---=}>___ - [@91,'C',@91,'C',R]
@91: __<{C<<i>>---=}>___ - [@91,'i',@91,'i',R]
@91: __<{Ci<<->>--=}>___ - [@91,'-',@91,'-',R]
@91: __<{Ci-<<->>-=}>___ - [@91,'-',@91,'-',R]
@91: __<{Ci--<<->>=}>___ - [@91,'-',@91,'-',R]
@91: __<{Ci---<<=>>}>___ - [@91,'=',@12,'-',L]
@12: __<{Ci--<<->>-}>___ - [@12,'-',@12,'-',L]
@12: __<{Ci-<<->>--}>___ - [@12,'-',@12,'-',L]
@12: __<{Ci<<->>---}>___ - [@12,'-',@12,'-',L]
@12: __<{C<<i>>----}>___ - [@12,'i',@12,'i',L]
```

```
@12: __<{<<C>>i----}>___ - [@12,'C',@13,'-',R]
@13: __<{-<<i>>----}>___ - [@13,'i',@49,'i',R]
@49: __<{-i<<->>---}>___ - [@49,'-',@50,'-',L]
@50: __<{-<<i>>----}>___ - [@50,'i',@50,'I',L]
@50: __<{<<->>I----}>___ - [@50,'-',@51,'-',R]
@51: __<{-<<I>>----}>___ - [@51,'I',@58,'-',L]
@58: __<{<<->>-----}>___ - [@58,'-',@59,'-',L]
@59: __<<<{>>------}>___ - [@59,'{',@60,'{',R]
@60: __<{<<->>-----}>___ - [@60,'-',@51,'I',R]
@51: __<{I<<->>----}>___ - [@51,'-',@51,'-',R]
@51: __<{I-<<->>---}>___ - [@51,'-',@51,'-',R]
@51: __<{I--<<->>--}>___ - [@51,'-',@51,'-',R]
@51: __<{I---<<->>-}>___ - [@51,'-',@51,'-',R]
@51: __<{I----<<->>}>___ - [@51,'-',@51,'-',R]
@51: __<{I-----<<}>>>___ - [@51,'}',@67,'-',L]
@67: __<{I----<<->>->___ - [@67,'-',@68,'-',L]
@68: __<{I---<<->>-->___ - [@68,'-',@68,'-',L]
@68: __<{I--<<->>--->___ - [@68,'-',@68,'-',L]
@68: __<{I-<<->>---->___ - [@68,'-',@68,'-',L]
@68: __<{I<<->>----->___ - [@68,'-',@68,'-',L]
@68: __<{<<I>>------>___ - [@68,'I',@69,'I',R]
@69: __<{I<<->>----->___ - [@69,'-',@51,'}',R]
@51: __<{I}<<->>---->___ - [@51,'-',@51,'-',R]
@51: __<{I}-<<->>--->___ - [@51,'-',@51,'-',R]
@51: __<{I}--<<->>-->___ - [@51,'-',@51,'-',R]
@51: __<{I}---<<->>->___ - [@51,'-',@51,'-',R]
@51: __<{I}----<<->>>___ - [@51,'-',@51,'-',R]
@51: __<{I}-----<<>>>___ - [@51,'>',@70,'_',L]
@70: __<{I}----<<->>____ - [@70,'-',@71,'_',L]
@71: __<{I}---<<->>_____ - [@71,'-',@71,'_',L]
@71: __<{I}--<<->>______ - [@71,'-',@71,'_',L]
@71: __<{I}-<<->>_______ - [@71,'-',@71,'_',L]
@71: __<{I}<<->>________ - [@71,'-',@71,'_',L]
@71: __<{I<<}>>_________ - [@71,'}',@72,'}',R]
@72: __<{I}<<_>>________ - [@72,'_',@73,'>',L]
@73: __<{I<<}>>>________ - [@73,'}',@73,'}',L]
@73: __<{<<I>>}>________ - [@73,'I',@73,'I',L]
@73: __<<<{>>I}>________ - [@73,'{',@5,'{',R]
@5: __<{<<I>>}>________ - [@5,'I',@7,'Y',R]
@7: __<{Y<<}>>>________ - [@7,'}',@76,'}',L]
@76: __<{<<Y>>}>________ - [@76,'Y',@76,'I',L]
@76: __<<<{>>I}>________ - [@76,'{',@77,'*',R]
@77: __<*<<I>>}>________ - [@77,'I',@77,'I',R]
@77: __<*I<<}>>>________ - [@77,'}',@77,'}',R]
@77: __<*I}<<>>>________ - [@77,'>',@73,'>',L]
@73: __<*I<<}>>>________ - [@73,'}',@73,'}',L]
@73: __<*<<I>>}>________ - [@73,'I',@73,'I',L]
@73: __<<<*>>I}>________ - [@73,'*',@74,'*',R]
@74: __<*<<I>>}>________ - [@74,'I',@74,'I',R]
@74: __<*I<<}>>>________ - [@74,'}',@75,')',L]
@75: __<*<<I>>)>________ - [@75,'I',@75,'I',L]
@75: __<<<*>>I)>________ - [@75,'*',@2,'(',R]
@2: __<(<<I>>)>________ - [@2,'I',@2,'I',R]
```

```
@2: __<(I<<)>>>________ - [@2,')',@2,')',R]
@2: __<(I)<<>>>________ - [@2,'>',@79,'*',L]
@79: __<(I<<)>>*________ - [@79,')',@79,')',L]
@79: __<(<<I>>)*________ - [@79,'I',@79,'I',L]
@79: __<<<(>>I)*________ - [@79,'(',@79,'(',L]
@79: __<<<>>(I)*________ - [@79,'<',@80,'<',R]
@80: __<<<(>>I)*________ - [@80,'(',@80,'(',R]
@80: __<(<<I>>)*________ - [@80,'I',@80,'I',R]
@80: __<(I<<)>>*________ - [@80,')',@80,')',R]
@80: __<(I)<<*>>________ - [@80,'*',@80,'>',H]
@80: __<(I)<<>>>________ - [@80,'*',@80,'>',H]
```

References

Church, A. (1936). An unsolvable problem of elementary number theory. *American journal of mathematics*, **58**(2), 345–363.

Clarke, T. J. W., Gladstone, P. J. S., MacLean, C. D., & Norman, A. C. (1980). SKIM - The S, K, I reduction machine. *Pages 128–135 of: Proceedings of the 1980 ACM Conference on LISP and Functional Programming*.

Cockshott, P., & Michaelson, G. (2007). Are there new models of computation: a reply to Wegner and Eberbach. *Computer journal*, **50**(2), 232–247.

Curry, H. B. (1930). Grundlagen der Kombinatorischen Logik [Foundations of combinatorial logic]. *American journal of mathematics*, **52**(3), 509–536.

Curry, H. B., Feys, R., & Craig, W. (1958). *Combinatory Logic: Volume I*. North-Holland.

Curry, H. B., Hindley, J. R., & Seldin, J. P. (1972). *Combinatory Logic: Volume II*. North-Holland.

Darlington, J., & Reeve, M. (1981). ALICE: a multi-processor reduction machine for the parallel evaluation of applicative languages. *Pages 65–76 of: Proceedings of the 1981 ACM Conference on Functional Programming Languages and Computer Architecture*.

Davidson, J. (2016). *An Information Theoretic Approach to the Expressiveness of Programming Languages*. Ph.D. thesis, University of Glasgow.

Elgot, C., & Robinson, A. (1964). Random-Access Stored-Program Machines, an Approach to Programming Languages. *Journal of the association for computing machinery*, 365–399.

Fitch, F. B. (1974). *Elements of Combinatory Logic*. Yale University Press.

Gödel, K. (1962). *On Formally Undecidable Propositions of Principia Mathematica and Related Systems, Translated by B. Meltzer*. Oliver and Boyd.

Hankin, C. L., Osmon, P. E., & Shute, M. J. (1985). Cobweb: A combinator reduction architecture. *Pages 99–112 of: Functional Programming Languages and Computer Architecture*. LNCS 201. Springer.

Hilbert, D. (1998). Die Grundlagen Der Elementaren Zahlentheorie, Mathematische Annalen, 104, 485-494, 1921, Translated by W. Ewald as 'The Grounding of Elementary Number Theory'. *Pages 266–273 of:* Mancosu, P. (ed), *From Brouwer to Hilbert: The debate on the foundations of mathematics in the 1920s*. Oxford University Press.

Hindley, J. R., Lercher, B., & Seldin, J. P. (1972). *Introduction to Combinatory Logic*. London Mathematical Society Lecture Note Series 7, Cambridge University Press.

Hughes, R. J. M. (1982). Super-combinators: a new implementation method for applicative languages. *Pages 1–10 of: Proceedings of the 1982 ACM symposium on LISP and Functional Programming*.

Johnsson, T. (1985). Lambda lifting: Transforming programs to recursive equations. *ACM Conference on Functional Programming Languages and Computer Architecture.*

Kleene, S. J. (1936). Lambda definability and recursiveness. *Duke mathematical journal*, **2**, 340–353.

Kleene, S. J. (1952). *Introduction to Meta-Mathematics*. North-Holland.

Minsky, M. (1972). *Computation: Finite and Infinite Machines*. Prentice-Hall.

Naur, P. (1993). Understanding Turing's Universal Machine - Personal Style in Program Description. *Computer journal*, **36**(4), 351–372.

Neary, T. (2008). *Small Universal Turing Machines*. Ph.D. thesis, National University of Ireland Maynooth.

Nöcker, E. G. J. M. H., Plasmeijer, M. J., & Smetserss, J. W. E. (1991). The parallel ABC machine. *Pages 383–407 of:* Glaser, H., & Harteli, P. (eds), *Proceedings of 3rd International Workshop on Implementation of Functional Languages on Parallel Architectures*. University of Southampton Technical Report 91-07.

Peyton Jones, S. (1987). *The Implementation of Functional Programming Languages*. Prentice-Hall.

Peyton Jones, S. L., Clack, C., Salkild, J., & Hardie, M. (1987). GRIP: A high-performance architecture for parallel graph reduction. *Pages 98–112 of: Proceeding of Conference on Functional Programming Languages and Computer Architecture*. Springer.

Sale, A. H. J. (1989). The Architecture of the PCM-1. *Australian computer journal*, **21**, 71–78.

Schönfinkel, M. (1967). Über die Bausteine der mathematischen Logik, translated as 'On the Building Blocks of Mathematical Logic'. van Heijenoort, Je. (ed), *From Frege to Gödel: a source book in mathematical logic, 1879-1931*. Harvard University Press.

Turing, A. M. (1937a). Computability and λ-Definability. *Journal of symbolic logic*, **2**(4), 153–163.

Turing, A. M. (1937b). On Computable Numbers with an Application to the Entscheidungsproblem. *Proceedings of london mathematical society*, **s2-42**, 230–265.

Turner, D. A. (1979a). Another Algorithm for Bracket Abstraction. *Journal of symbolic logic*, **44**(2).

Turner, D. A. (1979b). *SASL Language Manual 1976, (Revised August 1979 for "Combinators" Version)*. Tech. rept. Computer Laboratory, University of Kent.

Turner, D. A. (1981). *Aspects of the implementation of programming languages: the compilation of an applicative language to combinatory logic*. Ph.D. thesis, University of Oxford.