

Duplication and Universality in the Circle Calculus

Flyxion

Independent Researcher

October 2026

Abstract

The circle calculus of SpherePOP evaluates nested circles by popping a circle whose contents are values. In the pure fragment, where every operator consumes values and returns a value, each pop removes exactly one circle, so evaluation terminates, independent pops commute, and the set of evaluation orders is a classical combinatorial object. This essay asks what changes when a single new circle is admitted, a binary application circle whose operands need not be values, together with the two contraction rules of combinatory logic. The answer has three separable parts, and each is established separately. First, the size measure that proves termination fails: one rule deletes circles and the other can create arbitrarily many, because it duplicates an operand. Second, an explicit infinite reduction exists, for a term of thirteen circles, and the behaviour of that term depends on the reduction strategy. Third, counter machines are compiled to lambda terms and translated to circles by bracket abstraction, with an exact accounting of the designated translation steps. The halting direction is proved here; the converse, and hence the universality conclusion for this compilation, remains conditional on the operational adequacy hypotheses stated in Section 9. Along the way the essay proves confluence by the parallel-reduction method, separates confluence (agreement when results exist) from the ability of a strategy to find those results, reads the pure fragment and the extended calculus as dynamic programs over reduction graphs, and states precisely how the representation relates to Lisp. A final section treats construction traces, in which a decomposition such as $48 \times 52 = (50 - 2)(50 + 2) = 50^2 - 2^2$ is recorded together with its estimate and its correction. Every quantitative statement in the text is generated by an executable companion from measured data, and the essay distinguishes throughout what is proved here, what is cited, what is machine-checked within stated bounds, and what is not claimed.

Contents

1	Introduction, claims, and the status of each claim	3
1.1	What this changes in the book	3
1.2	Claims	4
1.3	How the evidence is organized	4
1.4	Reading guide	5
2	The pure fragment	5
2.1	Counting evaluation orders	6
3	The application circle	7
3.1	Strategies	7
4	The size measure and why it fails	8

5	Nontermination and the dependence on strategy	9
5.1	One step of W	9
5.2	Leftmost-outermost reduction of Ω	9
5.3	Other strategies cycle	10
5.4	Growth without a cycle	10
6	Confluence, uniqueness of results, and the difference between agreement and discovery	11
6.1	Parallel reduction	11
6.2	What the machine checked	13
6.3	The one-step diamond fails	13
6.4	Agreement is not discovery	13
7	Bracket abstraction: the exact theorem	14
7.1	Size of the translation	16
7.2	What was checked	16
8	The lambda calculus as a source language	17
8.1	Reduction under a binder is not simulated	18
8.2	How often the failure occurs	18
9	Universality	19
9.1	Counter machines	19
9.2	Compilation to lambda terms	19
9.3	The simulation theorem	20
9.4	The non-halting direction	21
9.5	What universality does and does not establish	22
10	Relationship to Lisp	22
10.1	The restricted fragment	22
10.2	What is added or absent	23
10.3	What is not claimed	24
11	The dynamic-programming reading	24
11.1	The reduction graph and its Bellman recursions	24
11.2	The pure fragment: orderings against states	25
11.3	Sharing is memoization	25
11.4	Interval programs over a flat expression	26
12	Construction traces: recording how a decomposition was chosen	27
12.1	A schema	27
12.2	The example of 48×52	27
12.3	Why the traces fit the circle calculus	28
12.4	What is claimed and what is not	28
13	The skeptic's ledger	29
13.1	What is not claimed	30
13.2	How to try to break it	30
A	The programs	31
B	Reproducing the numbers	31

1 Introduction, claims, and the status of each claim

SpherePOP is a notation in which a computation is a finite tree of nested circles. A circle carries an operator and contains operands, which are numbers or further circles. The single evaluation act is a pop: a circle all of whose operands are values is replaced by the value its operator computes from them. The introductory text on which this essay builds develops the pure fragment in detail, and an executable companion verifies its basic facts. The question here is deliberately narrower and harder than the question of whether the notation is pleasant to use. It is the question of what the notation can compute, and what has to be added to it before it can compute everything a Turing machine can.

The pure fragment has no internal mechanism for unbounded iteration: every pop removes one circle and never creates one, so a term with n circles admits at most n pops. For closed terms whose operator applications are all admissible, every maximal reduction reaches a value in exactly n pops. This statement assumes that primitive operators are computable and terminate on admissible inputs; counting a primitive invocation as one pop does not bound its internal cost. The extension studied below changes this operational structure by adding a binary *application circle* $\bullet(f, x)$ and two contraction rules. The second rule duplicates an operand and can increase circle count. This is a concrete route beyond the original size argument, not a claim that duplication is necessary for every universal model. Reading $\bullet$ as juxtaposition and associating to the left,

$$Kxy \rightarrow x, \quad Sxyz \rightarrow xz(yz).$$

The first rule deletes the circles of y . The second contains z twice on its right side and once on its left, which is the duplication. These are the combinators S and K of Schönfinkel and of Curry [15, 4], and the calculus they generate is combinatory logic. The choice of S and K as the foundation, rather than lambda abstraction, is a choice about where to place the burden of proof. Combinatory logic has no bound variables, so there is no substitution to define, no capture to avoid, and no question about reduction under a binder. Every circle in a term is either an atom or an application, and every contraction is a local rewrite of a pattern whose shape is fixed. Lambda binding is then treated afterwards, as a source language that is translated into circles, so that its relationship to the circle calculus becomes a theorem about a translation and not an assumption about a primitive.

1.1 What this changes in the book

One departure from the introductory text must be flagged, because it is the point at which a careful reader would otherwise object. In the pure fragment an operator is applied only to values: a circle is popped when its operands are numbers, and evaluation proceeds from the leaves inward. The application circle as used here accepts arbitrary terms as operands. In Kxy the operand y may be a circle that has not been popped and, if it were popped first, might never finish. The rule $Kxy \rightarrow x$ discards y unexamined. This is the difference between a call-by-value discipline, in which operands are reduced before the operator is applied, and a call-by-name discipline, in which they are not. It is not an incidental detail. It is precisely what makes the calculus sensitive to the order in which circles are popped, and Section 5 shows that the same term can have a value under one order and none under another. The pure fragment is the sub-calculus in which this sensitivity is absent, and Section 2 records exactly why.

1.2 Claims

The essay makes the following claims, and it keeps them apart because they are logically independent and are established by different means.

- (A) *Failure of the size measure under duplication* (Section 4). Contracting a K redex changes the number of circles by exactly $-(2 + |y|)$ and contracting an S redex changes it by exactly $+|z|$. No function of circle count alone strictly decreases at every contraction, and the termination argument of the pure fragment does not transfer.
- (B) *An explicit infinite reduction* (Section 5). The term WW with $W = SII$ and $I = SKK$ has six circles in W and an infinite reduction. Under leftmost-outermost reduction the exact sequence is computed in closed form and the term grows without bound. Under three other strategies it cycles with period five.
- (C) *Confluence, and its separation from strategy* (Section 6). Reduction is Church–Rosser, proved here by the parallel-reduction method, so a term has at most one normal form. This does not say that every strategy finds that normal form, and the term $Ka\Omega$ shows that it does not.
- (D) *Exact translation from lambda terms* (Sections 7 and 8). Bracket abstraction satisfies $([x]M)N \rightarrow M[x := N]$ with a step count that is computed exactly from the structure of M . It follows that weak β reduction is simulated step for step, and also that reduction under a binder is *not* simulated, which is shown by an explicit pair of distinct normal forms.
- (E) *Conditional universality* (Section 9). A halting counter-machine computation is translated to a circle reduction yielding the correct Scott numeral. The converse requires two additional operational adequacy hypotheses: divergence of the compiled program under weak-head evaluation, and reflection of that divergence by bracket abstraction. These are stated explicitly and are not proved here; finite budget exhaustion does not establish them.
- (F) *A dynamic-programming reading* (Section 11). Reduction graphs satisfy Bellman recursions for shortest and longest reduction and for path counts, the pure fragment’s pop orderings are counted by the hook length formula and by an ideal-counting recursion, sharing is memoization, and the number of ways to group a flat expression is computed by an interval recurrence that sums to Catalan numbers.
- (G) *Construction traces* (Section 12). A reduction can be accompanied by a record of why a decomposition was chosen, and the obligations of that decomposition can be stated and checked exactly.

1.3 How the evidence is organized

Each statement in the essay carries one of four statuses, and the status is part of the statement. A result is *proved here* when a proof appears in the text. It is *cited* when it is a published theorem whose proof is not reproduced, and the reference is given. It is *machine-checked within stated bounds* when an executable companion verified it on an exhaustive enumeration or a stated random corpus, in which case the bound is part of the claim and the check is evidence and not proof. And it is *not claimed* when it is something a reader might expect and the essay does not assert. Section 13 collects these into a single table. The companion is a small program in JavaScript with an independent reference implementation in Python that shares no code and no data structures with it. All numbers

appearing in the tables and in the running text are generated from the measured output of that program, so that no figure in the essay was typed by hand.

The scale of the checking is as follows. The reduction engine was exercised on every combinator word with up to six leaves over $\{S, K, x\}$, on every word with seven or eight leaves over $\{S, K\}$, and on a corpus of random words, and it was compared with the independent implementation on 724,860 strategy-by-term comparisons, of which 17 ended in a cycle and 66 and 287 in a step or size budget, so that the comparison is not confined to terminating cases. Bracket abstraction was verified, contraction by contraction, on 471,550 instances. The reader who distrusts any of this is invited to run the suite, to raise its bounds, and to read the oracle. The bounds are printed in the title of each test.

1.4 Reading guide

Sections 2 to 4 establish the pure fragment, the extension, and the size laws. Section 5 is about nontermination and strategy. Section 6 proves confluence. Sections 7 to 9 carry out the route to universality. Section 10 states the relationship to Lisp, and Section 11 gives the dynamic-programming reading. Section 12 treats construction traces, and Section 13 is the ledger.

2 The pure fragment

For the exact pop counts and ordering formulas in this section, restrict attention to closed numerical terms over deterministic, total computable operators. Thus every tuple of numerical operands is admissible for its operator. In the broader fragment of the introduction, operators may be partial and an inert circle may remain: every reduction still has at most $|t|$ pops, but reaching a numeral in exactly that many pops requires admissibility throughout. A ready circle is eligible for a pop only when its operand tuple is admissible.

Definition 2.1. A *pure term* is defined inductively: a numeral is a pure term, and if o is an operator of arity k and $t_1, \dots, t_k$ are pure terms then the circle $o(t_1, \dots, t_k)$ is a pure term. A circle is *ready* when every operand is a numeral. A *pop* replaces a ready circle by the numeral that its operator computes from its operands. The *size* $|t|$ is the number of circles in t .

Proposition 2.2 (Termination in exactly $|t|$ pops). *Every pop strictly decreases $|t|$ by exactly one. Consequently every maximal sequence of pops from a pure term t ends, after exactly $|t|$ pops, in a numeral, and the numeral does not depend on the order of pops.*

Proof. A pop removes one circle, the ready one, and leaves every other circle in place, so the size drops by one. A term with no circles is a numeral, and a term with a circle has a ready circle (take a circle of greatest depth, all of whose operands are then numerals), so the process continues until size zero. For the independence of the order, suppose two distinct circles c_1 and c_2 are ready in t . They are disjoint, since a ready circle has only numerals inside it, and popping one does not alter the other or its operands, so the two pops commute and yield the same term. A standard argument on the number of pops (the diamond property for a terminating system with commuting local steps, Newman's lemma in its simplest form) then gives that all maximal pop sequences end in the same numeral. $\square$

The proposition establishes termination for this fragment, and the extension of Section 3 is judged against it. Values and predicates can depend on numerical inputs, but the

fixed expression supplies no internal unbounded iteration or search. Pop counts measure invocations of the chosen primitives, not their internal arithmetic cost.

2.1 Counting evaluation orders

Although the result of a pure term is fixed, the set of ways to reach it is not. A *pop sequence* is a total order on the circles of t in which every circle occurs after all circles in its subtree. These are the linear extensions of the tree order in which a parent is above its children. For a rooted forest the number of linear extensions has a closed form, due to Knuth [9] (see also [16]), which is the hook length formula for trees.

Theorem 2.3 (Hook length for pop orderings). *Let t be a pure term with $n = |t|$ circles and let $h(v)$ denote the number of circles in the subtree rooted at the circle v . The number of pop sequences of t is*

$$\frac{n!}{\prod_v h(v)}.$$

Proof. Consider a uniformly random total order of the n circles. For a fixed circle v , the probability that v occurs after every other circle of its subtree is $1/h(v)$, because among the $h(v)$ circles of the subtree each is equally likely to come last. These events, over the different circles v , are independent: the event for v depends only on the relative order of the circles in the subtree of v , and conditioned on the relative order inside each of the subtrees of the children of v , the position of v among those circles is uniform and independent of the internal orders. By induction on the tree the joint probability that every circle follows its subtree is $\prod_v 1/h(v)$. Multiplying by $n!$ total orders gives the count of valid ones. $\square$

A second count is the number of distinct *states* reachable by pops, that is, the number of distinct partial results, which is the number of down-closed sets of circles (sets closed under taking subtrees).

Proposition 2.4 (States as order ideals). *Let $I(v)$ be the number of sets of circles in the subtree of v that are closed under taking descendants. Then $I(v) = 1 + \prod_c I(c)$, where c ranges over the circle children of v , and the empty product is 1.*

Proof. A closed set either omits v , in which case it is an arbitrary choice of a closed set in each child subtree, giving $\prod_c I(c)$ possibilities, or contains v , in which case closure forces it to contain the whole subtree of v , giving one possibility. $\square$

Example 2.5. For the balanced tree with seven circles (a root, two children, each with two circle children), h takes the values 7, 3, 3, 1, 1, 1, 1 so Theorem 2.3 gives $7!/63 = 80$ pop sequences. The recursion of Proposition 2.4 gives $I = 2$ for a bottom circle, $I = 1 + 2 \cdot 2 = 5$ for a middle circle, and $I = 1 + 5 \cdot 5 = 26$ for the root. Both counts were verified by memoized dynamic programming over states, against the closed form, on 225 random trees.

The two counts illustrate a contrast that returns in Section 11. The number of paths through the reduction graph is the larger quantity (80 here), and the number of states is the smaller (26 here). A dynamic program over states computes a quantity indexed by paths in time proportional to the number of states, and the saving, which is exponential in general, is precisely the saving that sharing provides in graph reduction.

3 The application circle

Definition 3.1 (Circle terms). Fix an infinite set of variables. The set of *circle terms* is the least set containing the constants **S** and **K**, every variable, and, whenever f and x are circle terms, the *application circle* $\bullet(f, x)$. Application is written by juxtaposition and associates to the left, so fxy abbreviates $\bullet(\bullet(f, x), y)$. The size $|t|$ is the number of application circles in t , so atoms have size 0. Write **I** for **S****K****K**.

Definition 3.2 (Contraction). A *redex* is a subterm of the form Kxy or $Sxyz$ for arbitrary terms x, y, z . Its *contractum* is x , respectively $xz(yz)$. Replacing a redex occurrence by its contractum is a *contraction*, written $t \rightarrow t'$, and $\rightarrow$ is its reflexive transitive closure. A term with no redex is a *normal form*.

Three remarks fix what is and is not being assumed.

Remark 3.3 (Operands need not be values). There is no requirement that x, y or z be normal, and this is the departure from the book noted in the introduction. In the pure fragment an operator waits for values. Here Kxy may be contracted although y contains redexes of its own, and $Sxyz$ may be contracted although z is an unevaluated circle that the contraction will copy. A reader who prefers to keep the book's discipline can ask for a value-restricted variant in which contraction is allowed only when x, y and z are normal. That variant is the call-by-value strategy and is analysed below as one of the four strategies; it is a restriction of the order of contractions, not a different set of rules.

Remark 3.4 (Fixed arity). Because **K** needs two arguments and **S** needs three, the “operator” of an application circle is not statically known. In fx the function position f is itself an arbitrary term that may reduce to Ka , to Sab or to a variable. This is the sense in which the calculus has computed function position, a feature the pure fragment lacks and Lisp possesses; Section 10 returns to it.

Remark 3.5 (Interning). In the implementation, terms are hash-consed: two syntactically equal terms are the same object. Reduction is unaffected, since terms are immutable. The effect is that a term has two sizes, the number of circles in its tree reading $|t|$, and the number of distinct nodes in its directed acyclic graph. The second is the memory cost of a sharing implementation and appears in the tables of Section 9 and Section 11.

3.1 Strategies

A term in general has several redex occurrences, and a *strategy* chooses one. Four sequential strategies are used throughout. Let a redex be *outermost* if it is not properly contained in another redex, and *innermost* if it contains no other redex. Leftmost and rightmost refer to the position of the head symbol, reading the term as a string.

name	choice
lo	leftmost outermost
li	leftmost innermost
ri	rightmost innermost
ro	rightmost outermost

In addition, the *Gross-Knuth* step contracts every redex occurrence simultaneously, in the sense defined in Section 6. The leftmost-outermost redex of a term is also the redex found by head reduction when the term is a redex in head position, and is the one that an abstract machine finds by unwinding the spine; the equivalence is exploited by the machine of Section 11.

4 The size measure and why it fails

The termination argument of the pure fragment is a measure, a function from terms to the natural numbers that decreases at every step. The extension must be examined to see whether the same function, circle count, still decreases. It does not, and the way in which it fails is exact.

Proposition 4.1 (Size laws). *Let $t \rightarrow t'$ by contracting a redex occurrence.*

- (i) *If the redex is $Kxy \rightarrow x$, then $|t'| - |t| = -(2 + |y|)$.*
- (ii) *If the redex is $Sxyz \rightarrow xz(yz)$, then $|t'| - |t| = +|z|$.*

Proof. The redex $Kxy = \bullet(\bullet(K, x), y)$ has $2 + |x| + |y|$ circles and its contractum has $|x|$. The redex $Sxyz$ has $3 + |x| + |y| + |z|$ circles and its contractum $\bullet(\bullet(x, z), \bullet(y, z))$ has $3 + |x| + |y| + 2|z|$. All other circles of t are untouched, since a contraction replaces one subterm by another. Subtracting gives the stated differences. $\square$

Corollary 4.2 (No circle-count measure). *Circle count is not monotone along reduction, and no function of circle count alone strictly decreases at every step. A contraction of a K redex always decreases the count, and a contraction of an S redex increases it whenever $|z| > 0$, that is, whenever the third argument is not an atom.*

Proof. The size laws show both decreasing and increasing contractions. For the stronger claim about any function of circle count, take distinct atoms a, b, c . The contraction $Sabc \rightarrow ac(bc)$ has three circles on each side. Every measure of the form $f(|t|)$ therefore has equal values on its endpoints and cannot strictly decrease at every contraction. $\square$

Example 4.3. The term $Sab(Kcd)$ has 5 circles and the single contraction at the root gives $a(Kcd)(b(Kcd))$ with 7 circles, since $z = Kcd$ has two circles and appears twice afterwards. In the exhaustive survey the laws were checked on every redex occurrence in every term up to the stated bound, 20,475 occurrences in all, of which 14,940 were K redexes and 5,535 were S redexes, and 1,539 of the contractions were strictly growing.

Corollary 4.2 excludes the particular proof of termination used for the pure fragment, and nothing more. It does not by itself show that any term fails to terminate. To show that, an explicit infinite reduction is needed, and that is the subject of the next section. The two statements are logically independent, and it is worth saying so plainly because the first is sometimes presented as if it were the second. A calculus can have no decreasing measure of one kind and still be terminating, by a different measure, as the simply typed lambda calculus is. The failure of circle count is a failure of one argument, and the existence of Ω is the failure of the conclusion.

The two laws also locate where the power comes from. The K rule erases, and erasure is what allows a strategy to discard a divergent argument. The S rule duplicates, and duplication is what allows an argument to be consumed more than once. A calculus with erasure alone has the size measure and terminates. Duplication without erasure is the situation of the λI -calculus of Church, in which a term that has a normal form is strongly normalizing; that comparison is mentioned only to show that the strategy-dependence examined next is a consequence of erasure and duplication occurring together, and it is not pursued further.

5 Nontermination and the dependence on strategy

Write $W = SII$ and $\Omega = WW$. The term W has six circles, because $I = SKK$ has two and SII adds two more, and Ω has thirteen. The behaviour of Ω is the standard example of nontermination, and the point of studying it here is that its behaviour depends on how the circles are popped, in a way that can be stated exactly.

5.1 One step of W

The term Wx is a redex of the S rule, since $SIIx \rightarrow Ix(Ix)$. This is the self-application: it applies x to Ix , which is a copy of x in disguise. If $x = W$, then the right side is $IW(IW)$, and if the I is discarded the result is $W(IW)$, which is Ω except that its argument is still wrapped in an identity.

Lemma 5.1 (Contracting an identity). *For every term a and b , the term Iab reduces in exactly two leftmost-outermost steps to ab : first $SKKa \rightarrow Ka(Ka)$, and then $Ka(Ka)b \rightarrow ab$. Similarly $Ia \rightarrow a$ in two steps.*

Proof. Direct. In $Iab = SKKa b$ the head redex is $SKKa$, with $x = K$, $y = K$, $z = a$, giving $Ka(Ka)$ applied to b . The leftmost-outermost redex of that term is $Ka(Ka)$, whose contractum is a . $\square$

5.2 Leftmost-outermost reduction of Ω

Let $I^n W$ denote I applied n times to W , with $I^0 W = W$ and $I^{n+1} W = I(I^n W)$. Then $\Omega = W(I^0 W)$.

Lemma 5.2 (The exact leftmost-outermost sequence). *For every $n \geq 0$, the term $W(I^n W)$ reduces under leftmost-outermost reduction in exactly $2n + 3$ steps to $W(I^{n+1} W)$, and has $13 + 3n$ circles. The term Ω has therefore an infinite leftmost-outermost reduction, in which the sizes of the terms $W(I^n W)$ are strictly increasing, so that no term repeats.*

Proof. Write $x = I^n W$. The term $Wx = SIIx$ is itself a redex, and it is both leftmost and outermost. Its contraction gives $Ix(Ix)$ in one step. By Lemma 5.1 the head of this term, Ix applied to the argument Ix , reduces in two further steps to $x(Ix)$. Now $x = I^n W = I(I^{n-1} W)$ when $n \geq 1$, so this term is $I(I^{n-1} W)(Ix)$, which by the lemma reduces in two steps to $I^{n-1} W(Ix)$. Repeating, each of the n outer identities is removed at the cost of two steps, and after $2n$ steps the term is $W(Ix) = W(I^{n+1} W)$. The total is $1 + 2 + 2n = 2n + 3$ steps. In each of these steps the redex that was contracted was the head redex, hence outermost and leftmost, since no redex occurs to its left. For the size, W has six circles and each application of I adds three circles (the application circle together with the two circles of I), so $|I^n W| = 6 + 3n$ and $|W(I^n W)| = 1 + 6 + (6 + 3n) = 13 + 3n$. $\square$

n	leftmost-outermost steps	circles
0	3	13
1	5	16
2	7	19
3	9	22
4	11	25
5	13	28

The measured values reproduce the closed forms: the rows of the table above are produced by the engine, and for $n = 0, 1, \dots, 5$ they match $2n + 3$ and $13 + 3n$. The sequence is also an instance of a more general fact about the pure and impure calculus. Under leftmost-outermost reduction the term grows and never repeats, so a simple cycle detector that watches for a repeated term would report no cycle and would run forever, which is the correct behaviour, and not a defect of the detector. Reaching $W(I^{40}W)$ takes $\sum_{n < 40} (2n + 3) = 1,680$ leftmost-outermost steps, and the engine's step budget reports the run as unfinished and not as cyclic.

5.3 Other strategies cycle

A different choice of redex gives a different, and in this case periodic, behaviour. Under leftmost-innermost, rightmost-innermost and rightmost-outermost reduction, the engine finds that Ω returns to itself after five contractions, and the cycle detector reports period five for each of the three. The path followed by leftmost-innermost reduction is

```
S I I (S I I)
-> S K K (S I I) (S K K (S I I))
-> K (S I I) (K (S I I)) (S K K (S I I))
-> S I I (S K K (S I I))
-> S I I (K (S I I) (K (S I I)))
-> S I I (S I I)
```

where I is displayed unexpanded in the first and last terms. The path is as recorded by the engine. In it, after the first contraction, the left I has been expanded to SKK and is applied to W twice over, and the next three contractions evaluate the arguments to the form $W(IW)$ and then to WW once more. The comparison of all four strategies on Ω and on $Ka\Omega$ is given in the table below.

strategy	Ω	$Ka\Omega$
lo	no normal form in 400 steps	normal form in 1 step
li	cycle, period 5	cycle, period 5
ri	cycle, period 5	cycle, period 5
ro	cycle, period 5	normal form in 1 step

The row for $Ka\Omega$ is the cleanest demonstration of strategy-dependence. Under the two outermost strategies the redex $Ka\Omega$ is contracted first, discards Ω unexamined, and yields the normal form a in one step. Under the two innermost strategies the argument Ω is reduced first, and cycles. The same term therefore has a normal form that one strategy finds in a single step and another never finds. This is the exact sense in which the unrestricted application circle is call-by-name, and in which the value-restricted discipline of the pure fragment, which pops operands first, becomes unsafe: it would apply the innermost rule to a term the outermost rule handles at once.

5.4 Growth without a cycle

One more example shows that nontermination need not even be periodic. Let $M = S(SII)I$. Then, for every term x ,

$$Mx \rightarrow SIIx(Ix) \rightarrow xx,$$

since $SIIx \rightarrow xx$ after the identities are contracted and $Ix \rightarrow x$. Applying this with $x = M$ gives $MM \rightarrow MMM$. If M^k denotes the left-nested application of k copies of M , the same

equation applied to the leftmost two copies gives $M^k \rightarrow M^{k+1}$ for every $k \geq 2$, so MM has an infinite reduction in which the number of circles is unbounded. Whether the particular strategies of the engine follow this reduction is a matter of measurement and not of proof, and the engine reports it: starting at a term with 21 circles and 9 distinct nodes, the leftmost-outermost reduction of MM does not repeat in 600 steps, and has grown, by that step, to 1,329 circles in 57 distinct nodes. The table gives intermediate values.

step	circles	distinct nodes
0	21	9
100	294	27
200	488	35
300	659	41
400	884	47
500	1,014	51
600	1,329	57

The comparison of the two size columns repays attention. The tree grows roughly linearly in the number of steps while the number of distinct nodes grows much more slowly, which is the effect of sharing: the same subterm is copied many times, and a representation that shares copies pays for it once. The consequence is taken up in Section 11, where the cost of reduction is separated into the number of steps and the size of the term at each step.

6 Confluence, uniqueness of results, and the difference between agreement and discovery

Two questions are easily conflated. The first is whether two different orders of popping can lead to different results. The second is whether a given order of popping is guaranteed to reach a result when one exists. The first is a statement about agreement and the second is a statement about discovery, and the calculus answers them differently. Agreement holds, by the Church–Rosser theorem proved below. Discovery depends on the strategy, as the examples of Section 5 have already shown, and a positive answer for one strategy is a cited theorem and not a result proved here.

6.1 Parallel reduction

The proof follows the method of Tait and Martin-Löf as presented by Takahashi [17], adapted to the two rules of the circle calculus.

Definition 6.1 (Parallel reduction). The relation $t \Rightarrow t'$ is the least relation satisfying

- (P1) $a \Rightarrow a$ for every atom a (S, K, or a variable);
- (P2) if $f \Rightarrow f'$ and $x \Rightarrow x'$ then $f x \Rightarrow f' x'$;
- (P3) if $x \Rightarrow x'$ then $K x y \Rightarrow x'$, for every term y ;
- (P4) if $x \Rightarrow x'$, $y \Rightarrow y'$ and $z \Rightarrow z'$ then $S x y z \Rightarrow x' z' (y' z')$.

Rule (P2) applies to every application circle, including those that are redexes, so that $\Rightarrow$ is reflexive on all terms (by induction on the term). Rules (P3) and (P4) contract a redex at the root while allowing its operands to be reduced simultaneously.

Definition 6.2 (Complete development). The *Gross–Knuth development* $\text{gk}(t)$ contracts every redex occurrence of t that is present in t , each operand reduced by development as well: $\text{gk}(a) = a$ for an atom; $\text{gk}(Kxy) = \text{gk}(x)$; $\text{gk}(Sxyz) = \text{gk}(x)\text{gk}(z)(\text{gk}(y)\text{gk}(z))$; and $\text{gk}(fx) = \text{gk}(f)\text{gk}(x)$ when fx is not a root redex.

Lemma 6.3 (Inclusions). (a) If $t \rightarrow t'$ then $t \Rightarrow t'$. (b) If $t \Rightarrow t'$ then $t \twoheadrightarrow t'$. Hence the reflexive transitive closure of $\Rightarrow$ equals $\twoheadrightarrow$.

Proof. (a) By induction on the position of the contracted redex. At the root use (P3) or (P4) with the operands reduced by reflexivity. Below the root use (P2) with reflexivity on the untouched side. (b) By induction on the derivation of $t \Rightarrow t'$. For (P1) there is nothing to prove, and for (P2) reduce the left part and then the right part inside the application. For (P3), contract the root redex $Kxy \rightarrow x$ and then reduce x to x' . For (P4), contract $Sxyz \rightarrow xz(yz)$ and then reduce x to x' , y to y' , and each of the two copies of z to z' in place. The last claim follows because $\twoheadrightarrow \subseteq \Rightarrow^* \subseteq \twoheadrightarrow$. $\square$

Lemma 6.4 (Triangle property). If $t \Rightarrow a$ then $a \Rightarrow \text{gk}(t)$.

Proof. By induction on the structure of t . If t is an atom then $a = t = \text{gk}(t)$. Suppose $t = fx$.

Case 1: t is not a root redex. Then the last rule of the derivation must be (P2), since (P3) and (P4) require a root redex. Hence $a = f'x'$ with $f \Rightarrow f'$ and $x \Rightarrow x'$. By the induction hypothesis, $f' \Rightarrow \text{gk}(f)$ and $x' \Rightarrow \text{gk}(x)$, and by (P2), $a \Rightarrow \text{gk}(f)\text{gk}(x) = \text{gk}(t)$. (This holds even when $f'x'$ is itself a root redex, which can happen when an operand of t reduces to a redex-forming head.)

Case 2: $t = Kuv$. If the derivation ends with (P3), then $a = u'$ with $u \Rightarrow u'$, and the induction hypothesis gives $a \Rightarrow \text{gk}(u) = \text{gk}(t)$. If it ends with (P2), then $a = f'v'$ with $f = Ku \Rightarrow f'$ and $v \Rightarrow v'$. Because Ku is not a redex and K is an atom, the only way to derive $Ku \Rightarrow f'$ is (P2), so $f' = Ku'$ with $u \Rightarrow u'$. Then $a = Ku'v'$, the induction hypothesis gives $u' \Rightarrow \text{gk}(u)$, and (P3) gives $a \Rightarrow \text{gk}(u) = \text{gk}(t)$.

Case 3: $t = Suvw$. If the derivation ends with (P4), then $a = u'w'(v'w')$ and the induction hypothesis with (P2) gives $a \Rightarrow \text{gk}(u)\text{gk}(w)(\text{gk}(v)\text{gk}(w)) = \text{gk}(t)$. If it ends with (P2), then as before $a = Su'v'w'$ with $u \Rightarrow u'$, $v \Rightarrow v'$, $w \Rightarrow w'$, because Suv and Su are not redexes, and (P4) together with the induction hypothesis gives $a \Rightarrow \text{gk}(t)$. $\square$

The triangle property is stronger than the diamond property: it exhibits a *single* common reduct, $\text{gk}(t)$, that works for every parallel reduct simultaneously.

Theorem 6.5 (Church–Rosser). If $t \twoheadrightarrow a$ and $t \twoheadrightarrow b$, then there is a term c with $a \twoheadrightarrow c$ and $b \twoheadrightarrow c$.

Proof. By Lemma 6.4, if $t \Rightarrow a$ and $t \Rightarrow b$ then $a \Rightarrow \text{gk}(t)$ and $b \Rightarrow \text{gk}(t)$, so $\Rightarrow$ has the diamond property. The standard strip argument lifts the diamond property from a relation to its reflexive transitive closure: by induction on the lengths of the two chains, any two $\Rightarrow^*$ chains from t can be completed. By Lemma 6.3 the closure $\Rightarrow^*$ is $\twoheadrightarrow$. $\square$

Corollary 6.6 (Uniqueness of normal forms). A term has at most one normal form. If $t \twoheadrightarrow n_1$ and $t \twoheadrightarrow n_2$ with n_1 and n_2 normal, then $n_1 = n_2$.

Proof. By Theorem 6.5 there is c with $n_1 \twoheadrightarrow c$ and $n_2 \twoheadrightarrow c$, and a normal form reduces only to itself. $\square$

Why the proof is short. The argument uses nothing about the rules beyond two properties: each left side is a linear pattern (no variable occurs twice on the left of a rule), and no left side overlaps another or itself at a non-root position. The left sides $Kxy = \bullet(\bullet(K, x), y)$ and $Sxyz$ contain no proper subterm of the form of a left side, since Kx , Sx and Sxy are not redexes. Rewriting systems with these two properties are called *orthogonal*, and confluence of orthogonal systems is a theorem of Rosen [14] and Huet [6], of which Theorem 6.5 is a special case proved directly. The same reasoning would fail if a rule were non-linear, for instance a rule that compared two arguments for equality, and it is worth recording because the duplication in the S rule happens on the *right* side, where it does no harm to confluence.

6.2 What the machine checked

The triangle property was verified exhaustively on every term with at most five leaves over $\{S, K, x\}$: 3,873 terms, with 5,655 parallel reducts altogether, of which 1,710 are nontrivial in the sense that they differ from the term. Every term in this range has a complete reduction graph, and in each of the 3,873 graphs exactly one normal form is reachable, as Corollary 6.6 predicts. These are checks of a proved theorem and not independent evidence for it.

6.3 The one-step diamond fails

It is instructive to see why parallel reduction is used. The one-step relation $\rightarrow$ does not have the diamond property, that is, there are terms t with $t \rightarrow a$ and $t \rightarrow b$ but no c with $a \rightarrow c$ and $b \rightarrow c$ in a single step each. The smallest examples have six leaves, and none exists with five or fewer. One of them is

$$t = S S S (K S S), \quad a = S (K S S) (S (K S S)), \quad b = S S S S.$$

Contracting the root of t duplicates the argument KSS and gives a , which contains two copies of that redex. Contracting the inner redex first gives b , which has a single redex at the root, and its only contractum is $SS(SS)$, a normal form. To close the square from a one must contract *both* copies of KSS , which is two steps and not one. The number of terms with six leaves for which the diamond fails is 81 in the exhaustive enumeration. Parallel reduction repairs exactly this: it allows all the copies produced by a duplication to be reduced together.

6.4 Agreement is not discovery

Corollary 6.6 says that no two strategies can disagree about a normal form when both find one. It says nothing about whether they find one. Three observations from the earlier sections and from the survey make this concrete.

First, the term $Ka\Omega$ has the normal form a , and leftmost-outermost and rightmost-outermost reduction find it in one step while leftmost-innermost and rightmost-innermost reduction cycle (Section 5). Second, the strategies differ in the *length* of the path even when they all succeed. Among all strongly normalizing terms in the survey (all terms up to six leaves over $\{S, K, x\}$ together with all seven- and eight-leaf terms over $\{S, K\}$, 161,094 terms in all), the number of steps taken was compared with the true minimum, which is the shortest path in the reduction graph, computed by dynamic programming (Section 11).

strategy	optimal	strictly worse
lo	158,357 (98.3%)	2,737
li	136,439 (84.7%)	24,655
ri	136,439 (84.7%)	24,655
ro	157,149 (97.6%)	3,945

The leftmost-outermost strategy was never shorter than the minimum, which is a tautology, and it attained the minimum on 98.3% of the terms but not on all of them: for $S S S (K S S)$, the term already seen above, it takes 3 steps where 2 suffice, because contracting the root first duplicates a redex that the inner-first order would have contracted once. The leftmost-innermost and rightmost-innermost strategies have identical counts in this survey, and are strictly worse than the minimum on 24,655 terms. For the reverse case, the term $K S (K S S)$ takes 2 steps under leftmost-innermost and 1 under any outermost strategy.

Third, and most important, is the question of whether a strategy can *miss* a result entirely. For leftmost-outermost reduction the answer is given by a theorem that is cited and not reproved.

Theorem 6.7 (Leftmost normalization; Curry and Feys [4], O’Donnell [13], Huet and Lévy [7]; see also [8]). *If a term of combinatory logic has a normal form, then leftmost-outermost reduction reaches it in finitely many steps.*

The result is a standardization theorem for combinatory logic in Curry and Feys, and was extended to the left-normal orthogonal term rewriting systems by O’Donnell, whose condition (every constant of a left side occurs before every variable when the left side is written in prefix notation) is satisfied by the applicative form $\bullet(\bullet(K, x), y)$ and $\bullet(\bullet(\bullet(S, x), y), z)$. Huet and Lévy give the general theory of needed redexes of which it is an instance. The theorem is the reason why leftmost-outermost reduction is used as the *definition* of “the circle term halts” in Section 9.

The machine checked a finite consequence. Over 130,593 terms (every word up to five leaves over $\{S, K, x\}$ and every word of seven and eight leaves over $\{S, K\}$, with a budget of 3,000 steps and 50,000 circles), leftmost-outermost reduction reached a normal form on 130,521 terms. On none of the rest did any of the four strategies reach a normal form within the budget, and on no term did another strategy reach a normal form that leftmost-outermost reduction failed to reach. No two strategies reached different normal forms. The strategies that missed a result found by leftmost-outermost reduction within the budget were leftmost-innermost or rightmost-innermost on 17 terms and rightmost-outermost on 2 terms. These are failures within a budget and not proofs of divergence, and the check on words of at most six leaves, on which all words normalize, is vacuous for this question; this is the reason the survey was extended to seven and eight leaves, where the first words without a normal form occur. The two smallest such words, each of seven leaves, are $S S S (S S) S S$ and $S (S S) S S S S$; under all four strategies their size exceeds the budget.

7 Bracket abstraction: the exact theorem

The translation of Section 8 rests on one construction, which turns a term with a distinguished variable into a term in which that variable has been abstracted. It is the combinatory counterpart of $\lambda x.M$ (see [4, 18] for the algorithm and its variants), and it is the only place where the circle calculus simulates binding. A feature of the approach followed here is that

the central statement is proved by structural induction with an *exact* step count, not by a uniform bound per β step. A uniform bound would be false, because the cost of simulating one β step depends on the structure of the body, as the cubic growth measured at the end of the section shows.

Definition 7.1 (Substitution and bracket abstraction). For circle terms M and N and a variable x , the term $M[x := N]$ is obtained by replacing every occurrence of x in M by N . Since circle terms have no binders this is plain syntactic replacement. The *bracket abstraction* $[x]M$ is defined by

$$[x]M = \mathsf{K} M \quad (x \notin \mathsf{FV}(M)), \quad [x]x = \mathsf{I}, \quad [x](PQ) = \mathsf{S}([x]P)([x]Q) \quad (x \in \mathsf{FV}(PQ)).$$

Here $\mathsf{FV}(M)$ is the set of variables occurring in M . The η -optimized abstraction $[x]_\eta M$ adds the clause $[x]_\eta(Px) = P$ whenever $x \notin \mathsf{FV}(P)$, applied before the third clause.

Definition 7.2 (Designated length). Define $\mathsf{steps}(M, x)$ by $\mathsf{steps}(M, x) = 1$ if $x \notin \mathsf{FV}(M)$; $\mathsf{steps}(x, x) = 2$; and $\mathsf{steps}(PQ, x) = 1 + \mathsf{steps}(P, x) + \mathsf{steps}(Q, x)$ if $x \in \mathsf{FV}(PQ)$.

Theorem 7.3 (Bracket abstraction). *For every term M , variable x and term N , the term $([x]M)N$ reduces to $M[x := N]$ by a reduction of exactly $\mathsf{steps}(M, x)$ contractions, each of which contracts a redex whose position is determined by the structure of M .*

Proof. By induction on M , following the three clauses.

Case $x \notin \mathsf{FV}(M)$. Then $([x]M)N = \mathsf{K} MN$, a redex, whose contractum is $M = M[x := N]$. This is one contraction.

Case $M = x$. Then $([x]x)N = \mathsf{I}N = \mathsf{S}KKN \rightarrow \mathsf{K}N(\mathsf{K}N) \rightarrow N = x[x := N]$, two contractions.

Case $M = PQ$ with $x \in \mathsf{FV}(M)$. Then $([x]M)N = \mathsf{S}([x]P)([x]Q)N$, and a single contraction at the root gives $([x]P)N([x]Q)N$. By the induction hypothesis the left factor $([x]P)N$ reduces to $P[x := N]$ in $\mathsf{steps}(P, x)$ contractions. These contractions take place inside the left factor, which is a proper subterm of the term at hand; they are legitimate contractions of that term because a contraction of a subterm is a contraction of the whole, there being no binder anywhere that could obstruct it. The right factor then reduces to $Q[x := N]$ in $\mathsf{steps}(Q, x)$ contractions in the same way. The result is $P[x := N]Q[x := N] = (PQ)[x := N]$ and the total is $1 + \mathsf{steps}(P, x) + \mathsf{steps}(Q, x) = \mathsf{steps}(M, x)$. $\square$

Remark 7.4 (Why a per-step bound would be wrong). The proof says that the number of contractions needed to simulate the replacement of x by N is a function of M alone and does not depend on N . It grows with the number of occurrences of x in M and with the depth of the structure containing them. Any bound that was uniform over all β steps would have to bound $\mathsf{steps}(M, x)$ over all bodies M , and there is no such bound. The cost is also unrelated to the size of the argument N as long as the reduction is performed in the designated order; the argument is copied by the S rule but not reduced.

Remark 7.5 (The reduction is not leftmost-outermost). The designated reduction contracts the root first, then the contractions within the left factor, then those within the right. This is a valid reduction sequence and is leftmost in the left factor, but it is not in general the leftmost-outermost sequence for the whole term, since the redexes that the right factor contains remain unreduced throughout the left phase. Theorem 7.3 is a statement that a reduction exists with a given length and a given endpoint. By confluence (Theorem 6.5), any other reduction from $([x]M)N$ to a normal form reaches the same normal form as $M[x := N]$ does, whenever one exists.

Theorem 7.6 (The η variant). *The conclusion of Theorem 7.3 also holds for $[x]_\eta$, with a number of contractions that never exceeds $\text{steps}(M, x)$.*

Proof. The three cases above are unchanged, except that in the recursive case the subterms are abstracted by $[x]_\eta$ and the induction hypothesis gives at most $\text{steps}(P, x)$ and $\text{steps}(Q, x)$ contractions respectively. The new clause applies to $M = P x$ with $x \notin \text{FV}(P)$, and then $([x]_\eta M) N = P N = (P x)[x := N]$ with zero contractions, while $\text{steps}(P x, x) = 1 + 1 + 2$ is positive. $\square$

7.1 Size of the translation

Proposition 7.7 (Linear growth of one abstraction). *Let $\ell(M)$ denote the number of leaves (atoms) of M . Then $\ell([x]M) \leq 4\ell(M) - 1$.*

Proof. By induction on M . For an atom x the left side is $\ell(1) = 3 = 4 \cdot 1 - 1$. For an atom other than x it is $\ell(K M) = 2 \leq 3$. If $x \notin \text{FV}(M)$ then $\ell(K M) = 1 + \ell(M) \leq 4\ell(M) - 1$ because $\ell(M) \geq 1$. For $M = P Q$ with $x \in \text{FV}(M)$, $\ell(S([x]P)([x]Q)) = 1 + \ell([x]P) + \ell([x]Q) \leq 1 + (4\ell(P) - 1) + (4\ell(Q) - 1) = 4\ell(M) - 1$. $\square$

A single abstraction therefore costs a constant factor, but a body with n nested abstractions applies the construction n times and can produce a result whose size is polynomial in n of degree higher than one. The extreme case is the body $v_0 v_1 \cdots v_{n-1}$, abstracted over v_{n-1} first and v_0 last. For that family the engine measured the number of leaves for $n = 1, 2, \dots, 16$ and found, in every case, the value

$$\frac{n^3 + 6n^2 + 8n - 6}{3},$$

and the machine check asserts this formula for each of these sixteen values. The formula is a cubic, and the check is evidence for the cubic and not a proof of it; the reader who wishes to prove it can do so by solving the recurrence that Proposition 7.7 suggests. The measured values are given below, with the η -optimized abstraction, which collapses the entire term to 1, three leaves, since $\lambda v_0 \dots v_{n-1}. v_0 v_1 \cdots v_{n-1}$ is η -equivalent to the identity.

n	plain leaves	eta leaves
2	14	3
4	62	3
6	158	3
8	318	3
10	558	3
12	894	3

7.2 What was checked

Theorem 7.3 was verified, contraction by contraction, on 471,550 pairs of a body and an argument term, drawn from 94,310 bodies. In each instance the engine produced the derivation as an explicit log of positions, validated at each position that the rule claimed was in fact the rule applicable there, checked that the length of the log was $\text{steps}(M, x)$, and checked that the final term was $M[x := N]$. In all, 2,801,660 contractions were verified in this way. For the η variant, which is a different construction with the same conclusion, the check is of reachability only, and it was run on 1,428 terms, among which the η variant produced a strictly smaller term than the plain algorithm in 441 cases.

8 The lambda calculus as a source language

Lambda terms are built from variables, application, and abstraction $\lambda x.M$, which binds x in M . Terms are identified up to renaming of bound variables, written $\equiv_\alpha$, and $M[x := N]$ denotes capture-avoiding substitution. The β rule is $(\lambda x.M) N \rightarrow M[x := N]$. A *weak* step is a β step whose redex does not lie inside any abstraction. Weak reduction is the notion of evaluation of a functional programming language, and, as this section shows, it is exactly what the circle calculus simulates.

Definition 8.1 (Translation). The translation $\mathcal{T}$ from lambda terms to circle terms is $\mathcal{T}(x) = x$, $\mathcal{T}(MN) = \mathcal{T}(M)\mathcal{T}(N)$, and $\mathcal{T}(\lambda x.M) = [x]\mathcal{T}(M)$, using the plain bracket abstraction of Definition 7.1. Since the abstraction operation erases the bound variable, $\text{FV}(\mathcal{T}(M))$ equals the set of free variables of M .

Lemma 8.2 (Invariance under renaming). *If $M \equiv_\alpha M'$ then $\mathcal{T}(M) = \mathcal{T}(M')$ as circle terms.*

Proof. By induction on the structure of M . Variables and applications are immediate. If $M = \lambda x.P$ and $M' = \lambda x'.P'$ with $P' \equiv_\alpha P[x := x']$, then in $[x]\mathcal{T}(P)$ the variable x occurs only to determine, at each node, whether the K, I or S clause applies, and the same pattern of clauses is selected for x' in $\mathcal{T}(P')$ by the induction hypothesis. The two results are then equal term by term. $\square$

Lemma 8.3 (Abstraction commutes with substitution). *Let $y \neq x$ and $y \notin \text{FV}(N)$. For every circle term P , $([y]P)[x := N] = [y](P[x := N])$.*

Proof. By induction on P . If $y \notin \text{FV}(P)$ then $y \notin \text{FV}(P[x := N])$ either, because the substitution introduces only variables of N , so both sides are K($P[x := N]$). If $P = y$ then both sides are I, since x does not occur in I and $y \neq x$ gives $y[x := N] = y$. If $P = P_1 P_2$ with $y \in \text{FV}(P)$, then $y \in \text{FV}(P[x := N])$, because the occurrences of y in P are not touched, and both sides equal S($[y]P_1[x := N]$ ($[y]P_2[x := N]$) and S($[y](P_1[x := N])$ ($[y](P_2[x := N])$) respectively, which agree by the induction hypothesis. $\square$

Theorem 8.4 (Substitution lemma). *For all lambda terms M and N , $\mathcal{T}(M[x := N]) = \mathcal{T}(M)[x := \mathcal{T}(N)]$.*

Proof. By induction on M , after renaming bound variables (which does not change the translation, by Lemma 8.2) so that every bound variable of M is distinct from x and from the free variables of N . For a variable the claim is the definition of substitution. For an application it follows from the induction hypothesis. For $M = \lambda y.P$ with $y \neq x$ and $y \notin \text{FV}(N)$,

$$\mathcal{T}((\lambda y.P)[x := N]) = [y]\mathcal{T}(P[x := N]) = [y](\mathcal{T}(P)[x := \mathcal{T}(N)]) = ([y]\mathcal{T}(P))[x := \mathcal{T}(N)],$$

where the second equality is the induction hypothesis and the third is Lemma 8.3, applicable because $\text{FV}(\mathcal{T}(N)) = \text{FV}(N) \not\ni y$. $\square$

Theorem 8.5 (Weak β steps are simulated). *If $M \rightarrow M'$ by contracting a β redex that lies inside no abstraction, then $\mathcal{T}(M) \twoheadrightarrow \mathcal{T}(M')$. If the redex is $(\lambda x.P)Q$, the number of contractions is exactly $\text{steps}(\mathcal{T}(P), x)$, they are given by the designated reduction of Theorem 7.3, and they all take place inside the image of the redex.*

Proof. Because the redex is inside no abstraction, $M = C[(\lambda x.P) Q]$ for a context C built from applications only, and $\mathcal{T}(M) = \mathcal{T}(C)[\mathcal{T}((\lambda x.P) Q)]$, where the translation of a context made of applications is the context of the translations. The translation of the redex is $([x]\mathcal{T}(P)) \mathcal{T}(Q)$, which reduces by Theorem 7.3 in steps $(\mathcal{T}(P), x)$ contractions to $\mathcal{T}(P)[x := \mathcal{T}(Q)]$, and this equals $\mathcal{T}(P[x := Q])$ by Theorem 8.4. Replacing the redex by its reduct inside C gives $\mathcal{T}(M')$. $\square$

What the machine checked. The substitution lemma was verified on all lambda terms up to a size bound over two variables, on 2,984 substitutions. Theorem 8.5 was verified constructively, as an explicit contraction sequence, on 872 weak β steps in the same universe, with 1,648 contractions in all: for each step the engine wrote out the designated contractions, applied them inside the translated term at the position of the translated redex, and compared the result with the translation of the contractum.

8.1 Reduction under a binder is not simulated

The restriction to weak steps in Theorem 8.5 is not a limitation of the proof. It is a fact about the translation, and it can be exhibited with one example.

Proposition 8.6 (The ξ rule fails). *Let $M = \lambda x.(\lambda y.y) x$ and $M' = \lambda x.x$. Then $M \rightarrow_{\beta} M'$ by a β step under a binder, but $\mathcal{T}(M) = \mathbf{S}(\mathbf{K}\mathbf{I})\mathbf{I}$ and $\mathcal{T}(M') = \mathbf{I}$ are distinct normal forms, so $\mathcal{T}(M)$ does not reduce to $\mathcal{T}(M')$, nor the reverse, nor do the two have a common reduct.*

Proof. Compute $\mathcal{T}((\lambda y.y) x) = \mathbf{I}x$, so $\mathcal{T}(M) = [x](\mathbf{I}x)$. Since $x \in \text{FV}(\mathbf{I}x)$ and the term is an application, $[x](\mathbf{I}x) = \mathbf{S}([x]\mathbf{I})([x]x) = \mathbf{S}(\mathbf{K}\mathbf{I})\mathbf{I}$, because $x \notin \text{FV}(\mathbf{I})$. This term has the head $\mathbf{S}$ applied to two arguments only, so it is not a redex, and neither of the arguments $\mathbf{K}\mathbf{I}$ and $\mathbf{I}$ contains a redex ($\mathbf{K}\mathbf{I}$ has one argument and $\mathbf{I} = \mathbf{S}\mathbf{K}\mathbf{K}$ has two). So $\mathcal{T}(M)$ is normal. The term $\mathcal{T}(M') = \mathbf{I}$ is normal for the same reason. They are distinct as circle terms, and two distinct normal forms have no common reduct, since a normal form reduces only to itself. $\square$

The two terms are equal in a weaker sense that the circle calculus does not see by reduction alone. Applied to any term a , both reduce to a : $\mathbf{S}(\mathbf{K}\mathbf{I})\mathbf{I}a \rightarrow \mathbf{K}\mathbf{I}a(\mathbf{I}a) \rightarrow \mathbf{I}(\mathbf{I}a) \rightarrow a$ and $\mathbf{I}a \rightarrow a$. The identification of functions that agree on all arguments is the rule of *extensionality*, and it is known that weak combinatory equality together with extensionality is equivalent to $\beta\eta$ -equality for the lambda calculus [4, 5, 1]. This is the correct statement of the relationship between the two calculi: $\mathcal{T}$ preserves weak evaluation exactly and preserves reduction under binders only up to extensionality. The distinction is immaterial for the programs studied in Section 9, which are closed terms applied to arguments, evaluated by weak reduction, and decoded by applying the result to two fresh variables, but it is material for any claim that the circle calculus *is* the lambda calculus, and no such claim is made.

8.2 How often the failure occurs

For a small survey, the lambda terms of at most seven nodes over two variables were examined whose only β redexes lie under binders, and the first normal-order step (which does reduce under a binder) was compared with the translation. Of the 656 such terms, 256 had a translation that already reduced or joined to that of the result. For the remaining 400 terms the two translations had no common reduct as they stand, and applying both to as many fresh variables as the term has leading abstractions made them join in 380 cases. In the final 20 cases no such application helps, and in every one of them the two translations

are distinct normal forms (20 of 20). The cases printed by the engine all have the shape $x (\lambda y. (\lambda z. z) y) \rightarrow x (\lambda y. y)$, a head variable applied to an abstraction that contains the redex, so that no application of fresh variables to the whole term exposes it. They are the generic situation of Proposition 8.6, and they agree with each other only extensionally. The survey does not measure how likely this situation is in realistic programs. It shows only that it occurs in 20 of the 656 small terms examined, and that for the rest a suitable number of fresh arguments suffices.

9 Universality

The route to universality follows the chain of translations that the preceding sections prepared. A counter machine is compiled to a lambda term. The lambda term is translated to a circle term by bracket abstraction. Weak evaluation of the lambda term is simulated, contraction by contraction, by reduction of the circle term, and the output is read from the resulting term by applying it to two fresh variables. Each link is either proved here, cited, or checked, and the three kinds of link are marked.

9.1 Counter machines

A *counter machine* with k registers holds natural numbers $r_0, \dots, r_{k-1}$ and has a finite program of numbered instructions, each of one of three forms: *inc i j* increments register i and continues at instruction j ; *dec i j j'* decrements register i and continues at j if the register was positive, and otherwise leaves it and continues at j' ; and *halt* stops. The machine starts at instruction 0 with the inputs in the first registers and zeros elsewhere, and when it halts the output is the content of register 0. A step is the execution of one instruction.

Theorem 9.1 (Minsky [11]). *The functions computed by counter machines in this sense are exactly the partial recursive functions. Already two registers suffice, given a suitable encoding of the input and output.*

This is cited. It is the point at which the essay depends on classical computability theory, and the dependence is intended: the theorem converts the question of universality into the question of whether the circle calculus can simulate counter machines. For the purposes of this essay the number of registers is arbitrary and the compiler is uniform in it.

9.2 Compilation to lambda terms

Numbers are represented by *Scott numerals*, $\bar{0} = \lambda z t. z$ and $\overline{n+1} = \lambda z t. t \bar{n}$, so that a numeral selects between its two arguments according to whether it is zero, and passes the predecessor to the second argument in the successor case. The successor function is $\text{succ} = \lambda n z t. t n$, and $\text{succ } \bar{n}$ weakly reduces to $\overline{n+1}$ in one step. Scott numerals (see [12]) are used in place of the more familiar Church numerals because the predecessor is a constant-time operation on them, which keeps the compiled terms small. The compiler uses a fixed-point combinator that is sound for call-by-name evaluation, $Y = \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$.

For a program with m instructions the compiler builds the term

$$F = \lambda f s r_0 \cdots r_{k-1}. \text{dispatch}_0(s), \quad \text{dispatch}_i(s_i) = s_i B_i (\lambda p_i. \text{dispatch}_{i+1}(p_i)),$$

where the last dispatch continues with a halting body, and the body B_i of instruction i is as follows. For *inc r j* it is $f \bar{j} r_0 \cdots (\text{succ } r_r) \cdots r_{k-1}$. For *dec r j j'* it is

$r_r (f \bar{j}' r_0 \cdots r_{k-1}) (\lambda q. f \bar{j} r_0 \cdots q \cdots r_{k-1})$. For halt it is r_0 . The compiled program applied to inputs $n_0, \dots$ is

$$\text{Init} = (Y F) \bar{0} \bar{n}_0 \cdots \bar{n}_{k-1},$$

and its image in the circle calculus is $\mathcal{T}(\text{Init})$. The state of the machine is the first argument, a Scott numeral that the dispatch chain consumes one predecessor at a time until it reaches zero, at which point the body B_i is selected. All the numerals are closed terms, so $\mathcal{T}(\text{Init})$ is a closed circle term.

9.3 The simulation theorem

Say that a lambda term a represents n if $a \rightarrow_w^* \bar{n}$ by weak steps, and write $P = U U$ with $U = \lambda x. F(x x)$. Then $Y F \rightarrow_w P$ and $P \rightarrow_w F P$, since $U U \rightarrow_w F(U U)$.

Lemma 9.2 (One machine step). *Let a_0 represent the number of an instruction i and let $a_1, \dots, a_k$ represent the registers $r_0, \dots, r_{k-1}$ of a configuration, and let the machine move in one step to the configuration (j, r') . Then $P a_0 a_1 \cdots a_k \rightarrow_w^+ P b_0 b_1 \cdots b_k$ for terms $b_0, \dots, b_k$ representing j and $r'_0, \dots, r'_{k-1}$.*

Proof. The weak step $P \rightarrow_w F P$ and $k + 2$ further contractions instantiate $f := P$ and the arguments in the body of F , giving $\text{dispatch}_0(a_0)$ with the arguments substituted. Reducing the argument a_0 weakly to $\bar{i}$, which is allowed since an argument is not under an abstraction, each skipped instruction costs a fixed number of steps: $\bar{n} + \bar{1} B N \rightarrow_w^2 N \bar{n}$ and $N \bar{n}$ contracts to the next dispatch with $p := \bar{n}$. After i such stages the dispatch is $\bar{0} B_i N_i \rightarrow_w^2 B_i$ with the substitutions in place. Now case on the instruction. For inc the body is a term $P \bar{j} \cdots$ with one register replaced by $\text{succ } a_r$, and $\text{succ } a_r \rightarrow_w^* \text{succ } \bar{n} \rightarrow_w \bar{n} + \bar{1}$ when a_r represents n , so the new register represents $n + 1$. For dec, if a_r represents 0 then $a_r Z S \rightarrow_w^* \bar{0} Z S \rightarrow_w^2 Z$ where Z is the zero branch, and the term is $P \bar{j}' a_1 \cdots a_k$; if a_r represents $n + 1$ then $a_r Z S \rightarrow_w^* \bar{n} + \bar{1} Z S \rightarrow_w^2 S \bar{n} \rightarrow_w P \bar{j} \cdots \bar{n} \cdots$. In every case the resulting term has the claimed form, with the numbers of the new configuration. $\square$

Theorem 9.3 (Halting direction). *Suppose the machine started on inputs $n_0, \dots, n_{j-1}$ halts after m steps with $r_0 = n$. Then $\text{Init} \rightarrow_w^* a$ for a term a representing n , and consequently $\mathcal{T}(\text{Init}) \rightarrow \mathcal{T}(a) \rightarrow \mathcal{T}(\bar{n})$. Applying $\mathcal{T}(\bar{n})$ to two distinct variables z and s gives a reduction to z if $n = 0$ and to $s u$ for a term u such that $\mathcal{T}(\bar{n} - \bar{1})$ is the normal form of u otherwise.*

Proof. The initial configuration is represented by the numerals themselves, with $Y F \rightarrow_w P$ giving the shape $P \bar{0} \cdots$. By induction on m using Lemma 9.2, the term Init weakly reduces to a term $P b_0 \cdots b_k$ representing the configuration after m steps. At that point b_0 represents the number of a halting instruction, the same weak reductions as in the proof of the lemma apply, and the halting body r_0 is reached, which represents n . The step from lambda terms to circle terms is Theorem 8.5, applied to each weak contraction in the sequence; all the contractions in the proof are weak, since none lies under an abstraction. The last sentence is the computation $\mathcal{T}(\bar{n} + \bar{1}) z s \rightarrow s \mathcal{T}(\bar{n})$, which is Theorem 8.5 applied to $\bar{n} + \bar{1} z s \rightarrow_w^2 s \bar{n}$, together with uniqueness of normal forms (Corollary 6.6). $\square$

Checked. The halting direction was verified end to end on 56 runs of the five programs listed in Appendix A, over a range of inputs including the boundary values. For each run four independent evaluators agreed on the output: a direct interpreter of the machine, the independent Python implementation, weak β reduction of the compiled lambda term, and leftmost-outermost reduction of its circle translation, decoded by repeated application to two fresh variables. The table gives representative step counts, where the circle count is the

number of contractions performed by the abstract machine, summed over the layers of the decoding.

program	input	machine	lambda	circle	output
add	3,2	5	77	750	5
add	3,3	7	100	1,022	6
double	3	23	403	5,141	6
monus	3,2	5	76	779	1
monus	3,3	7	97	1,057	0
mult	3,2	53	1,235	27,796	6
mult	3,3	74	1,738	40,594	9
parity	3	5	74	596	1

The ratios are large, and honest accounting requires saying so. A step of the machine costs roughly ten to twenty lambda steps in these examples and roughly a hundred to five hundred circle contractions; compilation to S, K by bracket abstraction is not an efficient implementation, and no claim of efficiency is made. The sizes are also considerable, as the table below shows. The column of shared nodes counts the nodes of the directed acyclic graph of the interned term, which is the memory used by an implementation that shares identical subterms, and the last column is the size with the η -optimized abstraction.

program	states	registers	circles	shared nodes	circles (eta)
add	3	2	321	108	192
double	6	2	751	183	509
monus	4	2	404	125	243
mult	10	4	2,576	491	1,501
parity	4	1	348	99	244

9.4 The non-halting direction

The converse requires two additional operational statements. First, for this particular compiler, a non-halting machine must yield an infinite *weak-head* evaluation of the compiled lambda term. Lemma 9.2 constructs weak reductions simulating machine steps, but an infinite weak reduction alone does not establish divergence of a specified strategy: an argument may reduce forever even when a head contraction would discard it. Determinism of the source machine does not close that gap. Second, the chosen bracket-abstraction translation must reflect absence of a weak-head normal form on these compiled terms by absence of a circle normal form. The forward simulation theorem and circle confluence do not by themselves prove this reflection statement. Classical accounts of combinatory logic and standardization supply the relevant background [4, 5], but a precise theorem with hypotheses matched to this compiler and translation is not established here. Accordingly both statements are explicit adequacy hypotheses for the conditional corollary below, rather than consequences claimed from the finite experiments.

The evidence in the non-halting case is therefore of two kinds. First, for the program `loop`, which increments register 0 forever, the lambda term and the circle term were run under a large budget and neither produced a normal form, and the leftmost-outermost sequence of the circle term was extended for 20,000 steps, at each of which a redex was

present. This is a statement about a finite prefix, which no finite computation can strengthen into non-termination, and it is reported as such. Second, a program was constructed whose halting depends on its input: it counts register 0 down by two and halts exactly on even inputs. On the inputs 0, 2, 4, 6 the reference machine halts and the circle evaluation decodes a numeral, and on the inputs 1, 3, 5 the reference machine does not halt and the circle evaluation exhausts a budget of 200,000 contractions. This shows that the behaviour of the compiled term is not a property of the program alone, and again it is a statement about the inputs tried.

Corollary 9.4 (Undecidability for circle terms, conditional). *Assume, in addition to Theorem 9.1 and Theorem 9.3, that every non-halting machine yields a compiled lambda term with no weak-head normal form, and that the translation $\mathcal{T}$ maps each such compiled term to a circle term with no normal form. Then there is no algorithm deciding normal-form existence for closed circle terms. Under these hypotheses every partial recursive function has a circle representation whose defined inputs reduce to the translated output numeral and whose undefined inputs yield no normal form.*

The halting theorem gives a circle normal form for each halting computation, while the two adequacy hypotheses exclude one for each non-halting computation. A decision procedure for circle normal-form existence would therefore decide counter-machine halting. This is a conditional argument; the two operational hypotheses remain to be discharged for this compiler.

9.5 What universality does and does not establish

The application-circle rules are precisely the classical S,K rules, whose computational universality is a standard result [4, 5]. The specific compiler developed here has a proved halting simulation and a conditional non-halting argument. It supplies no efficiency theorem and does not make the pure value-returning fragment universal. Its contribution is an explicit operational extension, a translation with designated contraction counts, and executable evidence whose bounds remain attached to the claims.

10 Relationship to Lisp

The relationship between circles and Lisp is stated in one sentence and then qualified at length, because the sentence is true and the qualifications are where the content lies. *Circles and S-expressions are the same tree representation with different computational commitments.* A circle $o(t_1, \dots, t_k)$ and the S-expression $(o\ t_1\ \dots\ t_k)$ are the same finite ordered tree, with the operator in the first position and the operands following, and the pure fragment, which uses only a fixed table of operators applied to numerals and to further circles, is a restriction of the set of S-expressions. The application circle $\bullet(f, x)$ is the binary S-expression $(f\ x)$ and a left-associated chain $f\ x\ y\ z$ is the spine of the nested pairs of the same tree, which Lisp abbreviates as the list $(f\ x\ y\ z)$ by the convention of currying.

10.1 The restricted fragment

On the pure fragment the correspondence extends from the representation to the evaluation. Let eval be the evaluator of a Lisp with applicative-order evaluation of operands and with a fixed table of primitive arithmetic procedures, and restrict it to S-expressions built from numerals and symbols in the table. Evaluating $(o\ t_1\ \dots\ t_k)$ evaluates each t_i and then applies o to the values, which is exactly a sequence of pops, in the order chosen by the

evaluator, and the result is the numeral that the pops produce. By Proposition 2.2 the result does not depend on the order. This correspondence is a matter of inspection and was not machine-checked, and it is a statement about the restricted fragment only.

10.2 What is added or absent

Beyond this fragment the two systems diverge, and the divergences are substantive. They are listed below, together with where each appears in the circle calculus of this essay.

feature	Lisp	circle calculus
computed function position	the head of a form may be any expression, and evaluating it yields a procedure	present once the application circle is admitted; absent in the pure fragment, where the operator is a fixed symbol
binding	<code>lambda</code> , <code>let</code> and an environment of bindings	absent; simulated by bracket abstraction (Section 7), which erases names
recursion	named procedures, <code>define</code> , <code>labels</code> ; or fixed-point combinators	only through duplication, as in M and Ω and the compiled fixed point $Y F$
special forms	<code>if</code> , <code>cond</code> , <code>and</code> , <code>quote</code> : operands are not all evaluated	the K rule discards an operand unevaluated, and under the outermost strategies every operand is lazy, so strictness is a property of the strategy and not of the form
quotation	<code>quote</code> turns code into data and <code>eval</code> turns data into code	absent; terms are data only at the meta level, in the sense that the engine manipulates them
effects	assignment, input and output, mutation of shared structure	absent; reduction is a purely functional rewriting of immutable terms
default strategy	applicative order	none; confluence makes the result independent of strategy when it exists, and the strategy decides whether it is found

Several of these entries bear on how the two systems are compared. A Lisp program that avoids special forms, effects and quotation, and that is evaluated under normal order, is closer to the calculus of this essay than to Lisp as it is usually run, and the difference is not cosmetic. The special form `if` is the Lisp mechanism for obtaining the effect of the K rule, which discards the unevaluated branch: it is the place where applicative order is suspended by fiat. In the circle calculus there is no special form, because under outermost reduction no operand is evaluated until its value is needed, and the lazy behaviour that Lisp obtains by exception is the rule. The cost, which is the subject of Section 5, is that the evaluation order matters to termination, whereas in the restricted fragment it does not.

10.3 What is not claimed

McCarthy’s 1960 paper [10] shows that a small set of primitives, together with `lambda`, conditional expressions and recursion, suffices for a universal function `eval` written in the language itself. No SpherePOP counterpart of that construction is given here: the essay does not define a self-interpreter for circles, and it makes no claim about the existence of one beyond what follows from universality, namely that some circle term computes any partial recursive function, including the function that interprets an encoding of circle reduction. The claim of Section 9 is a statement about what functions can be computed, and says nothing about the clarity or economy of any particular program.

11 The dynamic-programming reading

A dynamic program is a recurrence over a space of subproblems in which the answer to a problem is assembled from answers to smaller problems that overlap. Reduction in the circle calculus supplies such a space in four distinct ways, each with its own recurrence, and the section takes them in turn: the graph of reductions, the pure fragment’s ideals, sharing as memoization, and the intervals of a flat expression.

11.1 The reduction graph and its Bellman recursions

Definition 11.1 (Reduction graph). For a term t , the *reduction graph* $G(t)$ has as vertices the terms reachable from t by $\rightarrow$ and as edges the single contractions. A vertex with no outgoing edge is a normal form. Since terms are interned, vertices are identified by the identity of the node.

The graph is finite for the terms studied in the exhaustive survey except for the explicit nonterminating cases, for which the exploration is cut off at a budget and reported as incomplete. When the graph is finite and acyclic, three quantities are defined by recurrences over successors. With $\text{succ}(u)$ the set of terms reachable by one contraction from u :

$$\begin{aligned} d(u) &= \begin{cases} 0 & u \text{ normal,} \\ 1 + \min_{v \in \text{succ}(u)} d(v) & \text{otherwise,} \end{cases} \\ \ell(u) &= \begin{cases} 0 & u \text{ normal,} \\ 1 + \max_{v \in \text{succ}(u)} \ell(v) & \text{otherwise,} \end{cases} \\ \pi(u) &= \begin{cases} 1 & u \text{ normal,} \\ \sum_{v \in \text{succ}(u)} \pi(v) & \text{otherwise.} \end{cases} \end{aligned}$$

Here $d(u)$ is the length of a shortest reduction to a normal form, $\ell(u)$ the length of a longest, and $\pi(u)$ the number of distinct reduction sequences. Each is an instance of Bellman’s principle of optimality [2]: a shortest (respectively longest) reduction from u begins with some contraction to a successor v and continues with a shortest (respectively longest) reduction from v , since any other continuation could be exchanged for it without lengthening (respectively shortening) the whole. Acyclicity makes the recursion well founded, and memoization over the vertices evaluates it in time proportional to the number of edges.

When cycles occur, as they do for Ω under the innermost strategies and may occur in the graph of a term under all strategies simultaneously, shortest distances can instead be computed by initializing normal forms at 0 and other vertices at ∞ , then propagating improvements backwards. For a finite explored graph with unit-cost edges this yields the

shortest-path distances; a vertex with no route to a normal form retains distance ∞ . Cycles alone do not justify a claim of multiple solutions. The engine computes it twice, by Bellman value iteration and by breadth-first search backwards from the normal forms along reversed edges, and requires agreement; the number of paths is reported as infinite when a cycle lies on a path to a normal form, and as a finite natural number, in arbitrary precision, otherwise. On 471 acyclic graphs (every acyclic complete graph among the terms of at most four leaves) the three recurrences were compared with brute-force enumeration of all paths and agreed.

Strategies as policies. In the language of dynamic programming a strategy is a *policy*, a rule that selects a successor at each vertex, and the table of Section 6 is a measurement of how far four particular policies fall from the optimal policy d . The optimal policy exists, depends on the whole graph, and is not computable from the local view of a term that a strategy has. The leftmost-outermost policy is optimal on 98.3% of the terms surveyed, the rightmost-outermost policy on 97.6%, and the two innermost policies on 84.7%. These figures describe the surveyed words, which are short and chosen exhaustively, and they should not be read as predicting the behaviour of large programs.

11.2 The pure fragment: orderings against states

Section 2 gave two counts for the same tree, the number of pop sequences from the hook length formula and the number of states from the ideal recursion. They are the two natural dynamic programs on the pure fragment. The first counts paths and the second counts vertices of the graph of partial evaluations, and the contrast between them, 80 against 26 for the balanced tree of seven circles, grows without bound with the size of the tree. A computation over the states can recover any quantity that is defined by a recurrence over paths, such as the number of paths itself, in time proportional to the number of states. This is the same compression that graph reduction achieves, and it is for the same reason: the subproblems overlap.

11.3 Sharing is memoization

In a tree representation a duplicated subterm is copied, and each copy is reduced separately. In a graph representation the duplicated subterm is a single node with two incoming edges, and a reduction of the node is performed once and seen by both. The engine represents terms by interned nodes, so every term is already a graph, and a reduction procedure that records the result of reducing each node, by the identity of the node, is exactly a memo table over subproblems. This is Wadsworth’s observation [19], and the engine implements two forms of it: a table of head normal forms indexed by node, and a table of normal forms indexed by node.

The first measurement compares the tree evaluator with the graph evaluator on terms that can be normalized by both. Over 34,491 terms with a normal form the two always agreed on the result, and the number of contractions performed with normal-form memoization was strictly smaller on 78 of them. A second measurement isolates the effect. Let $d = \lambda x. \text{plus } x x$ be doubling on Church numerals and let $d^n(1)$ be its n -fold iterate, translated to circles and applied to two variables to read off the numeral.

n	tree steps	shared steps	ratio	output circles
1	94	60	1.57	2
2	264	126	2.10	4
3	616	228	2.70	8
4	1,344	402	3.34	16
6	5,952	1,326	4.49	64
8	25,536	4,842	5.27	256
10	108,480	18,726	5.79	1,024
12	458,688	74,082	6.19	4,096

The measured contraction counts in both columns grow rapidly over the tested range. The explicit output of $d^n(1)$ is a unary numeral with 2^n application circles, so materializing that tree entails output work proportional to its size. This is not a lower bound of 2^n combinator contractions: one contraction may duplicate a large subterm, and contraction count differs from allocation, traversal and output-writing cost. The ratio of tree contractions to shared contractions increases monotonically over the measured range, from 1.6 at $n = 1$ to 6.2 at $n = 12$. These are measured ratios, not an asymptotic law or an evaluator-independent lower bound. Memoization avoids recomputing head normal forms already cached for a node; the reported costs belong to these particular evaluators and representations.

11.4 Interval programs over a flat expression

The last dynamic program is the oldest. A flat expression $a_0 \circ_1 a_1 \circ_2 \cdots \circ_n a_n$, written without parentheses, becomes a circle term only after a choice of structure, and the possible structures are the full binary trees with $n + 1$ leaves, which are the parenthesizations. The set $V(i, j)$ of values obtainable from the interval $a_i \cdots a_j$ satisfies

$$V(i, i) = \{a_i\}, \quad V(i, j) = \bigcup_{i \leq k < j} \{x \circ_{k+1} y : x \in V(i, k), y \in V(k+1, j)\},$$

the recurrence of matrix-chain multiplication, and the number of parenthesizations $C(i, j)$ obeys $C(i, i) = 1$ and $C(i, j) = \sum_k C(i, k) C(k+1, j)$, whose solution for $n + 1$ operands is the Catalan number $C_n = \frac{1}{n+1} \binom{2n}{n}$. The same recurrence counts the circle structures of a flat word of n leaves over an alphabet of a letters: it is $C_{n-1} a^n$, a closed form that agrees with the exhaustive enumeration for $n \leq 7$ over $\{\mathbf{S}, \mathbf{K}, x\}$, where the counts are 288,684 at seven leaves.

expression	values (number of parenthesizations)
$2 + 3 \times 4$	14 (1), 20 (1)
$2^{2^{2^2}}$ read freely	65536 (2), 256 (3)
$8 - 3 - 2$	7 (1), 3 (1)
Catalan numbers $C_1, \dots, C_8$	1, 2, 5, 14, 42, 132, 429, 1430

The first row is the familiar ambiguity of $2 + 3 \times 4$, in which the two groupings give 14 and 20. The notation of circles removes the ambiguity by making the structure part of the expression, so that the two values belong to two different circle terms. The second row shows that the interval program is a computation over distinct *values* and not only over structures: of the five ways of grouping a tower of four twos, three give 256 and two give

65536. The recurrence was verified against brute-force enumeration of all parenthesizations on 46 expressions and the counts agree with the Catalan numbers.

A remark on interpretation links the section to the next. The recurrence for V has the form of a decision problem: at each interval a split point k is chosen, and the value of the whole depends on the choice. A dynamic program does not make the choice for the user. It exhibits the space in which the choice is made and the cost, in subproblems, of exploring it. This is the sense in which the introduction of a circle, which is a record of a choice of structure, is already the first move of a construction trace.

12 Construction traces: recording how a decomposition was chosen

Everything so far has treated a circle term as something to be reduced. A circle term can also be read as a *record*, in the sense that its structure preserves a choice that could have been made otherwise, and the recurrence of the preceding section makes the sense precise: the parenthesization is the record of a choice of split. This section develops one consequence. A method of solving a problem, by identifying a part that can be handled, acting on it, and reorganizing what remains, can be written down as a sequence of circle terms together with the obligations that justify each move, and the record can be inspected, compared with alternatives, and checked.

12.1 A schema

The schema has four moves. *Split*: replace a problem by a combination of a part that is manageable and a remainder, in such a way that the combination equals the original. *Rewrite*: replace an expression by an equal one, licensed by an identity. *Estimate*: solve the manageable part, exactly or approximately, with a stated bound on the error if approximately. *Correct*: account for the remainder, either exactly, which restores exactness, or by a bound, which leaves the result as an interval. A *trace* is the list of these moves with, for each, the equation or inequality that licenses it and a status, discharged when the equation has been verified and pending when it has only been asserted. The invariant of a trace is that its value is the sum of its estimate and its corrections, and that the value is exact if and only if every correction is discharged.

Definition 12.1 (Trace entry). A *trace entry* is a triple $(\kappa, \varepsilon, \sigma)$ consisting of a kind $\kappa \in \{\text{split}, \text{rewrite}, \text{estimate}, \text{correct}\}$, an obligation ε (an equation or inequality between terms), and a status $\sigma \in \{\text{discharged}, \text{pending}\}$. A *trace* is a finite sequence of entries.

The definition is deliberately light. It does not say how a decomposition is found, which remains a matter of judgment, and it does not say that the sequence is a proof. What it provides is that the judgment is separated from the justification: the choice of a split is recorded as a choice, and the justification, which can be verified mechanically, is recorded as an obligation attached to it. The movement between exploratory judgment and exact justification is thereby made visible, and a reader can see at which entries the justification is complete and at which it is only promised.

12.2 The example of 48×52

Consider the product 48×52 . A decomposition that a person doing the calculation without aids would naturally try is to notice that both factors lie near 50. The trace is as follows.

1. *Split.* Write the factors as $50 - 2$ and $50 + 2$. The obligation is $48 = 50 - 2$ and $52 = 50 + 2$, which is discharged by arithmetic.
2. *Rewrite.* The obligation $(c - d)(c + d) = c^2 - d^2$ is the difference of squares, an identity of polynomials in c and d . It was verified by expanding both sides symbolically with integer coefficients, not by instance. Instantiating $c = 50$, $d = 2$ gives $48 \times 52 = 50^2 - 2^2$.
3. *Estimate.* Evaluate the manageable part, $50^2 = 2,500$.
4. *Correct.* Apply the correction $-2^2 = -4$ for the remainder, giving 2,496.

The trace thus has the estimate 2,500 and the exact correction -4 , and the sum is the product. In terms of circles the original problem is one multiplication circle with 1 circle, the split form $(50 - 2) \times (50 + 2)$ has 3 circles, and the squared form $50^2 - 2^2$ has 3 circles. The point of recording the circle counts is also to disclaim them: the number of pops is not a measure of difficulty, and the version with more pops is in this case the easier one for a human calculator, because the pops involved are on numbers that are easy to multiply. The cost model in which every pop is unit cost is inadequate for this purpose, and nothing in this section claims otherwise.

Proposition 12.2 (Exactness of centre-and-correct). *Let a and b be integers of the same parity, $m = (a + b)/2$ and $d = (b - a)/2$. Then $a = m - d$, $b = m + d$ and $ab = m^2 - d^2$. Moreover for any $\delta \geq |d|$ the estimate m^2 satisfies $m^2 - \delta^2 \leq ab \leq m^2$.*

Proof. Adding and subtracting give $m - d = a$ and $m + d = b$, so $ab = (m - d)(m + d) = m^2 - d^2$ by the polynomial identity. Since $0 \leq d^2 \leq \delta^2$ the inequalities follow. $\square$

The interval form is the reason an estimate can be useful before the correction is known: if only the bound δ on the offset is available, the product lies in an interval whose width is δ^2 , and the estimate is an upper bound. The pending status in a trace is the formal counterpart of this situation.

For products whose factors differ in parity, the same idea uses the polarization identity $4ab = (a + b)^2 - (a - b)^2$, valid for all integers; it was likewise verified symbolically. The polarization identity was checked numerically on every pair of integers from 1 to 99, and for the 4,901 pairs of equal parity in that range the centre-and-correct trace was constructed and every step verified as exact.

12.3 Why the traces fit the circle calculus

A circle term is a tree whose structure is a choice, and a trace is a sequence of such trees related by identities, so the representation already carries what a trace needs. A reduction, in the pure fragment, is a sequence of trees related by pops. A trace is a sequence of trees related by *decompositions*, which are rewrites in the opposite direction, from a term to an equal term of larger size, licensed by an identity. The circle calculus thus can express the two kinds of move that a method of solution needs, the reduction that simplifies and the decomposition that reorganizes, and can keep them in one record. The obligations are written in the same notation as the terms, and they are decidable in the cases above, by evaluation for instances and by symbolic expansion for identities.

12.4 What is claimed and what is not

The claim is representational and limited: that a trace with the stated entries can be recorded, that each obligation in the example is discharged by a verified computation, and that the record distinguishes exact steps from approximate ones. The essay does not

claim that the method is the best method for the product, that the method generalizes to problems whose decompositions have no polynomial identity behind them, or that recording a trace helps a person learn. The last is an empirical question. A test would present two groups with the same problems, one with traces and one without, and compare accuracy and transfer on problems that require a new decomposition. No such study has been run, and the section is offered as a notation in which such a study could be set up, with the strategies themselves stated explicitly enough that others can try them, vary them, and report where they fail.

13 The skeptic's ledger

The following table lists every substantive claim of the essay with its status. The purpose is to make it possible to disagree with a particular claim without disagreeing with all of them. The bounds in the third column are the bounds of the executable companion as of this writing, and a reader may raise them.

claim	status	basis
Total numerical pure fragment terminates in exactly $ t $ pops, independent of order	proved	Proposition 2.2
Pop orderings counted by the hook length formula; states by ideals	proved; checked	Theorem 2.3, Proposition 2.4; random trees, memoized dynamic program
Size laws: K changes size by $-(2 + y)$, S by $+ z $	proved; checked	Proposition 4.1; every redex occurrence in the survey
No circle-count measure decreases along reduction	proved	Corollary 4.2
Exact leftmost-outermost sequence of $W(I^n W)$ and its growth	proved; checked	Lemma 5.2; table
Ω cycles under three other strategies with period five	machine-checked	finite check; not stated as a theorem
$K a \Omega$ has a normal form under outermost and not under innermost reduction	proved by inspection; checked	Section 5
$M M$ has an infinite reduction with unbounded size	proved	$M^k \twoheadrightarrow M^{k+1}$
The engine's strategies follow that reduction without repeating	machine-checked	finite prefix only
Church–Rosser; unique normal forms	proved	Theorem 6.5, parallel reduction
Triangle property of complete developments	proved; checked	Lemma 6.4; every term of at most five leaves over $\{S, K, x\}$
One-step diamond fails; smallest failures have six leaves	machine-checked	exhaustive up to six leaves; witness exhibited
Leftmost-outermost reduction finds the normal form whenever one exists	cited	Curry and Feys; O'Donnell; Huet and Lévy
Finite consequence of the above on the 130,593-term corpus	machine-checked	no violation in budgeted runs, with no claim beyond the budget

claim	status	basis
Bracket abstraction theorem with exact step count	proved; checked	Theorem 7.3; contraction by contraction on every pair of the survey
Cubic growth of nested plain abstraction on $v_0 \cdots v_{n-1}$	machine-checked	asserted for $n = 1, \dots, 16$; not proved
Substitution lemma for $\mathcal{T}$	proved; checked	Theorem 8.4
Weak β steps simulated exactly	proved; checked	Theorem 8.5
Reduction under a binder is not simulated; $\mathcal{T}$ agrees with lambda calculus only up to extensionality	proved for the example; cited for equivalence	Proposition 8.6; classification survey
Counter machines are universal	cited	Minsky 1967
Halting machine yields the correct numeral in circles	proved; checked	Theorem 9.3; runs against three independent evaluators
Non-halting machine yields no normal form	conditional	requires compiler weak-head divergence and translation reflection; neither is proved here; finite budget evidence only
Undecidability of normal-form existence for circle terms	conditional	Corollary 9.4
Bellman recursions for shortest, longest and path counts	proved; checked	Section 11; brute-force comparison
Sharing as memoization reduces work	checked	measured ratios; mechanism stated, not an asymptotic law
Interval program counts Catalan numbers; flat words $C_{n-1}a^n$	proved; checked	recurrence; enumeration to seven leaves
Exactness of centre-and-correct decompositions	proved; checked	Proposition 12.2; symbolic identity and evaluation

13.1 What is not claimed

The essay does not claim that the four-event history kernel used elsewhere in the SpherePOP repository simulates lambda reduction, and no statement above is about that kernel. It does not claim that any particular non-halting program fails to terminate, beyond a budget exhausted and the argument of Section 9. It does not claim that the translation of Section 8 is an equivalence of calculi, since it is not, as Proposition 8.6 shows. It makes no claim about efficiency beyond the measured tables. It does not claim that construction traces help anyone learn, only that they can be written down and checked. And it does not claim that the pure fragment of the book is Turing complete: the classical application-circle extension is universal, while the compiler-specific converse remains conditional as stated above.

13.2 How to try to break it

A skeptic with an afternoon can do the following. Raise the leaf bound of the exhaustive survey from eight to nine and compare the verdicts of the strategies, noting that the interning table grows with the survey, and that enumerating all words of eight leaves over three atoms already exhausts the table of the JavaScript engine. Replace the compiler's fixed-point combinator with another and rerun the machine comparison. Add a program to the list and run it through the three evaluators. Read the independent Python implementation and

look for a place where it shares an assumption with the JavaScript. Run the suite with a perturbed rule, for instance with the **K** rule returning its second argument, and confirm that it fails; a verification that cannot fail is not one.

A The programs

The five programs run to completion are listed with their registers and the function computed on register 0. An instruction **dec r nz z** decrements register **r** and goes to **nz** when the register was positive and to **z** otherwise; **inc r next** increments and goes to **next**.

program	registers	function
add	2	$r_0 := r_0 + r_1$
double	2	$r_0 := 2r_0$, with r_1 as scratch
monus	2	$r_0 := \max(r_0 - r_1, 0)$
mult	4	$r_0 := r_0 \cdot r_1$, with r_2 as accumulator and r_3 restoring r_1
parity	1	$r_0 := 1$ if r_0 is odd, else 0
loop	1	increments r_0 forever

For example **add** is the three-instruction program

```
0: dec 1  nz 1  z 2
1: inc 0  next 0
2: halt
```

and multiplication uses ten instructions, nested loops that add r_1 into r_2 once for each unit of r_0 , restoring r_1 from r_3 after each pass, and finally move r_2 into r_0 . Their sizes after compilation are given in the table of Section 9.

B Reproducing the numbers

The companion is the directory **experiments/combinators**. Running **node test.mjs** executes the verification groups in order, prints the title and bounds of each, and writes the measured facts to **generated/facts.json**. Running **node report.mjs** converts these to **generated/facts.tex**, which defines one macro per number used in the text, and **generated/tables.tex**, which defines each table. The essay includes both files, so that rerunning the two commands regenerates every number. The independent implementation is **oracle.py**, invoked by the suite as a subprocess.

References

- [1] H. P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, revised edition, North-Holland, 1984.
- [2] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.
- [3] A. Church and J. B. Rosser, Some properties of conversion, *Transactions of the American Mathematical Society* 39 (1936), 472–482.
- [4] H. B. Curry and R. Feys, *Combinatory Logic*, volume I, North-Holland, 1958.

- [5] J. R. Hindley and J. P. Seldin, *Lambda-Calculus and Combinators: An Introduction*, Cambridge University Press, 2008.
- [6] G. Huet, Confluent reductions: abstract properties and applications to term rewriting systems, *Journal of the ACM* 27 (1980), 797–821.
- [7] G. Huet and J.-J. Lévy, Computations in orthogonal rewriting systems, in J.-L. Lassez and G. Plotkin (eds.), *Computational Logic: Essays in Honor of Alan Robinson*, MIT Press, 1991.
- [8] J. W. Klop, Term rewriting systems, in S. Abramsky, D. Gabbay and T. Maibaum (eds.), *Handbook of Logic in Computer Science*, volume 2, Oxford University Press, 1992.
- [9] D. E. Knuth, *The Art of Computer Programming*, volume 3: *Sorting and Searching*, section 5.1.4, Addison-Wesley.
- [10] J. McCarthy, Recursive functions of symbolic expressions and their computation by machine, part I, *Communications of the ACM* 3 (1960), 184–195.
- [11] M. L. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, 1967.
- [12] T. Æ. Mogensen, Efficient self-interpretation in lambda calculus, *Journal of Functional Programming* 2 (1992), 345–364.
- [13] M. J. O’Donnell, *Computing in Systems Described by Equations*, Lecture Notes in Computer Science 58, Springer, 1977.
- [14] B. K. Rosen, Tree-manipulating systems and Church–Rosser theorems, *Journal of the ACM* 20 (1973), 160–187.
- [15] M. Schönfinkel, Über die Bausteine der mathematischen Logik, *Mathematische Annalen* 92 (1924), 305–316.
- [16] R. P. Stanley, *Enumerative Combinatorics*, volume 2, Cambridge University Press, 1999.
- [17] M. Takahashi, Parallel reductions in λ -calculus, *Information and Computation* 118 (1995), 120–127.
- [18] D. A. Turner, Another algorithm for bracket abstraction, *Journal of Symbolic Logic* 44 (1979), 267–270.
- [19] C. P. Wadsworth, *Semantics and Pragmatics of the Lambda-Calculus*, DPhil thesis, University of Oxford, 1971.