The Structure of the Collatz Trajectories with an Inductive/Constructive Proof of the Conjecture

Kenneth Conrow, Kansas State University (retired)

Introduction

The Collatz conjecture has attracted hundreds of interested workers during the seventy years since its proposal but is still without a generally accepted proof. The conjecture involves a recursive iteration on the positive integers and proposes that the trajectories of these iterations are such that convergence to 1 will result from any starting value. We need only prove that the Collatz graph is comprehensive of the positive integers and weakly connected (see Lagarias [1]). Additional recent references, not necessarily credible, are available from Cawaling [2], who references portions of this work.

Approach to a Proof of the Collatz Conjecture

The conjecture can be proven easily (Lagarias [1]) by developing a structure which accommodates the peculiar behavior of its trajectories. This work develops that structure.

Accordingly, only four aspects of the Collatz problem are employed here:

a) the state transition diagram for the iteration establishes the valid Collatz successor and predecessor transitions,

b) the abstract predecessor graph constructed from the allowed predecessor transitions represents an infinite set of infinitely recurring constructs for the examination of the trajectories at a higher level,

c) the combinatorics identified in the elements of the abstract predecessor tree based upon the densities of the integers in the recurring constructs show that all odd integers are included in it, and

d) an inductive constructive process builds the binary predecessor tree from the individual elements of the recurring constructs to complete the proof.

Instead of the usual formula for the Collatz trajectory producing a successor trajectory element (T_s) from its predecessor (T_p)

$$T_s = \begin{cases} \frac{3*T_p+1}{2} & \text{for } T_p \in 1[2] \\ \frac{T_p}{2} & \text{for } T_p \in 0[2], \end{cases}$$

we will employ the successor(1) and predecessor(2) formulas:

$$\{T_s\} = \frac{(3 * \{Tp\} + 1)}{2^i}$$

where $i|\{T_s\} \in \{1[2]\}$ equation (1)

$$\{T_p\} = \frac{(-1 + \{T_s\} * 2^i)}{3}$$

where $i \in \{0[2]\}$ if $T_s \in \{1[3]\}$ or $i \in \{1[2]\}$ if $T_s \in \{2[3]\}$ equation (2)

Note: the predecessor direction is from a trajectory element to its Collatz predecessor, defined by equation 2, opposite to that of the Collatz trajectory, defined by equation 1. These two equations thus define the terms successor and predecessor. The defining equations are inclusive of arbitrary numbers of divisions or multiplications by 2, with special attention reserved for $i \in \{1, 2\}$. While the predecessors and successors are shown as sets, these equations apply equally well on an element by element basis. It might

appear that equation 2 introduces an element of choice into the determination of the predecessor, but in fact the value of $T_s[3]$ (i.e. T_s mod 3) determines the power of 2 to be used in the calculation of its predecessor.

By way of introduction to the elementary behavior of the trajectories, and especially to bits of useful nomenclature, both a general tree [PG-gentree] and a binary tree [PG-bintree] picture of the initial portions of the Collatz predecessor graph are offered. The general tree faithfully depicts all the Collatz trajectories from deep in the tree back to the root node at 1. The binary tree follows the dissection of the tree into left descent assemblies and extensions. These views alone contribute little to a proof of the Collatz conjecture.

The National Institute of Standards [3] gives a concise definition of a tree and many specialized trees. Wikipedia [4] provides in depth discussion and associated definitions *re* trees both as graphical structures and data structures. However, "left descent assembly" and "extensions" are peculiar to this work, and derive from the appearance/attribute of certain substructures in a binary tree. A left descent assembly is a branch of the binary tree descending to the left. (Its properties will be more carefully defined later). Extensions are the right descents in the binary tree. (The name reflects the fact that they all share a common Collatz successor.) Clearly, this adoption of trees and bits of nomenclature associated with trees as graphical structures to represent the Collatz graph is premature; nothing has yet been produced to support this usage.

State Transition Diagram for the Collatz Iteration

A state diagram [5] is a graph depicting the actions and capabilities of an automaton – a mindless device whose actions are completely described by a set of rigid rules. Each state is shown as a node (vertex) in the graph. The transitions from state to state are shown as arcs (edges) in the graph. Both features are combined in a single diagram to form a state transition diagram. The state machine employed here is a much simplified one: it is deterministic; any state at all can be the starting state so the transition to be made at any time is entirely determined by the value associated with each state; there are no external inputs.

Def'n: $\{T_a\}$ is defined as a set of transition results in the abstract predecessor graph. It represents a set of trajectory elements $\{T_a \in \{c[d]\}$, later to be resolved into $T_a \equiv \{c[d]\}$, where$\{c[d]\}$ represents a residue set (defined as all integers congruent to c modulo d).

The Collatz trajectories are complex enough that considerable simplification is attained by hiding the details on the diagram's [PG-ldastate] legend. In this simplified and higher level representation, the behavior of the itineraries is presented without the underlying clutter.

The interplay between powers of 2 and 3 in the Collatz trajectories requires that the state descriptors (i.e. the labels used to characterize each state of the state machine) capture these two aspects. The residue sets $\{T_a[3]\}$ and $\{T_a[8]\}$ are employed. The state transition diagram is then readily constructed. It [PG-ldastate] shows each Collatz transition as an edge from every $\{T_a\}$ to its immediate predecessor. No transitions other than those shown exist for Collatz itineraries (except that we omit edges to leaf nodes).

$\{T_a \in \{5[8]\}$ have no incoming edges. All the other nodes accept incoming edges and can chain together to form possibly lengthy left descent assemblies. The selection of $\{5[8]\}$ as the root of the abstract predecessor tree leaves the complementary sets $\{1[8]\}, \{3[8]\}, \{7[8]\}$ to constitute all the other nodes of the abstract tree. As $\{5[8]\}$ cycles through its values mod 3, the double descriptors $\{2[3]$ and $5[8]\}$, $\{1[3]$ and $5[8]\}$, and $\{0[3]$ and $5[8]\}$, or, equivalently, $\{5[24], \{13[24]\}$, and $\{21[24]\}$, respectively, emerge to develop the sets of integer predecessors for each set of elements in the Collatz graph. Certain states (those consisting of integers in $\{0[3])\}$) do not have predecessors when $i \in \{1, 2\}$ (see equation 2). These "leaf nodes" are omitted from the diagram.

The Collatz graph is built solely on the odd positive integers. All positive odd integers appear in $\{\{0, 1, 2\}[3]\}$ and $\{1, 3, 5, 7\}[8]\}$, so there is at least the potential of complete coverage. The even numbers are included later as the powers-of-two multiples of the odd numbers, i.e. $\{T_a \in \{0[2]\}\} \leftarrow 2^k * T_a \in \{1[2]\}, k \in N$.

The Structure of the Collatz Trajectories

A diagram depicting transitions available to even numbers [PG-evensdgm] is presented. It shows that a sequence of even iterates must return to the all-odd state transition diagram.

The transition state diagram is written from the viewpoint of identifying predecessors for any integer or set of integers. This practice will be continued throughout this work. In the diagram and in the legend, the arrows point from Collatz successor to Collatz predecessor. The accuracy of the numerics in the legend, used to identify each transition, may easily be checked by applying the familiar Collatz transition rules from right (predecessor) to left (successor). The C to C transition is given two prototypes in the legend, because the $1 \rightarrow 1$ transition occurs only at the root of the Collatz graph (thus hardly typical), whereas the $73 \rightarrow 97$ example is typical of the C to C transitions everywhere else in the predecessor graph.

An augmented state transition diagram [PG-pthstate] showing the linkage between three left descent assemblies is offered and discussed briefly here. In that diagram the same internal nodes are shown, but leaf nodes and extensions are shown as collected by large curly braces which explicitly provide entries into deeper left descent assemblies at their header nodes. The extensions, $\{T_a \in \{5[8]\}$, come from equation 2 with i>2. This diagram illustrates how transition from any left descent assembly to a deeper left descent assembly occurs whether they arise from an internal node of the originating left descent assembly or from a leaf node.

To summarize, the state transition diagrams indicate that any Collatz predecessor graph is composed of left descent assemblies linked together by extensions which serve as headers for another generation of left descent assemblies. Every odd number has extensions, but only those extensions in $\{1[3]\}$ or $\{2[3]\}$ appear as headers of additional left descent assemblies. The extensions in both $\{0[3]\}$ and $\{5[8]\}$ are both leaves and left descent assembly headers, so these nodes represent null left descent assemblies containing no internal nodes.

With the observations gleaned from the state transition diagram it is a small step to develop an abstract predecessor graph. A tree can be the only result because only $\{5[8]\}$ can serve as the root and the other nodes can only bifurcate at best. This is the first justification for using a tree structure and its related nomenclature as the basis of discussion.

The Abstract Predecessor Tree

The abstract tree is a data structure whose nodes are infinite sets of odd integers $\{c[d]\}$, i.e. congruent to c modulo d. We often refer to (and manipulate) these sets using formulas $\{dn+c\}$, with $d \in \{2^a * 3^b\}, [a,b] \in Z+, n \in N\}$, reflecting the means of their derivation (below). The abstract tree serves to construct and give underlying meaning to the key feature of this work, left descent assemblies.

Def'n: A left descent assembly is a path from the root of the abstract predecessor tree (i.e. $\{T_a \in \{5[8]\}\}$) to a leaf set (i.e. $\{T_a \in \{0[3]\}\}$), inclusive of the root, internal nodes, and leaf node. In equation 2, $i \in \{1,2\}$ in left descent assemblies, thus constraining the number of divisions by 2 in a left descent assembly portion of a Collatz trajectory. $\{\{T_a \in \{1[3]\}\}$ produce predecessors **b**igger than themselves, and $\{T_a \in \{2[3]\}$ produce predecessors **s**maller than themselves, whence we call the former **b**-steps and the latter **s**-steps.

Internal nodes in left descent assemblies $\{T_a\}$ are subsets of $\{k[8], k \in \{1,3,7\}\}$ which in turn contain two subsets ($\{1[3]\}$ and $\{2[3]\}$) which lead via **b**- and **s**-steps to two predecessor sets, as controlled by the value of $floor(mod(T_a, 71)/24)$. Only the header and internal nodes are included in the state transition diagram because the leaf nodes $\{T_a \in 0[3]\}$ have no predecessors and cannot contribute to elaboration of a left descent assembly under the restriction that $i \in \{1,2\}$. These ideas are diagrammed [PG-fig3] to indicate their recursive application to the sets which constitute the abstract tree. The name of the abstract tree derives from the fact that it provides a high level (more abstract?) view of the structure which exists in the Collatz graph.

Any set in the abstract tree other than a leaf node set can give rise to a predecessor set by either an **s**- or a **b**- step. Development the resulting abstract tree is shown schematically [PG-fig4] using a process of

subdividing the set contents of 5[8] and all its predecessor sets through three levels of development. As is evident all possible left descent assemblies are formed by systematic use of both **s**- and **b**-steps from the root and then, in turn, each successively developed node. A diagram scaled proportionate to the "size" of the residual sets after three stages of this subdivision into separate sets shows [PG-nestsets] how few generations these operations take to reduce the residual sets in the abstract tree to minuscule fractions of the original "size" (see more below).

Every path from root to a terminating leaf set in the abstract tree represents a left descent assembly. Left descent assemblies can be denoted by a string "$e(b|s)_k t$", where the e represents the extension header, the parenthesized expression indicates $k(\in N)$ selections in any order of **s**- or **b**-steps, and the t represents termination due to having reached a leaf set. This expression denoting the detailed structure of a left descent assembly is in predecessor order. Read left to right to follow the growth of the predecessor tree. Read right to left to follow a path segment in the direction of the Collatz trajectory. We will most often use the predecessor direction for left descent assemblies and other itinerary segments.

We need now to develop the $\{c[d]\}$ residue class for every element of every left descent assembly. For example, the root node $\{5[8]\}$ in the abstract tree includes as subsets the particular $\{c[d]\}$ residue class for the root of every left descent assembly. All the nodes at deeper levels in the abstract tree similarly contain the particular $\{c[d]\}$ describing the left descent assembly element located at that depth of that left descent assembly. When equation 2 is employed to produce each node's predecessors, a leaf set is identified as well as two sets destined to produce predecessor sets. Application of the predecessor equation to these two subset provides a formula for each predecessor set, so the sets are not only repeatedly divided, but each generation of predecessors are also tentatively identified as particular $\{c[d]\}$ residue sets. The leaf node's $\{c[d]\}$ value is precisely that describing the leaf set resulting from the left descent assembly traced to a leaf node at that point. This process is repeatedly applied to achieve the set subdivision iteratively.

Once some leaf sets are identified, applying equation 1 through the successor (upward) pathway, provides a series of $\{c[d]\}$ values which represent those of the individual member sets of the left descent assemblies traced, always subsets of the values observed in the predecessor direction, and concluding with the particular subset of $\{5[8]\}$ which represents the header node of the left descent assembly traced. It is at this stage where predecessors are identified as particular $\{c[d]\}$ residue sets within a particular left descent assembly.

The process is pictured schematically, and can be accomplished by hand in a down- [PG-fig5] and up-again [PG-fig6] process using the $dn + c$ formula (or by computer program [PL-treegrow]).

Note that there has been a subtle transformation effected by the determination of residue classes for the nodes in the left descent assemblies in the abstract tree. The abstract tree nodes which were first described as members ($\in$) of all the sets in this path to leaf nodes (at whatever depth) have been converted to an identity ($\equiv$) describing the residue set of that single set representing each upwardly traversed node of the left descent assembly which culminated at the operative leaf node.

Two illustrative subsets (complete [MG-sievord] and headers only [MG-ulamprim]) of the formulas for left descent assemblies elements are provided. Margaret Maxfield has provided a proof [MG-disjoint] that all these residue classes are unique. This uniqueness is a consequence of the tree structure of the Collatz graph, and is helpful in allaying claims that loops are hidden somewhere in the graph.

Since every element of every left descent assembly is a unique residue set $\{T_a \equiv c[d]\}$ we may turn to the task of "counting" the integers in that infinite set (of left descent assemblies) of infinite sets (instantiations over $\{k \in N\}$, i.e. $k(dn + c)$ of every element in every left descent assembly).

The Density Metric, Combinatorics, and Infinite Summations

Cardinality is an extensive measure (e.g. mass, volume, length) and cannot distinguish ∞ from $\infty - k$, $k \in N$. An intensive measure involves a ratio of two measurements and can be used to determine the ratio of

the number of integers appearing in some span d of the number line to the number, d, which would be present if the span were completely filled. Since the density among the odd integers of that one integer included in any $\{c[d]\}$ is $2/d$, the density contributed by any set of i left descent assemblies is readily determined by summation of the $2/d$ values of all the j elements in each set of left descent assemblies under inspection.

$$\sum_i \sum_{j_i} \frac{2}{d_{j_i}}$$

Two first examples (appearing as legends to [PG-nestsets]) may serve to illustrate the use of density as a metric. These examples serve to validate the contents of the abstract predecessor tree, by showing that the total density of the sets of odd numbers in it is 1, i.e. the totality of them, and that there are as many header elements as leaf elements among the left descent assemblies. But a much more detailed (and therefore more convincing) accounting is available.

An annotated Pascal triangle [PG-pascalt] reveals the binomial pattern of the frequencies of occurrence of left descent assemblies sharing a common (a, b) of the $2^a * 3^b$ expression which is the d of $c[d]$, i.e. the number of ways paths can be traced to any $2^a * 3^b$ using **s**- and **b**- steps. Since **s**-steps introduce a single power of 2 into their predecessors, while **b**-steps introduce 2 powers of 2, paths heavy with **b**-steps attain a higher a value in $2^a * 3^b$ than those at the same depth but heavy with **s**-steps. An "isobar" is superimposed over the entries in the Pascal triangle sharing a common power-of-2 depth into the abstract tree. The cardinalities in the isobars are the successive elements of the Fibonacci series.

An exploration of the predecessors starting from 5 up to several tens of millions was made by program [MG-treegrow] and the $c[d]$ values for every element of every left descent assembly encountered reported. A cardinality table [PG-fig7] listing the number of occurrences of elements of left descent assemblies sharing an (a, b) value was produced from this output. This table, combined with the known summation[7] of the Fibonacci series (sum(F(i)/2(i+1),i=1..infinity)=1) where F(i) are the Fibonacci numbers, permits (with allowance for offsets and multiples among the F(i) in the cardinality table) a summation of the densities of the odd integers across the whole infinite abstract predecessor tree. The total density will be the sum(2**i/3**i+2,i=0..infinity) which Maple reports as 1/2. This summation indicates that the abstract tree contains all the odd integers. (Taking the limit of the uncounted odd integers as a goes to infinity confirms this result.)

The summation of the densities of the left descent assembly element sets depends upon the boundaries being carefully carefully aligned, since two partially overlapping regions might otherwise both have their content (or lack of content) in the overlapped region, but setting the left edge of all the regions at 3, knowing that all regions have a width of $2^a * 3^b$, establishes accurate overlap of the regions at their common multiple.

The Constructive Inductive Proof

Two illustrations have been prepared depicting the nodes available for Collatz graph construction and their properties. The first [PG-graphiso] presents isolated nodes without context, and the second [PG-graphcon] shows the node types in the context of a binary predecessor tree in which a tree's potentially great depth is indicated by a flight of stairs before attention is given to the details of fitting each node type into its proper context in the tree construction.

The nodes available for the inductive proof by construction are infinite in number, each unique, and each labeled with an integer in Z^+ which determines that node's node type.

The construction is this: start [PG-bldtree] with a two-node binary tree consisting of the root, 1, and its first extension, 5. It is a tree, subsuming whatever problems are associated with the unique root. This tree has 2 unsatisfied edges. This two-node tree is the basis for induction by repeated addition of an appropriate new node as determined by the value of T_s which selects the correct T_p according to equation 2.

Now two cases appear for building the tree. Either a left descent assembly element is added as a left child or an extension element is added as a right child.

Case a: If a left descent assembly element is added as a left child, a tree will result and the resulting tree will either have the same number of unsatisfied edges (if the left descent assembly element is a leaf node), or one additional unsatisfied edge (if the left descent assembly element is not a leaf). It makes no difference to the inductive argument whether the added element is the result of a **b-** or an **s-** step at this point – the choice is forced by the numeric value T_s and the effect on the growing tree is identical.

Case b: If an extension element is added as a right child, a tree is again obtained, either with no increased number of unsatisfied edges (if the extension heads a null left descent assembly), or one additional unsatisfied edge (if the extension is a left descent assembly header).

In either case a binary tree results, and the resulting tree will contain at least as many unsatisfied edges as the previous one, but often one more. At any stage of this inductive tree construction, a tree is the point of departure, and an elaborated tree results from the addition of a new node. The number of unsatisfied edges increases monotonically as additional nodes are added, assuring development of an infinite tree. The existence of every odd positive integer in the stock of nodes being used to construct the tree was established by the infinite summations of the content of the abstract tree.

That completes the induction. The growing demand of the unsatisfied edges in the growing tree increases without limit, and thus the tree cannot stop growing. All integers are assured of a place because all have been shown to exist in the abstract tree. The only conclusion possible is that the Collatz predecessor graph is an infinite directed binary tree. This kind of tree is weakly connected, as are most trees. Even a weaker result would have sufficed to prove the Collatz conjecture. That the tree is a directed binary tree is a bonus.

Acknowledgements

The symbolic algebra package, Maple, was the *sine qua non* for this work. I've had email correspondence with several dozen people around the world, much of which was helpful, but special mention must be made to Joe Parranto, and Mensanator.

References

1. Jeffrey Lagarias, "The 3x+1 problem and its generalizations"
   @Internet://www.cecm.sfu.ca/organics/papers/Lagarias/index.html
   and http://arxiv.org/abs/math.NT/0309224
2. Benjamin E. Cawaling Jr., "Collatz Conjecture Proved"
   @Internet://www.geocities.com/bencawaling/collatz/conjecture/proved.pdf
   and http://www.geocities.com/bencawaling/collatz/conjecture/proved/appendix.pdf.
3. "binary tree"
   @internet://http://www.nist.gov/dads
4. "Tree (Data Structure)"
   @Internet://en.wikipedia.org/wiki/Tree_data_structure
5. "finite state machine"
   @INternet:http://www.nist.gov/dads/HTML/finiteStateMachine.html
6. Picture Gallery [PG-xxxxx] of relevant material produced by the author and made available at
   @Internet:http://www-personal.k-state.edu/∼kconrow/pictgal.html

Readers may wish to keep a browser window open to the picture gallery for easy reference while working through the TeX paper. The pictures are a mixture of .gif files and .html files. When the entry in pictgal is

an .html it appears as a pointer; click it and later return to the picture gallery by using the browser's back arrow. When the entry is a .gif, the picture itself will appear .

PG-gentree. Predecessor graph rendered as a general tree.

PG-bintree. Predecessor graph rendered as a binary tree.

PG-ldastate. The state transition diagram for a left descent assembly in the Collatz predecessor graph.

PG-evensdgm. The state transition diagram among the even iterates in the Collatz predecessor graph.

PG-pthstate. Three left descent assemblies state transition diagrams sequenced to show inter-left descent assembly connections.

PG-fig3. Recursive construction of the abstract predecessor tree.

PG-fig4. The abstract predecessor tree through 3 levels of depth, annotated with the nodes' left descent assembly's names.

PG-nestsets. The abstract tree through 3 levels of depth with the horizontal axis scaled to show the rate of assignment of c[d] values to isolated subsets.

PG-fig5. Downward stage of dissection of c[d] subsets i.e. of successors into those of predecessors.

PG-fig6. Upward stage of dissection of c[d] subsets i.e. of predecessors into those of successors.

PG-pascalt. Annotated Pascal Triangle showing the source of the Fibonacci numbers on the power of two isobars.

PG-fig7. Cardinality Table of left descent assemblies sets sharing a common value of $(2^a * 3^b)$ in d

PG-graphiso. Characterization (in isolation) of the three (or is it five?) node types available for the inductive construction process.

PG-graphcon. A schematic of the Collatz predecessor graph accompanied by the available node types.

PG-bldtree. A detailed example of the result of the first several steps of the inductive construction of the binary predecessor tree.

7. "The Fibonacci Series ... One over Eighty-nine". Source (with 2 substituted for 10) of the Fibonacci sum expression

   @Internet://library.thinkquest.org/27890/applications3p.html

8. Miscellany Gallery [MG-xxxxx]of relevant material produced by the author and made available at @Internet:http://www-personal.k-state.edu/∼kconrow/miscgal.html

Readers may wish to keep a browser window open to the miscellany gallery for easy reference while working through the TeX paper. The miscellany items are all .html files so the entry in miscgal.html will appear as a pointer; click it and later return to the miscellany gallery by using the browser's back arrow.

MG-treegrow. This is the listing of the program which systematically generates all the left descent assemblies up to specified limits.

MG-sieveord. This file shows the first 47 complete left descent assemblies in order of increasing magnitude of their header node.

MG-ulamprim. This file gives the headers only of the first 745 left descent assemblies

MG-disjoint. This is Margaret Maxfield's proof that the c[d] values derived during the subdivision of the abstract tree are disjoint.