ABSTRACT INTERPRETATION USING LAZINESS: PROVING CONWAY'S LOST COSMOLOGICAL THEOREM

KEVIN WATKINS

Abstract. The paper describes an abstract interpretation technique based on lazy functional programming, and applies it to the proof of Conway's Lost Cosmological Theorem, a combinatorial proposition analogous to the four color theorem or Kepler's conjecture, which essentially states that a certain predicate holds of all lists of integers from 1 to 4. The technique makes use of the semantics of Haskell in the following way: evaluating a predicate on a partial lazy list to True proves that the predicate would evaluate to True on any list extending the partial list. In this way proving a property of all lists can be reduced to evaluating the property on sufficiently many partial lists, which cover the set of all lists. The proof is completed by proving the correctness of the code implementing the predicate by hand. The oracle that chooses a covering set of partial lists need not be verified. In this way the amount of program code which must be verified by hand in order to complete the proof is reduced, increasing confidence in the result.

1. Introduction

This paper is about how to use the programming language Haskell's lazy semantics as a kind of abstract interpretation, and how this idea can yield a proof of Conway's Lost Cosmological Theorem [Con87]. The Theorem was proved by hand, by Conway and others before 1987, but the proof was lost, hence the *Lost* Cosmological Theorem. It was re-proved by Zeilberger and his computer Ekhad in 1997 [EZ97], and another computerized proof with tighter bounds was given by Litherland in 2003 [Lit03b].

I was unable to completely verify Zeilberger's and Litherland's proofs to myself, because the computer programs they used were not given in the text of their papers, and the high-level descriptions they provided of their algorithms were not closely related enough to the code for me to be convinced of the programs' correctness. Perhaps a reader cleverer or more persistent than I was could have seen why their code was correct. But it seemed to me that a convincing proof should be presented as a simple program whose invariants would be easy to understand. The program should be small enough to include in a paper in full.

While Zeilberger's and Litherland's programs were given in Maple and C, respectively, my considerations led me to Haskell because its well defined semantics supports simple equational reasoning principles that Maple and C do not. It was after some initial work on the proof in Haskell that I discovered the abstract interpretation technique based on laziness that I will describe. Other well defined functional languages such as ML or Scheme might have been used instead; it is possible to define lazy primitives in the latter two languages equivalent to Haskell's, if a bit more cumbersome. The proof and the abstract interpretation technique do not make any sophisticated use of these languages' higher-order computation capabilities, or of their powerful type systems. This suggests that the popular phrase HOT (Higher-Order Typed) used to refer to these languages is leaving out a key benefit of their designs: namely, that they support simple reasoning principles.

Briefly, the abstract interpretation technique relies on Haskell's lazy evaluation to check properties of infinitely many sequences in finite time. This is possible because evaluating a predicate $p\ (1:2:3:\bot) = \mathit{True}$, say, on a partial list proves that the predicate would evaluate to *True* on any finite list extending the partial list (e.g. [1,2,3], [1,2,3,1], [1,2,3,7,7,7], etc.), by the monotonicity of p's denotation. (Here $\bot$ is Haskell's expression *undefined*, the bottom element of Haskell's denotational semantics.)

In order to prove a property of all finite sequences (or all those in an interesting subset of the finite sequences), it is necessary to select a finite set of approximants like $1:2:3:\bot$ which together cover all the finite sequences of interest. The approximants must be carefully selected so that evaluating the predicate p of interest will complete rather than yielding $p\ (1:2:3:\bot) = \bot$. So in general the selection of the approximants can involve rather complicated code.

If it were necessary to verify all this code by hand, the proof would be hard to understand and hard to trust. Fortunately, it is possible to define, once and for all, a function *cover* which selects an appropriate covering set of approximants. The function *cover* invokes an oracle to decide how far to refine the set of approximants, but in such a way as to make it easy to show that the approximants form a covering set, no matter what the oracle does. This makes it unnecessary to verify any property of the code implementing the oracle.

This paper walks through the theory of Conway's "audioactive decay", showing how some key results in Conway's theory can be proved by the abstract interpretation technique: the Starting Theorem, the correctness of a parsimonious splitting function, and the Cosmological Theorem. For the most part the development is self-contained, although proofs are not given for some results proved directly in Conway's article.

All of the code in this paper is presented in the language Haskell 98 [Jon03]. The study of this elegant language is highly recommended, and it will be assumed that the reader has a basic familiarity with it. The code is shown piecemeal as each part is discussed. The source file for this paper, available from the author's web site, is a literate Haskell program and can be input directly to a Haskell compiler. The paper thus uses the notational conventions, e.g. $\in$ for `` `elem` ``, of the `lhs2TeX` package by Andres Loeh and Ralf Hinze.

2. Conway's theory of audioactive decay

Conway's Cosmological Theorem concerns a mathematical recreation, invented by him, called "audioactive decay" [Con87]. (The pun on "radioactive decay" will be made clear later.) Conway proposes the following transformation on finite sequences (lists) of positive integers: given a sequence, read it aloud, and record as the transformed sequence what you say. For example, if the input sequence were 33114555, you would say "two threes, two ones, one four, three fives", and the output sequence would be 23211435.[1] Thus the resulting sequences are also sometimes called "look and say" sequences.

The Haskell code will represent these sequences as lists of type $[\mathit{Int}]$ all of the members of which are positive.[2] (In this paper, the elements of a list will be called "members" to avoid confusion with a different notion of "element" introduced below.) The code performing the look and say transformation can be written as follows.

$$\begin{aligned}
&\mathit{say} :: [\mathit{Int}] \to [\mathit{Int}]\\
&\mathit{say} = \mathit{concat} \circ \mathit{map}\ \mathit{code} \circ \mathit{runs}
\end{aligned}$$

The transformation is the composition of three stages. In the first stage, the input list is separated into maximal runs of repeated integers.

$$\begin{aligned}
&\mathit{runs} && :: [\mathit{Int}] \to [[\mathit{Int}]]\\
&\mathit{runs}\ [\,] && = [\,]\\
&\mathit{runs}\ (x : \mathit{xs}) && = (x : \mathit{ys}) : \mathit{runs}\ \mathit{zs}\\
& && \quad\ \ \mathbf{where}\ (\mathit{ys}, \mathit{zs}) = \mathit{span}\ (== x)\ \mathit{xs}
\end{aligned}$$

The second stage replaces each run with its verbalization, by mapping the following function over the list of runs.

$$\begin{aligned}
&\mathit{code} && :: [\mathit{Int}] \to [\mathit{Int}]\\
&\mathit{code}\ \mathit{xs} && = [\mathit{length}\ \mathit{xs}, \mathit{head}\ \mathit{xs}]
\end{aligned}$$

Finally, the verbalizations are catenated.

It is assumed that the reader has experience with the algebraic calculations needed to establish the correctness of functions like *say*. A good introduction to this mode of reasoning is Bird and de Moor's excellent book [BdM96]. As these manipulations are straightforward for many of the functions presented in this paper, like *say*, they are left to the reader.

Conway investigates the behaviour of these sequences as the function *say* is iterated. We write:

$$\begin{aligned}
&\mathit{iterate} && :: (a \to a) \to (a \to [a])\\
&\mathit{iterate}\ f\ x && = x : \mathit{iterate}\ f\ (f\ x)\\
&\mathit{isay} && = \mathit{iterate}\ \mathit{say}
\end{aligned}$$

and now we have, for instance,

```
? take 10 (isay [2])
[[2],
 [1,2],
 [1,1,1,2],
 [3,1,1,2],
 [1,3,2,1,1,2],
 [1,1,1,3,1,2,2,1,1,2],
 [3,1,1,3,1,1,2,2,2,1,1,2],
 [1,3,2,1,1,3,2,1,3,2,2,1,1,2],
```

[1]The sequences we will be dealing with will contain only small integers, so they will be run together without punctuation in the text.

[2]The reader may easily verify that the proofs that follow are not materially affected by the restriction to integers within the representable range of *Int*.

$$[1,1,1,3,1,2,2,1,1,3,1,2,1,1,1,3,2,2,2,1,1,2],$$
$$[3,1,1,3,1,1,2,2,2,1,1,3,1,1,1,2,3,1,1,3,3,2,2,1,1,2]]$$

(In this paper, expressions one might type into a Haskell interpreter are flagged by **?**, followed on the next line by the interpreter's response.)

The members of *isay xs* are called *descendants* of *xs*. A particular descendant can be picked out with Haskell's list indexing operator (!!); for example, *isay xs* !! 5 is the fifth descendant of *xs*, and *xs* is its own zeroth descendant.[3] We also call any sequence of the form *isay xs* !! *n* for some *xs* "*n* days old." An *n*-days-old sequence is thus also *m*-days-old for any $0 \leqslant m \leqslant n$.

2.1. **Overview of the Cosmological Theorem.** Now immediately questions arise about the behavior of sequences under *isay*. Do they generally get longer, or shorter? What is the asymptotic length of a sequence at the *n*th step, as *n* goes to infinity? Do the sequences have a simple structure, or are they essentially random? Conway proves the Cosmological Theorem in order to answer all these questions, in a way which will be described once the theorem itself has been proved.

The following is an overview of the proof, given in order to provide a framework for understanding the results which will be presented in the rest of the paper.

The overall structure of the theorem proceeds in three stages:

(1) Split sequences into parts (*elements*) that evolve independently under *isay*.
(2) Investigate how elements evolve into (*decay* into) combinations of other elements.
(3) Classify the elements that appear in *n*-day-old sequences for arbitrarily large *n*.

The first stage relies on the idea of splitting a sequence into parts which evolve independently. We say a sequence *xs* splits into *ys* . *zs* if

$$\mathit{isay}\ \mathit{xs} = \mathit{zipWith}\ (+\!\!+)\ (\mathit{isay}\ \mathit{ys})\ (\mathit{isay}\ \mathit{zs})$$

For example, 2111 splits into 2 . 111 (although we don't yet have the tools to prove this):

2 . 111
12 . 31
1112 . 1311
3112 . 111321
132112 . 31131211
...

Looking at the table above, each line is both the *n*th iterate of *say* on 2111, $0 \leqslant n \leqslant 4$, and the catenation of the *n*th iterate of *say* on 2 and on 111. Assuming for the moment that this pattern continues for all $n \geqslant 0$, we have that 2111 splits into 2 and 111. On the other hand, 111 does not split further into 1 and 11, because *say* $[1,1,1] = [3,1]$ while *say* $[1] +\!\!+$ *say* $[1,1] = [1,1,2,1]$.

A major part of the setup for the Cosmological Theorem is the derivation of a decision procedure for splitting. Conway then defines an *element* as a sequence that is irreducible with respect to splitting, which is to say, it does not split into shorter sequences. Conway shows that any sequence splits into a unique finite

[3]In Haskell, operators like (!!) bind more weakly than function application.

sequence of elements. The decision procedure for splitting extends to an algorithm for computing these factorizations into elements.

The second stage of the development leading to the Cosmological Theorem is to characterize the way elements decay into other elements. If *xs* is an element, we say *xs* decays into the elements constituting the splitting for *say xs*. The elements in *say xs* then further decay into combinations of elements in *say* (*say xs*), and so forth.

Finally, Conway isolates 92 special elements, which he calls the *common elements*, and 2 infinite families of elements, which he calls the *transuranic elements*. We can now preview the statement of the Cosmological Theorem: *every sequence decays eventually into a compound of common and transuranic elements*. It so happens that there is a uniform bound on the number of steps required for this to happen: namely, a sequence has always decayed into common and transuranic elements after 24 iterations of *say*. Like Zeilberger, we will not attempt to prove the tight bound, but it can be established by a straightforward, if somewhat tedious, application of the methods of this paper.

3. Lemmas on sets of sequences

The proof of the Cosmological Theorem relies first on a number of lemmas regarding the structure of one-day-old and two-day-old sequences, stated in this section.

3.1. The One-Day Theorem.

The first step in Conway's analysis is the characterization of one-day-old and two-day-old sequences. The first characterization is given by the One-Day Theorem, which arises as follows. The definition of the look and say sequences might at first appear to be ambiguous: rather than reading 55, say, as "two fives," we might instead choose to read it as "one five, one five", so the resulting sequence would be not 25 but 1515. The definition in Haskell resolves this ambiguity in favor of decomposing the input sequence into the longest possible stretches of identical members, or equivalently in favor of the shortest possible output sequence.

This has the consequence that not every possible sequence of even length is an output of the look and say transformation; 1515 could only be the output corresponding to 55, but we see that 55 becomes 25 instead. So the look and say transformation is injective but not surjective.

The One-Day Theorem characterizes those sequences which are outputs of *say*, the "one-day-old" sequences. We call $x_1 x_3 \ldots$ the odd-indexed subsequence of $x_0 x_1 x_2 x_3 \ldots$ (sequences being indexed from zero). Then a one-day-old sequence is just a sequence of even length such that its odd-indexed subsequence has no consecutive repeated members.

Theorem 1 (Conway [Con87])**.** *A sequence is one day old iff its length is even and its odd-indexed subsequence has no consecutive repeated members.*

A Haskell predicate recognizing the one-day-old sequences is as follows:

```
oneday          :: [Int] → Bool
oneday []       = True
oneday [a]      = False
oneday [a, b]   = True
```

$$\begin{aligned} &\mathit{oneday}\ [a, b, c] &&= \mathit{False} \\ &\mathit{oneday}\ (a : b : c : d : \mathit{xs}) &&= b \neq d \wedge \mathit{oneday}\ (c : d : \mathit{xs}) \end{aligned}$$

3.2. **The Two-Day Theorem.** The criterion given by the One-Day Theorem further restricts the possible sequences that can arise on the second day, because a one-day-old sequence *say xs* cannot have a run of more than three consecutive identical members. This in turn means that the even-indexed members of *say* (*say xs*) must be in the range $[1\,..\,3]$. This necessary condition does not fully characterize two-day-old sequences, but it will be enough for the purposes of this paper. For a proof of the necessity, and a complete characterization of two-day-old sequences, see Conway's paper.

Theorem 2 (Conway [Con87])**.** *The even-indexed members of a two-day-old sequence are in the range* $[1\,..\,3]$.

3.3. **The large-integer simulation.** We will need an additional observation that will restrict the set of integers involved in the sequences we consider to the range $[1\,..\,4]$. The observation applies to two-day-old sequences; namely, that each member $m \geqslant 4$ of a two-day-old sequence *xs* is in a run by itself, because the even-indexed members of the sequence are all in the range $[1\,..\,3]$ by the Two-Day Theorem. Because of this, each such m will be coded by the function *code* as a subsequence of the form $1m$ in *say xs*. Since all the members of *say xs* not arising in this way will be in the range $[1\,..\,3]$ by the One-Day Theorem, there is a correspondence between occurrences of these *large integers* $\geqslant 4$ in *xs* and *say xs*. For example, the two occurrences of 5 in 22251511 correspond to the two occurrences of 5 in its descendant 3215111521.

Now the *value* of any large integer m is irrelevant to the evolution of the rest of the sequence, because it is simply propagated into the descendant in being coded $1m$. For example, $222m1n11$ becomes $321m111n21$, $1312111m311n1211$, and so forth for any $m, n \geqslant 4$. For this reason, the evolution of an arbitrary two-day-old sequence can be simulated by the evolution of a similar sequence in which all the occurrences of large integers are replaced by 4. In the example, the simulating sequence is 22241411 and its descendants are 3214111421, 1312111431141211, and so on.

We define a set Sim of simulating sequences by the following Haskell predicate:

$$\begin{aligned} &\mathit{sim} &&:: [\mathit{Int}] \rightarrow \mathit{Bool} \\ &\mathit{sim}\ [\,] &&= \mathit{True} \\ &\mathit{sim}\ [a] &&= \mathit{False} \\ &\mathit{sim}\ [a, b] &&= a < 4 \\ &\mathit{sim}\ [a, b, c] &&= \mathit{False} \\ &\mathit{sim}\ (a : b : c : d : \mathit{xs}) &&= a < 4 \wedge (b \geqslant 4 \vee d \geqslant 4 \vee b \neq d) \wedge \mathit{sim}\ (c : d : \mathit{xs}) \end{aligned}$$

It is not hard to show that:

(1) every two-day-old sequence is simulated by a sequence in Sim;
(2) runs in a sequence in Sim have length at most 3;
(3) no large integer in a sequence in Sim is adjacent to any other; and
(4) if *xs* is in Sim then *say xs* is in Sim.

However, not every sequence in Sim is even a one-day-old sequence: for example, 1414 is in Sim but is only zero days old.

In the rest of the paper, many analyses will be focused on sequences in Sim. The results can then be carried over to arbitrary two-day-old sequences by observing that the results all concern the evolution of sequences under *say*, and by the above considerations, any arbitrary two-day-old sequence *xs* is simulated by a sequence in Sim (namely, the sequence obtained by replacing each member of *xs* greater than 3 by 4) under the iteration of *say*.

4. Abstract interpretation

This section introduces the method of abstract interpretation which is the keystone of the proofs presented in this paper. The method is then applied to the problem of deriving a decision procedure for splitting a sequence.

We begin with some observations about lists in Haskell. Looking at the definition of *isay*, we see that the list returned by *isay xs* is an *infinite list.* There is another special kind of list in Haskell, a *partial list* such as $1:1:2:\bot$. The symbol $\bot$ is shorthand for the Haskell expression *undefined.* Every list is either finite, partial, or infinite, and no list falls into more than one of these categories. (A list such as $[1,1,2,\bot]$ is just an ordinary finite list with a special *member.*)

Sometimes a computation will be able to complete without touching the undefined part of a partial list. (This is the essence of laziness.) For example:

$$\textbf{?}\ \mathit{take}\ 2\ (\mathit{say}\ (1:1:2:\bot))$$
$$[2,1]$$

The computational behavior of this example can be described completely by the equation

$$\mathit{say}\ (1:1:2:\bot) = 2:1:\bot$$

as may be proved easily by algebraic methods.

For us, the usefulness of these observations is that the function *say*, by the semantics of Haskell [Jon03], is monotone with respect to approximation. That is, computing *say* of any finite list extending $1:1:2:\bot$ must yield a finite list extending $2:1:\bot$, by the equation, monotonicity, and the observation that *say* maps finite lists to finite lists. The idea of this paper is to exploit this behavior as a form of abstract interpretation [CC77].

4.1. Covering sets.

As a first application of the method, let us determine how just the first (leftmost) part of a given sequence evolves upon iteration of *say*. We are going to try to understand what happens at the beginning of the list, ignoring the details of what happens after a certain point, so the above notion of abstract interpretation is appropriate.

Our method will be to evaluate *say* on a finite set C of finite and partial lists, having the property that every finite list is in C or extends one of the partial lists in C. In this case we say C *covers* all the finite lists. We can construct a C with this property using the following function:

```
cover   :: ([Int] → Bool) → [[Int]]
cover f = if f [] then [⊥]
               else [] : [x : xs | x ← [1 . . 4], xs ← cover (f ▷ x)]
  where f ▷ x = f ∘ (x:)
```

Here the function f serves as an oracle, indicating when the approximation has been sufficiently refined. For this reason we call f a *refinement predicate*. The local definition $f \rhd x = f \circ (x{:})$ serves to introduce a function $(\rhd)$ which applies a number as the head of the lists tested by a predicate, producing a new predicate. For example, if f is a predicate, then $f \rhd 1$ is the predicate which, given xs, returns $f\ (1 : xs)$.

The following are examples of the use of *cover*:

$$\begin{aligned}
&\mathit{cover}\ (==[\,]) &&= [\bot]\\
&\mathit{cover}\ ((\geqslant 1)\circ \mathit{length}) &&= [[\,], 1:\bot, 2:\bot, 3:\bot, 4:\bot]\\
&\mathit{cover}\ ((\geqslant 2)\circ \mathit{length}) &&= [[\,], [1], 1:1:\bot, 1:2:\bot, 1:3:\bot, 2:4:\bot,\\
& && \quad [2], 2:1:\bot, 2:2:\bot, 2:3:\bot, 3:4:\bot,\\
& && \quad [3], 3:1:\bot, 3:2:\bot, 3:3:\bot, 3:4:\bot,\\
& && \quad [4], 4:1:\bot, 4:2:\bot, 4:3:\bot, 4:4:\bot]\ .
\end{aligned}$$

It is not difficult to see that if *cover f* evaluates to a finite list, then that list constitutes a covering set C. We show this by induction on the number of steps of the evaluation. This is the number of steps that a Haskell interpreter will execute in computing the value *True* of the finiteness testing function *finite* (*cover f*):

$$\begin{aligned}
&\mathit{finite} &&:: [a] \to \mathit{Bool}\\
&\mathit{finite}\ [\,] &&= \mathit{True}\\
&\mathit{finite}\ (x : xs) &&= \mathit{finite}\ xs
\end{aligned}$$

Theorem 3. *If cover f evaluates to a finite list c, then the members of c constitute a covering set.*

Proof. The proof is by induction on the number of steps required to evaluate the spine of the list c; i.e., the number of steps required to evaluate *finite c* to *True*.

If $f\ [\,]$ evaluates to *True* then *cover f* evaluates to $[\bot]$, which is a covering set. If on the other hand $f\ [\,]$ evaluates to *False*, then it must be the case that $f \rhd 1$, $f \rhd 2$, $f \rhd 3$, and $f \rhd 4$ evaluate to finite lists C_1, C_2, C_3, and C_4, each of which is a covering set. But then

$$\mathit{cover}\ f = [\,] : \mathit{concat}\ [\mathit{map}\ (1{:})\ C_1, \mathit{map}\ (2{:})\ C_2, \mathit{map}\ (3{:})\ C_3, \mathit{map}\ (4{:})\ C_4],$$

which is a covering set. □

Unfortunately this notion will not yet allow us to investigate the behavior of *say* at the beginning of a sequence because of cases such as $1:1:...:1:\bot$ in which any number of members of the input list may need to be examined in order to determine even the first member of the output list. This makes it impossible to get useful information out of a covering set for *all* finite lists of integers in $[1\,..\,4]$.

However, if we reduce the space of lists with which we are concerned to just the ones in Sim, cases like $[1,1,1,1]$ cannot occur, because they are not in Sim. The easiest way of doing this is to generalize *cover* to take a *selection* predicate s, as follows:

```
$\mathit{cover}$     :: $([\mathit{Int}] \to \mathit{Bool}) \to ([\mathit{Int}] \to \mathit{Bool}) \to [[\mathit{Int}]]$
$\mathit{cover}\ s\ f$ = if $\neg\ (s\ [\,])$ then $[\,]$
             else if $f\ [\,]$ then $[\bot]$
             else $[\,] : [x : xs \mid x \leftarrow [1\,..\,4], xs \leftarrow \mathit{cover}\ (s \rhd x)\ (f \rhd x)]$
  where $f \rhd x = f \circ (x{:})$
```

This generalized definition of *cover* is the version used throughout the rest of the development.

We say that a selection predicate s is *acceptable* if it is defined on all finite lists of integers in $[1\,..\,4]$ and if it is prefix closed; that is, if $s\ (xs +\!\!+\ ys)$ implies $s\ xs$ when xs and ys are finite. A finite set C of finite and partial lists is now said to cover s when every finite list xs such that $s\ xs = \mathit{True}$ is in C or extends a partial list in C. The argument given above now establishes that if s is acceptable and $\mathit{cover}\ s\ f$ evaluates to a finite list, then that list constitutes a covering set for s.

Theorem 4. *If s is an acceptable selection predicate, and cover $s\ f$, as generalized, evaluates to a finite list c, then the members of c constitute a covering set for s.*

Proof. The previous argument goes through with the following modifications: We observe that by the prefix closed property of s, if $s\ [\,] = \mathit{False}$, then the covering set is allowed to be empty, since $s\ xs = \mathit{False}$ for any xs. We also observe that if s is prefix closed, $s \rhd n$ is prefix closed as well, so the induction hypothesis can be applied. □

The predicate *sim* is not acceptable because, for example, it rejects $[1]$ but accepts $[1, 1]$. It can be extended to an acceptable predicate by defining

$$\begin{array}{ll}
\mathit{simacc} & :: [\mathit{Int}] \to \mathit{Bool} \\
\mathit{simacc}\ [\,] & = \mathit{True} \\
\mathit{simacc}\ [a] & = a < 4 \\
\mathit{simacc}\ [a, b] & = a < 4 \\
\mathit{simacc}\ [a, b, c] & = a < 4 \wedge c < 4 \\
\mathit{simacc}\ (a : b : c : d : xs) & = a < 4 \wedge (b \geqslant 4 \vee d \geqslant 4 \vee b \neq d) \\
 & \quad \wedge\ \mathit{simacc}\ (c : d : xs)
\end{array}$$

We will only need the following facts concerning *simacc*: it is acceptable; $\mathit{sim}\ xs$ implies $\mathit{simacc}\ xs$; and *simacc* is sufficiently restrictive to reject unwanted sequences like $1 : 1 : ... : 1 : \bot$. We then have that if $\mathit{cover}\ \mathit{simacc}\ f$ is a finite list, it covers the sequences in Sim (as well as some additional sequences).

4.2. **The Starting Theorem.** We are now in a position to determine the behavior of *say* on the beginning part of a sequence. This relies on a trick: "finding the limit cycles".

We pick a particular refinement predicate (determined by trial and error) and form the set

$$c = \mathit{cover}\ \mathit{simacc}\ ((\geqslant 12) \circ \mathit{length})$$

covering Sim. The 20th iterates of *say* on the members of c are

$$c' = \mathit{map}\ ((!!20) \circ \mathit{isay})\ c$$

and we can look at the possible first parts of these by evaluating $\mathit{nub}\ (\mathit{map}\ (\mathit{take}\ 20)\ c')$, as shown in Figure 1. (Recall that *nub* removes duplicates from a list.) The parameters 12 and 20 were determined by experiment.

Now since this list is exhaustive, by the covering property, we see that the 20th iterate of *say* on any given list will start in one of the 14 ways given in the figure. Since the nth iterate for $n \geqslant 20$ is itself the 20th iterate of a sequence (namely, the $(n - 20)$-th iterate), it too must start in one of the ways given in the figure.

```
? nub (map (take 20) c′)
[[],
 [3,1,1,3,1,1,2,2,2,1,1,3,1,1,1,2,3,1,1,3],
 [1,3,2,1,1,3,2,1,3,2,2,1,1,3,3,1,1,2,1,3],
 [1,3,1,1,1,2,1,3,1,2,2,1,1,2,1,3,2,1,1,3],
 [3,1,2,3,2,1,1,2,3,1,1,3,2,1,3,2,2,1,1,2],
 [1,1,1,3,1,2,2,1,1,3,1,2,1,1,1,3,2,2,2,1],
 [2,2],
 [2,2,1,3,2,1,1,3,2,1,3,2,2,1,1,3,3,1,1,2],
 [2,2,1,3,1,1,1,2,1,3,1,2,2,1,1,2,1,3,2,1],
 [2,2,3,1,1,3,1,1,2,2,2,1,1,3,1,1,1,2,3,1],
 [2,2,3,1,2,3,2,1,1,2,3,1,1,3,2,1,3,2,2,1],
 [1,1,1,3,3,1,1,2,1,1,1,3,1,1,2,2,2,1,1,2],
 [2,2,1,1,1,3,1,2,2,1,1,3,1,2,1,1,1,3,2,2],
 [2,2,1,1,1,3,3,1,1,2,1,1,1,3,1,1,2,2,2,1]]
```

Figure 1. All possible ways a 20 day old sequence can start

So, in particular, supposing that the 20th iterate starts with 11131221131211132221, as in one of the lines of the figure, we can infer that the 21st iterate starts 311311..., hence, by inspection, the 21st iterate actually must start with 31131122211311123113, since that is the only possibility among the lines of the figure. Furthermore, the 22nd iterate then must start with 13211321322113311213, again by inspecting the possibilities, and then the 23rd iterate must start with 11131221131211132221 again, and so forth, leading to a cycle of period 3.

By continuing in this way we find that 20-day-old sequences must fall into one of 6 *limit cycles* given by the lines of the figure. Three of these are

$$
\begin{array}{l}
[\,] \Longrightarrow [\,] \Longrightarrow \ldots \\
[1,1,1,3,1,2,2,1,1,3,1,2,1,1,1,3,2,2,2,1,\ldots] \Longrightarrow \\
\quad [3,1,1,3,1,1,2,2,2,1,1,3,1,1,1,2,3,1,1,3,\ldots] \Longrightarrow \\
\quad [1,3,2,1,1,3,2,1,3,2,2,1,1,3,3,1,1,2,1,3,\ldots] \Longrightarrow \\
\quad [1,1,1,3,1,2,2,1,1,3,1,2,1,1,1,3,2,2,2,1,\ldots] \Longrightarrow \ldots \\
[1,1,1,3,3,1,1,2,1,1,1,3,1,1,2,2,2,1,1,2,\ldots] \Longrightarrow \\
\quad [3,1,2,3,2,1,1,2,3,1,1,3,2,1,3,2,2,1,1,2,\ldots] \Longrightarrow \\
\quad [1,3,1,1,1,2,1,3,1,2,2,1,1,2,1,3,2,1,1,3,\ldots] \Longrightarrow \\
\quad [1,1,1,3,3,1,1,2,1,1,1,3,1,1,2,2,2,1,1,2,\ldots] \Longrightarrow \ldots
\end{array}
$$

and the other 3 cycles are derived from these by prepending $[2, 2]$.

This is essentially the content of Conway's Starting Theorem [Con87]. We have proved it by direct calculation, using Haskell's own lazy semantics as a form of abstract interpretation. Furthermore, the function *say* acts as both the abstract interpreter and as the function being interpreted!

Theorem 5. *Any 20 day old sequence in Sim begins in one of the ways shown in Figure 1, and its further evolution must consist of sequences that start in such a way as to match one of the 6 limit cycles described above.*

We could prove the similar Ending Theorem concerning the behavior of *say* at the *end* of a sequence by defining a version of *say* acting on reversed lists; however, it is not needed in the development, and it will be an easy corollary of the Cosmological Theorem, below, so it is left to the reader to state and prove it.

4.3. **Splitting sequences.** As promised, we are now in a position to develop a decision procedure for splitting a sequence into subsequences which evolve independently under *say*. If

$$(isay\ xs\ !!\ n) = (isay\ ys\ !!\ n) +\!\!+ (isay\ zs\ !!\ n)$$

for finite lists xs, ys, zs and for all $n \geqslant 0$, then we say that xs splits into ys and zs. Since $(isay\ xs\ !!\ 0) = xs$, we have $xs = (ys +\!\!+ zs)$.

We can translate splitting into a Haskell predicate naïvely as follows:

$$\begin{aligned} &splits && :: [Int] \to [Int] \to Bool \\ &splits\ ys\ zs && = isay\ (ys +\!\!+ zs) == zipWith\ (+\!\!+)\ (isay\ ys)\ (isay\ zs) \end{aligned}$$

This is a semi-decision procedure; if ys and zs are not a splitting, then $splits\ ys\ zs = False$. But if ys and zs do constitute a splitting, it is not hard to show that $splits\ ys\ zs = \bot$. An example is $splits\ [2]\ [1,1,1]$, which runs forever when evaluated.

A sequence is called an *element* if it is non-empty and it does not split into non-empty subsequences. Elements are thus analogous to primes in the theory of numbers. Every sequence splits in a unique way into finitely many elements [Con87]. However, unlike in number theory, where no prime divides any other prime, it *is* possible for an element to appear as a subsequence of another element. For example, 1 and 11 are both elements, because $splits\ [1]\ [1] = False$. For this reason it is *not* true that every sequence is the *catenation* of finitely many elements in a unique way, because not every catenation of elements is a splitting of elements.

Our goal will be to investigate the elements. Our first task will be to develop a decision procedure for splitting. Conway's observation [Con87] is that xs and ys are a splitting just when the last member of $isay\ xs\ !!\ n$ is distinct from the first member of $isay\ ys\ !!\ n$ for all $n \geqslant 0$. Using this observation, the splitting test can be simplified to

$$\begin{aligned} splits\ ys\ zs = {} & null\ ys \lor null\ zs \lor \\ & and\ (zipWith\ (\neq)\ (map\ last\ (isay\ ys)) \\ & \qquad (map\ head\ (isay\ zs))) \end{aligned}$$

which is slightly more defined but is still not a decision procedure.

But by equational reasoning, for non-empty ys, $last\ (say\ ys) = last\ ys$ and so $map\ last\ (isay\ ys) = repeat\ (last\ ys)$. So *splits* can be simplified further to

$$splits\ ys\ zs = null\ ys \lor null\ zs \lor \neg\ (last\ ys \in map\ head\ (isay\ zs))$$

Finally, using the Starting Theorem, the members of $map\ head\ (isay\ zs)$ are exactly the members of $take\ 25\ (map\ head\ (isay\ zs))$ because by the 22nd day, zs will have reached one of the limit cycles, and the limit cycles have periods at most 3. Thus, we can rewrite *splits* into a decision procedure:

$$\begin{aligned} &\mathit{splits}\ \mathit{ys}\ \mathit{zs} = \mathit{null}\ \mathit{ys} \lor \mathit{null}\ \mathit{zs} \lor \\ &\qquad\qquad \neg\,(\mathit{last}\ \mathit{ys} \in \mathit{take}\ 25\ (\mathit{map}\ \mathit{head}\ (\mathit{isay}\ \mathit{zs}))) \end{aligned}$$

It is an easy consequence of this version of *splits* that a two-day-old sequence and its simulating sequence in Sim split into elements in the same way.

This version of *splits*, however, may examine much more of the list *zs* than is actually needed to determine the answer. In what follows it will be necessary to evaluate *splits* on covering sets of partial lists, and for the covering sets to have a feasible size, it is important that *splits* be as parsimonious as possible. Accordingly, I will exhibit another, more parsimonious function *splits′* (constructed by trial and error) and prove by abstract interpretation that it coincides with *splits* on sequences in Sim.

First we need to massage *splits* into a form amenable to abstract interpretation:

$$\begin{aligned} &\mathit{splits} && :: [\mathit{Int}] \to [\mathit{Int}] \to \mathit{Bool} \\ &\mathit{splits}\ \mathit{ys}\ \mathit{zs} && = \mathit{null}\ \mathit{ys} \lor \mathit{null}\ \mathit{zs} \lor \mathit{spl}\ (\mathit{last}\ \mathit{ys} : \mathit{zs}) \\ &\mathit{spl} && :: [\mathit{Int}] \to \mathit{Bool} \\ &\mathit{spl}\ (y : \mathit{zs}) && = \neg\,(y \in \mathit{take}\ 25\ (\mathit{map}\ \mathit{head}\ (\mathit{isay}\ \mathit{zs}))) \end{aligned}$$

Now *splits′* is introduced by

$$\begin{aligned} &\mathit{splits}' && :: [\mathit{Int}] \to [\mathit{Int}] \to \mathit{Bool} \\ &\mathit{splits}'\ \mathit{ys}\ \mathit{zs} && = \mathit{null}\ \mathit{ys} \lor \mathit{null}\ \mathit{zs} \lor \mathit{spl}'\ (\mathit{last}\ \mathit{ys} : \mathit{zs}) \end{aligned}$$

where $\mathit{spl}' :: [\mathit{Int}] \to \mathit{Bool}$ is defined in Appendix A. However, it is unnecessary to look at the definition of *spl′* because we are about to prove by direct calculation that it is correct.

In what follows we consider only suffixes of sequences in Sim. Now to show that *splits* and *splits′* coincide it suffices to show that *spl* and *spl′* coincide on finite lists of length $l \geqslant 2$, or equivalently that

$$\begin{aligned} &f\ [\,] && = \mathit{True} \\ &f\ [x] && = \mathit{True} \\ &f\ \mathit{xs} && = \mathit{spl}\ \mathit{xs} == \mathit{spl}'\ \mathit{xs} \end{aligned}$$

is *True* on all finite lists. We will establish this by an abstract interpretation. Here the selection predicate *simsuf* accepts suffixes of sequences in Sim (and a few additional sequences):

$$\begin{aligned} &\mathit{simsuf} && :: [\mathit{Int}] \to \mathit{Bool} \\ &\mathit{simsuf}\ \mathit{xs} && = \mathit{simacc}\ \mathit{xs} \lor \mathit{simacc}\ (\mathit{tail}\ \mathit{xs}) \end{aligned}$$

The abstract interpretation then proceeds as follows:

$$\begin{aligned} &\mathbf{?}\ \mathit{all}\ f\ (\mathit{cover}\ \mathit{simsuf}\ ((\geqslant 14) \circ \mathit{length} \circ \mathit{say})) \\ &\mathit{True} \end{aligned}$$

The refinement predicate $((\geqslant 14) \circ \mathit{length} \circ \mathit{say})$ was chosen by trial and error to make the interpretation complete in a reasonable amount of time.

Using *splits′* we can introduce a parsimonious function to split a sequence in Sim into its elements:

$$\begin{aligned} &\mathit{elements} && :: [\mathit{Int}] \to [[\mathit{Int}]] \\ &\mathit{elements}\ [\,] && = [\,] \\ &\mathit{elements}\ (x : \mathit{xs}) && = (x : \mathit{ys}) : \mathit{yss} \end{aligned}$$

$$\begin{aligned}&\textbf{where}\ \mathit{ys} : \mathit{yss}\\ &\quad \mid \mathit{spl}'\ (x : \mathit{xs}) = [\,] : \mathit{elements}\ \mathit{xs}\\ &\quad \mid \mathit{otherwise} \quad = \mathit{elements}\ \mathit{xs}\end{aligned}$$

The binding of $ys : yss$ is only evaluated when either ys or yss is demanded by subsequent computations. This in turn means that $spl'\ (x : xs)$ is only evaluated when ys or yss is needed. This makes *elements* more parsimonious than the alternative $elements'$ below in which the second case is defined in what might at first seem a more natural way:

$$\begin{aligned}&\mathit{elements}'\ (x : \mathit{xs})\\ &\quad \mid \mathit{spl}'\ (x : \mathit{xs}) = [x] : \mathit{elements}'\ \mathit{xs}\\ &\quad \mid \mathit{otherwise} \quad = (x : \mathit{ys}) : \mathit{yss}\\ &\quad \textbf{where}\ \mathit{ys} : \mathit{yss} = \mathit{elements}'\ \mathit{xs}\end{aligned}$$

4.4. **The Chemical Theorem.** Conway's development next proves an interesting result called the Chemical Theorem. This characterizes a certain special set of elements that are guaranteed to show up in any sufficiently late descendant of an arbitrary sequence other than the two "boring" sequences $[\,]$ and $[2, 2]$. This result is easily established in Haskell as follows.

First, we observe that by the Starting Theorem (Theorem 5) any non-boring sequence ends up in an limit cycle involving one of the following four kinds of sequences:

$$\begin{aligned}&[3, 1, 2, 3, 2, 1, 1, 2, 3, 1, 1, 3, 2, 1, 3, 2, 2, 1, 1, 2, ...]\\ &[2, 2, 3, 1, 2, 3, 2, 1, 1, 2, 3, 1, 1, 3, 2, 1, 3, 2, 2, 1, 1, 2, ...]\\ &[1, 3, 2, 1, 1, 3, 2, 1, 3, 2, 2, 1, 1, 3, 3, 1, 1, 2, 1, 3, ...]\\ &[2, 2, 1, 3, 2, 1, 1, 3, 2, 1, 3, 2, 2, 1, 1, 3, 3, 1, 1, 2, 1, 3, ...]\end{aligned}$$

In the first case, since

$$\mathit{head}\ (\mathit{elements}\ (3 : 1 : 2 : 3 : 2 : 1 : \bot)) = [3, 1, 2]$$

we see that some descendant of the sequence involves the element $[3, 1, 2]$. In the second case, we have

$$\mathit{take}\ 2\ (\mathit{elements}\ (2 : 2 : 3 : 1 : 2 : 3 : 2 : 1 : \bot)) = [[2, 2], [3, 1, 2]]$$

and so $[3, 1, 2]$ again must occur. By

$$\begin{aligned}&\mathit{head}\ (\mathit{elements}\ (1 : 3 : 2 : 1 : 1 : 3 : 2 : 1 : 3 : 2 : \bot)) = [1, 3, 2, 1, 1, 3, 2]\\ &\mathit{take}\ 2\ (\mathit{elements}\ (2 : 2 : 1 : 3 : 2 : 1 : 1 : 3 : 2 : 1 : 3 : 2 : \bot))\\ &\quad = [[2, 2], [1, 3, 2, 1, 1, 3, 2]]\end{aligned}$$

the element $[1, 3, 2, 1, 1, 3, 2]$ must occur in the third and fourth cases.

So any non-boring sequence must have a descendant containing 312 or 1321132 as an element. Because of the period 3 of the limit cycles involved, the element 312 or 1321132 must actually recur in every third descendant once it appears.

Now as these elements evolve, their descendants end up involving many more elements, which themselves must therefore occur in some descendant of every non-boring sequence. For example, starting from 312 we have the first evolution shown in Figure 2, and starting from 1321132 we have the second evolution shown in the figure, where the dots indicate how the sequences split into elements.

```
312
131112
11133112
312.32112
131112.13122112
11133112.111311222112
312.32112.31132.1322112
131112.13122112.13211312.1113222112
11133112.111311222112.11131221131112.3113322112
312.32112.31132.1322112.3113112221133112.132.123222112
...
1321132
111312211312
3113112221131112
1321132.13221133112
111312211312.1113222.12.32112
3113112221131112.311332.1112.13122112
1321132.13221133112.132.12.312.3112.111311222112
   ...
```

Figure 2. Audioactive decay starting from 312 and 1321132

Given an element, its descendant will split into some number of elements, their descendants will split further, and so on. In this way a directed graph is determined on all the elements. Defining

$$\begin{aligned}
&\mathit{fix} && :: \mathit{Eq}\ a \Rightarrow (a \to a) \to (a \to a)\\
&\mathit{fix}\ f\ x && = \mathbf{if}\ x == y\ \mathbf{then}\ x\ \mathbf{else}\ \mathit{fix}\ f\ y\\
& && \quad \mathbf{where}\ y = f\ x
\end{aligned}$$

we can compute a fixpoint over this process starting with a given element:

$$\begin{aligned}
&\mathit{fixelt} && :: [\mathit{Int}] \to [[\mathit{Int}]]\\
&\mathit{fixelt}\ xs && = \mathit{fix}\ f\ [xs]\\
&\quad \mathbf{where}\ f = \mathit{sort} \circ \mathit{nub} \circ \mathit{concat} \circ \mathit{map}\ (\mathit{elements} \circ \mathit{say})
\end{aligned}$$

We then verify in a Haskell interpreter the following:

$$\begin{aligned}
&\mathbf{?}\ \mathit{fixelt}\ [3,1,2] == \mathit{fixelt}\ [1,3,2,1,1,3,2]\\
&\mathit{True}\\
&\mathbf{?}\ \mathit{length}\ (\mathit{fixelt}\ [3,1,2])\\
&92
\end{aligned}$$

This establishes that the part of the graph reachable from 312 or from 1321132 consists of the same 92 elements. Conway calls these 92 elements the *common elements*, and assigns them symbols based on the symbols of the 92 chemical elements H–U.

```
commonelts :: [[Int]]
commonelts = fixelt [3, 1, 2]

common     :: [Int] → Bool
common     = (∈ commonelts)
```

Since as observed above 312 and 1321132 occur in some descendant of any given interesting sequence, every common element occurs in some descendant of the sequence. As a corollary, the graph of all the common elements but 22 is strongly connected.

This observation can be strengthened by computing the following infinite list:

```
? [[3, 1, 2] ∈ elements xs | xs ← isay [3, 1, 2]]
[True, False, False, True, False, False, True, False, False,
 True, False, False, True, False, False, True, True, False,
 True, True, False, True, True, True, True, True, True, ...
```

At first, 312 occurs only every third day, because of the period of its limit cycle. However, 312 can be reached in multiple ways through the graph of common elements; one of these ways is a cycle with period 16, and another is a cycle of period 23, as can be deduced by inspecting the data above. This means that once 312 occurs, it eventually ends up occurring every day. But then every common element also ends up occurring every day. This establishes the Chemical Theorem:

Theorem 6. *Every common element occurs in every sufficiently late descendant of any given interesting sequence.*

4.5. **The transuranic elements.** We would like to show that additionally, all sufficiently late descendants of a sequence involve *only* the common elements. However, this is obviously false, for example because every descendant of [4] must itself end with 4, but only the integers [1 . . 3] appear in the common elements.

Examining the evolution starting from [4] shown in Figure 3 (which suppresses commas), we may conjecture that eventually the last element involved in a descendant of $[n]$ for $n \geqslant 4$ must be one of the two so-called *transuranic elements*:

$$ {}^{n}\mathrm{Pu} = 31221132221222112112322211n $$
$$ {}^{n}\mathrm{Np} = 1311222113321132211221121332211n $$

The pairs of transuranic elements for each distinct n are called, of course, *isotopes.* As with the Ending Theorem, this conjecture can be established easily once the Cosmological Theorem has been proved.

The following Haskell predicate tests for the transuranic elements:

```
transuranic    :: [Int] → Bool
transuranic xs = last xs ⩾ 4 ∧ init xs ∈
  [[3, 1, 2, 2, 1, 1, 3, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 1, 1, 2, 3, 2, 2, 2, 1, 1],
   [1, 3, 1, 1, 2, 2, 2, 1, 1, 3, 3, 2, 1, 1, 3, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 3, 3, 2, 2, 1, 1]]
```

Note that a two-day-old sequence is a transuranic element if and only if the sequence in Sim that simulates it is transuranic.

5. The Cosmological Theorem

The final illustration of the abstract interpretation method will be the proof of a counterpart to the Chemical Theorem, namely, Conway's Cosmological Theorem.

```
? map (last ∘ elements) (isay [4])
[[4],
 [14],
 [1114],
 [3114],
 [132114],
 [1113122114],
 [311311222114],
 [1322114],
 [1113222114],
 [3113322114],
 [123222114],
 [111213322114],
 [31121123222114],
 [132112211213322114],
 [111312212221121123222114],
 [3113112211322112211213322114],
 [1321132122211322212221121123222114],
 [11131221131211322113321132211221121332211 4],
 [312211322212221121123222114],
 [13112221133211322112211213322114],
 [312211322212221121123222114],
 [13112221133211322112211213322114],
      ...
```

FIGURE 3. Evolution of [4], showing only the last element at each step

It states that every sufficiently late descendant of every sequence involves only common and transuranic elements. We say that every sequence eventually decays into a compound of common and transuranic elements.

The theorem was originally proved by Conway and Richard Parker on the basis of extensive hand calculations enumerating the cases. Mike Guy also found a simpler proof involving hand enumeration of cases, leading to the tight bound 24 on the number of days before an arbitrary sequence is guaranteed to have fully decayed. Both these proofs were said to have occupied many pages—Conway calls the theorem "ASTONISHINGLY hard to prove"—which were subsequently lost.

The proof given here will establish a weaker bound, but it can be improved to recreate the tight bound by improving the selection predicates, at some cost in perspicuity.

The overall concept for the proof of the Cosmological Theorem is to use abstract interpretation to calculate all the elements that might occur in sufficiently late descendants of an arbitrary sequence. Since the calculation has to involve only finitely many approximations, three different forms of abstraction are used to reduce the space of possible sequences to be considered.

The first form of abstraction reduces the space by restricting the members of the sequences to just the integers [1 . . 4] using the large-integer simulation described in Section 3.3. The evolution of an arbitrary two-day-old sequence can be related to the evolution of the corresponding sequence in Sim with all the members $m \geqslant 4$ replaced by 4.

The second form of abstraction allows the content of the sequence beyond (to the right of) a point of interest to be ignored; this is implemented by the laziness-based abstraction of the kind we have seen already. The sequence beyond a certain point is represented by Haskell's $\perp$.

Finally, the third form of abstraction allows the content of a sequence before (to the left of) a point of interest to be ignored. This is achieved by annotating a sequence with *marks* that are propagated to the sequence's descendants by a generalized version of *say*. This section introduces the theory of marks and explains how they can be used to abstract away the initial part of a sequence. Marks are an original contribution of the present paper.

5.1. **Marked sequences.** A *marked sequence* is a finite sequence of positive integers each of which is either annotated with a mark or left unmarked, and satisfying a certain condition. For simplicity, the Haskell development will represent unmarked members by positive integers and marked members by the corresponding negative integers. When presenting the sequences in condensed format in the text, an overbar $\overline{n}$ is used.

The condition on the marks is that in any run of consecutive identical members, at most one of them is to be marked. So for example, $1\overline{2}23\overline{3}$ is a properly marked sequence, but $1\overline{2}\overline{2}3\overline{3}$ is not.

We can define a function *unmark* taking a marked sequence to its corresponding unmarked one:

$$
\begin{aligned}
&\mathit{unmark} :: [\mathit{Int}] \to [\mathit{Int}] \\
&\mathit{unmark} = \mathit{map}\ \mathit{abs}
\end{aligned}
$$

The generalized version of *say* is a function *gsay*. It is a refinement of *say* in the sense that if $\mathit{gsay}\ \mathit{xs} = \mathit{ys}$ then $\mathit{say}\ (\mathit{unmark}\ \mathit{xs}) = \mathit{unmark}\ \mathit{ys}$. In the original *say*, a run of consecutive identical members such as 555 is read "three fives" and encoded 35 in the output sequence. For *gsay*, this is how an unmarked run is coded, and a marked run such as $5\overline{5}5$ is coded by $3\overline{5}$, propagating the mark onto the odd-indexed member of the output sequence.

$$
\begin{aligned}
&\mathit{gsay} && :: [\mathit{Int}] \to [\mathit{Int}] \\
&\mathit{gsay} && = \mathit{concat} \circ \mathit{map}\ \mathit{gcode} \circ \mathit{gruns} \\
&\mathit{gcode} && :: [\mathit{Int}] \to [\mathit{Int}] \\
&\mathit{gcode}\ \mathit{xs} && = \mathbf{if}\ \mathit{ismarked}\ \mathit{xs}\ \mathbf{then}\ [\mathit{length}\ \mathit{xs}, -(\mathit{abs}\ (\mathit{head}\ \mathit{xs}))] \\
& && \quad\ \mathbf{else}\ [\mathit{length}\ \mathit{xs}, \mathit{head}\ \mathit{xs}] \\
&\mathit{ismarked} && :: [\mathit{Int}] \to \mathit{Bool} \\
&\mathit{ismarked} && = \mathit{any}\ (<0)
\end{aligned}
$$

The helper function *gruns* differs from *runs* in order to be more parsimonious. It takes advantage of the observations about Sim in Section 3.3 to avoid looking unnecessarily far for the end of a run of consecutive identical members. This does no harm because *gsay* will only be used on sequences in Sim.

$$
\begin{aligned}
&\mathit{gruns} && :: [\mathit{Int}] \to [[\mathit{Int}]] \\
&\mathit{gruns}\ (a : b : \mathit{xs}) && = \mathbf{if}\ \mathit{abs}\ a \neq \mathit{abs}\ b\ \mathbf{then}\ [a] : \mathit{gruns}\ (b : \mathit{xs})
\end{aligned}
$$

```
                          else case $xs$ of
                            $c : ys \to$ if $abs\ b \neq abs\ c$ then $[a, b] : gruns\ xs$
                              else $[a, b, c] : gruns\ ys$
                            $[\,] \to [[a, b]]$
$gruns\ [a]$         $= [[a]]$
$gruns\ [\,]$         $= [\,]$
```

An example evolution of a marked sequence is:

$$\begin{array}{l}
\overline{131} \\
1\overline{1}1\overline{3}1\overline{1} \\
3\overline{1}1\overline{3}2\overline{1} \\
132\overline{1}1\overline{3}121\overline{1} \\
1113122\overline{1}1\overline{3}11122\overline{1} \\
311311222\overline{1}1\overline{3}312211\overline{1} \\
\ldots
\end{array}$$

It can be shown that the result of *gsay* on a marked sequence in Sim is a properly marked sequence; in particular, each run of consecutive identical elements again has at most one mark. It is also not difficult to see that the number of marks remains constant throughout the evolution. By relating corresponding marked members of a sequence and its descendants, a sort of coordinate system for the parts of the sequence can be maintained throughout the evolution. This allows the common features of evolutions starting from different sequences to be abstracted.

For example, the above evolution for $\overline{131}$ may be compared to the evolution beginning with $3\overline{31}$:

$$\begin{array}{ll}
\overline{131} & 3\overline{31} \\
1\overline{1}1\overline{3}1\overline{1} & 2\overline{3}1\overline{1} \\
3\overline{1}1\overline{3}2\overline{1} & 121\overline{3}2\overline{1} \\
132\overline{1}1\overline{3}121\overline{1} & 1112111\overline{3}121\overline{1} \\
1113122\overline{1}1\overline{3}11122\overline{1} & 3112311\overline{3}11122\overline{1} \\
311311222\overline{1}1\overline{3}312211\overline{1} & 13211213211\overline{3}312211\overline{1} \\
\ldots & \ldots
\end{array}$$

While the corresponding descendants from the two evolutions differ, their suffixes starting with the member $\overline{3}$ coincide. This phenomenon allows the part of a sequence to the left of a point of interest to be abstracted away.

Accordingly, we have the Mark Abstraction Theorem:

Theorem 7. *If xs and ys have a common suffix zs the first member of which is marked, then $gsay\ xs$ and $gsay\ ys$ again have a common suffix zs' the first member of which is marked, and the number of marked members of zs and zs' is the same.*

Proof. If we establish the special case when $ys = zs$, then the theorem in full generality follows easily. In order to prove the special case, we take $zs' = tail\ (gsay\ zs)$ and observe that the marks of zs and of $gsay\ zs$ are in correspondence by the behavior of *gsay*, and that the head of $gsay\ zs$ is not marked. □

It will also be convenient to have a function transforming an unmarked sequence in Sim into a canonically marked one:

$$
\begin{aligned}
&\mathit{mark} && :: [\mathit{Int}] \to [\mathit{Int}] \\
&\mathit{mark}\ [\,] && = [\,] \\
&\mathit{mark}\ (x : \mathit{xs}) && = x : \mathit{mark}'\ \mathit{xs} \\
&\mathit{mark}'\ [\,] && = [\,] \\
&\mathit{mark}'\ (x : \mathit{xs}) && = (-x) : \mathit{mark}\ \mathit{xs}
\end{aligned}
$$

This sequence is properly marked because all the marked members come from the odd-indexed subsequence, hence if two marks were to belong to a run, they would have to look like $\overline{1}1\overline{1}$, $\overline{2}2\overline{2}$, $\overline{3}3\overline{3}$, or $\overline{4}4\overline{4}$, all of which are impossible for sequences in Sim by the remarks in Section 3.3.

5.2. **Proof of the Lost Cosmological Theorem.** At this point all the tools are in hand to find a set *collect* of elements such that every sequence eventually decays into elements all of which are in *collect*. Having found it we will then see that every element in *collect* decays into common and transuranic elements, proving the Cosmological Theorem.

Suppose a sequence xs is given. Its two-day-old descendant $xs_2 = \mathit{say}\ (\mathit{say}\ xs)$ is simulated by a sequence ys in Sim, which we canonically mark, giving a sequence $zs = \mathit{mark}\ ys$. Now suppose an element occurs in, say, the 10th descendant of xs. Then it is related by the large-integer simulation to an element in the 8th generalized descendant ds of zs. This element of ds occurs in a shortest suffix ds' of ds starting with a mark, or in $ds' = ds$ if there is no mark to the left of it. Thus the element occurs in the 8th generalized descendant of the corresponding suffix zs' of zs, by the Mark Abstraction Theorem, and does so with at most one mark to its left.

This shows that in order to find every element that can occur in the 10th descendants of an arbitrary sequence, it suffices to find every element occuring in an 8th descendant of a canonically marked sequence in Sim, such that there is at most one mark to the left of the element's occurrence.

For example, starting from the sequence 111213 in Sim, we compute its canonical marking $1\overline{1}1\overline{2}1\overline{3}$, then proceed with the generalized evolution through eight more steps (with the splittings into elements indicated):

$$
\begin{array}{l}
1\overline{1}1\overline{2}\,.\,1\overline{3} \\
3\overline{1}1\overline{2}\,.\,111\overline{3} \\
132\overline{1}1\overline{2}\,.\,311\overline{3} \\
1113122\overline{1}1\overline{2}\,.\,13211\overline{3} \\
311311222\overline{1}1\overline{2}\,.\,111312211\overline{3} \\
1321132\,.\,1322\overline{1}1\overline{2}\,.\,31131122211\overline{3} \\
111312211312\,.\,1113222\overline{1}1\overline{2}\,.\,1321132\,.\,132211\overline{3} \\
3113112221131112\,.\,3113322\overline{1}1\overline{2}\,.\,111312211312\,.\,111322211\overline{3}
\end{array}
$$

Now since 111312211312 occurs with *two* marks to its left, it must also occur in the evolution of a shorter sequence, namely:

$$\begin{array}{l}
1\overline{2}\,.\,1\overline{3}\\
111\overline{2}\,.\,111\overline{3}\\
311\overline{2}\,.\,311\overline{3}\\
13211\overline{2}\,.\,13211\overline{3}\\
111312211\overline{2}\,.\,111312211\overline{3}\\
31131122211\overline{2}\,.\,31131122211\overline{3}\\
1321132\,.\,132211\overline{2}\,.\,1321132\,.\,132211\overline{3}\\
111312211312\,.\,11132221\overline{12}\,.\,111312211312\,.\,111322211\overline{3}
\end{array}$$

This evolution looks quite different but it again contains the element 111312211312 in the final descendant, this time with only a single mark to its left.

The Haskell code implementing this abstract interpretation is as follows:

```
collect     :: [[Int]]
collect     = nub (concat (map gather (cover simacc oracle)))
gather      :: [Int] → [[Int]]
gather      = takeelts ∘ gsay8 ∘ mark
gsay8       :: [Int] → [Int]
gsay8       = gsay ∘ gsay ∘ gsay ∘ gsay ∘ gsay ∘ gsay ∘ gsay ∘ gsay
takeelts xs = case findIndices (<0) xs of
                 (_ : n : _) → g (f n)
                 _ → ls
              where ls = elements (unmark xs)
                 f n = find ((⩾ n) ∘ length ∘ concat) (inits ls)
                 g (Just x) = x
```

It turns out that the function *oracle* selecting the covering set for the abstract interpretation is rather complex, because it must look ahead to see how long a partial list is needed in order to ensure that its 8th descendant, also a partial list, has at least two marks. Fortunately, the code for *oracle* is irrelevant to the correctness of the abstract interpretation, as long as it terminates. The code is therefore given in Appendix B.

The first few elements in *collect* turn out to be the following:

```
? take 5 collect
[[3, 1, 1, 3, 1, 2],
 [1, 1, 1, 3, 1, 2, 2, 1],
 [1, 3, 2, 1, 1, 3, 1, 1, 1, 2],
 [3, 1, 1, 3, 1, 1, 2, 2, 1, 1],
 [1, 3, 2, 1, 1, 3, 2]]
```

It turns out that there are many non-common elements in *collect*; the first is *collect* !! 10 = [1, 3, 2, 2, 1, 1, 3, 3, 1, 2, 2, 2, 1, 1, 3, 1, 1, 1, 2], which takes 4 further iterations of *say* to decay into common elements:

```
? common (collect !! 10)
False
```

$$\textbf{?}\ \textit{map common}\ (\textit{elements}\ (\textit{say}\ (\textit{say}\ (\textit{say}\ (\textit{say}\ (\textit{collect}\ !!\ 10))))))$$
$$[\textit{True}, \textit{True}, \textit{True}, \textit{True}, \textit{True}, \textit{True}, \textit{True}, \textit{True}, \textit{True}]$$

Having collected all the possible elements appearing in 10-day-old sequences, it remains to test that each of them decays into common and transuranic elements. We call a sequence in Sim *cosmological* if it decays into common and transuranic elements:

$$\begin{aligned} &\textit{cosmological} :: [\textit{Int}] \to \textit{Bool} \\ &\textit{cosmological} = \textit{any}\ (\textit{all}\ f \circ \textit{elements}) \circ \textit{isay} \\ &\qquad\qquad \textbf{where}\ f\ \textit{xs} = \textit{common xs} \vee \textit{transuranic xs} \end{aligned}$$

Of course this is only a semi-decision procedure; *cosmological xs* is *True* if *xs* is cosmological, but $\perp$ otherwise.

Finally, we can establish the Cosmological Theorem simply by testing each of the elements in *collect*:

$$\textbf{?}\ \textit{all cosmological collect}$$
$$\textit{True}$$

Thus we have the following:

Theorem 8. *Every sequence eventually decays into a compound of common and transuranic elements.*

By finding the principal eigenvector of the graph of common elements, Conway then proves that no matter what interesting sequence one starts with, asymptotically the number of occurrences of the various common and transuranic elements tend to certain fixed ratios, their *elemental abundances.* One can also compute the corresponding eigenvalue, *Conway's constant* $\lambda = 1.3035772690\ldots$, which is the asymptotic rate of growth of every interesting sequence.

6. Conclusions

This paper has introduced a new kind of abstract interpretation based on the denotational semantics of Haskell, and applied it to three different problems that arise in the theory of Conway's audioactive decay, leading up to and including the proof of his Cosmological Theorem.

Audioactive decay is an amusing mathematical recreation, but hopefully there are other applications of the method. The technique described is applicable to general datatypes, not just lists, and it could be developed in other languages besides Haskell.

From a higher level point of view, this work seems to lie in the middle ground between the use of the computer essentially as a labor-saving calculator and its use as a machine for creating formal deductions in a logic. The former sort of use is exemplified by the proofs of the four color theorem by Appel et al. [AHK77] and of Kepler's conjecture by Hales [Hal98, Hal02]. These proofs rely on careful checking of a complex computer program to establish the correctness of the result. The latter use, on the other hand, is exemplified by Hales' more recent project to formally prove the Kepler conjecture in HOL Light [Hal06]. The proof by formal deduction is certainly ultimately the most convincing, but it can require a significantly higher expenditure of effort. It might be hoped that techniques such as the one described in this paper could provide some additional confidence, by reducing the amount of code that must be checked, at moderate effort.

As a simple measure of the complexity of the computer proofs of the Cosmological Theorem, we may consider the number of lines of code involved. The source file of this technical report is a literate Haskell program containing 181 lines of Haskell code, of which only 98 lines are in the body of the report; the other 83 lines are in the appendices, and need not be considered when establishing the correctness of the code. By comparison, Zeilberger's Maple proof [EZ97] contains 2234 lines of code (including some self-documentation), and Litherland's C proof [Lit03b, Lit03a] contains 1650 lines of code (including some comments). Even accounting for the fraction of these totals taken up by documentation, the preceding proofs involve substantially more code, more complex code, and code written in languages harder to reason about, than the 98-line proof presented in this report.

References

[AHK77] Kenneth Appel, Wolfgang Haken, and John Koch. Every planar map is four colorable. *Illinois Journal of Mathematics*, 21:439–567, December 1977.

[BdM96] Richard Bird and Oege de Moor. *The Algebra of Programming*. Prentice–Hall, 1996. Available at `http://www.comlab.ox.ac.uk/oucl/publications/books/algebra/`.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[Con87] J. H. Conway. The weird and wonderful chemistry of audioactive decay. In T. M. Cover and B. Gopinath, editors, *Open Problems in Communications and Computation*, pages 173–188. Springer Verlag, New York, 1987.

[EZ97] Shalosh B. Ekhad and Doron Zeilberger. Proof of Conway's lost cosmological theorem. *Electronic Research Announcements of the American Mathematical Society*, 3:78–82, August 1997. arXiv math.CO/9808077.

[Hal98] Thomas C. Hales. The Kepler conjecture. arXiv math.MG/9811078, 1998.

[Hal02] Thomas C. Hales. A computer verification of the Kepler conjecture. *Proceedings of the ICM*, 3:793–804, 2002. arXiv math.MG/0305012.

[Hal06] Thomas C. Hales. The flyspeck project fact sheet. Available at `http://www.math.pitt.edu/~thales/flyspeck/index.html`, December 2006.

[Jon03] Simon Peyton Jones, editor. *The Haskell 98 Language and Libraries: the Revised Report*. Cambridge University Press, 2003. Available at `http://haskell.org/onlinereport/`.

[Lit03a] Richard A. Litherland. The Audioactive package. Available at `http://www.math.lsu.edu/~lither/jhc/`, April 2003.

[Lit03b] Richard A. Litherland. Conway's cosmological theorem. Available at `http://www.math.lsu.edu/~lither/jhc/`, April 2003.

Appendix A. Oracle for splitting into elements

```
$spl'$             $:: [Int] \to Bool$
$spl'\ (1 : [\,]) = True$
$spl'\ (1 : 2 : 2 : xs)$      $= spl''\ xs$
$spl'\ (2 : xs)$              $= spl''\ xs$
$spl'\ (3 : [\,])$            $= True$
$spl'\ (3 : 2 : 2 : xs)$      $= spl''\ xs$
$spl'\ (4 : 4 : \_)$          $= False$
$spl'\ (4 : \_)$              $= True$
$spl'\ \_$                    $= False$

$spl''$            $:: [Int] \to Bool$
$spl''\ (1 : 1 : 1 : \_)$     $= True$
$spl''\ (1 : [\,])$           $= False$
$spl''\ (1 : 1 : \_)$         $= False$
$spl''\ (1 : 2 : 2 : \_)$     $= False$
$spl''\ (1 : 3 : 3 : \_)$     $= False$
$spl''\ (1 : 4 : 4 : \_)$     $= False$
$spl''\ (2 : \_)$             $= False$
$spl''\ (3 : 1 : 1 : 1 : \_)$ $= False$
$spl''\ (3 : 2 : 2 : 2 : \_)$ $= False$
$spl''\ (3 : 3 : \_)$         $= False$
$spl''\ (3 : 4 : 4 : 4 : \_)$ $= False$
$spl''\ (4 : 4 : 4 : \_)$     $= True$
$spl''\ (4 : 4 : \_)$         $= False$
$spl''\ \_$                   $= True$
```

Appendix B. Oracle for the Cosmological Theorem

```
$oracle$            $:: [Int] \to Bool$
$oracle$            $= enoughelts \circ tsay8 \circ mark$

$tsay$              $:: [Int] \to [Int]$
$tsay$              $= concat \circ map\ gcode \circ truns$

$truns$             $:: [Int] \to [[Int]]$
$truns\ (a : b : xs)$ $=$ if $abs\ a \neq abs\ b$ then $[a] : truns\ (b : xs)$
                      else case $xs$ of
                         $c : ys \to$ if $abs\ b \neq abs\ c$ then $[a, b] : truns\ xs$
                            else $[a, b, c] : truns\ ys$
                         $[\,] \to [\,]$
$truns\ [a]$        $= [\,]$
$truns\ [\,]$       $= [\,]$

$tsay8$             $:: [Int] \to [Int]$
$tsay8$             $= tsay \circ tsay \circ tsay \circ tsay \circ tsay \circ tsay \circ tsay \circ tsay$

$enoughelts$        $:: [Int] \to Bool$
```

```
enoughelts xs      = case findIndices (<0) xs of
                         (_ : n : _) → length (concat (init ls)) ⩾ n
                         _ → False
                     where ls = telements (unmark xs)

telements          :: [Int] → [[Int]]
telements []       = [[]]
telements (x : xs) = (x : ys) : yss
                       where ys : yss
                               | tspl′ (x : xs) = [] : telements xs
                               | otherwise      = telements xs

tspl′                  :: [Int] → Bool
tspl′ (1 : 2 : 2 : xs)  = tspl″ xs
tspl′ (2 : xs)          = tspl″ xs
tspl′ (3 : 2 : 2 : xs)  = tspl″ xs
tspl′ [4]               = False
tspl′ (4 : _) = True
tspl′ _                 = False

tspl″                  :: [Int] → Bool
tspl″ (1 : 1 : 1 : _)   = True
tspl″ []                = False
tspl″ (1 : [])          = False
tspl″ (1 : 1 : _)       = False
tspl″ (1 : 2 : [])      = False
tspl″ (1 : 2 : 2 : _)   = False
tspl″ (1 : 3 : [])      = False
tspl″ (1 : 3 : 3 : _)   = False
tspl″ (2 : _)           = False
tspl″ (3 : [])          = False
tspl″ (3 : 1 : [])      = False
tspl″ (3 : 1 : 1 : [])  = False
tspl″ (3 : 1 : 1 : 1 : _) = False
tspl″ (3 : 2 : [])      = False
tspl″ (3 : 2 : 2 : [])  = False
tspl″ (3 : 2 : 2 : 2 : _) = False
tspl″ (3 : 3 : _)       = False
tspl″ _                 = True
```