Abstract Interpretation Using Laziness: Proving Conway's Lost Cosmological Theorem

Kevin Watkins

CMU CSD POP Seminar
December 8, 2006

In partial fulfillment
of the speaking skills requirement

?

2111

1231

11121311

3112111321

13211231131211

1113122112132113111221

3113112221121113122113312211

13211321322112311311222123112221

Look and say

- Read string out loud
  - 2111 → "one two, three ones" → 1231
  - 1231 → "one one, one two, one three, one one" → 11121311
  - ... etc.
- Invented by John Conway
  - paper: *The Weird and Wonderful Chemistry of Audioactive Decay*, 1987
  - also invented Life and surreal numbers
- So what happens?
  - asymptotics of string length
  - random looking? patterns in strings?
  - answered by the *Cosmological Theorem* ...

“Cosmological Theorem”

Characterizes how strings evolve

- split strings into *elements*
- elements *decay* into other elements
- *classify* all elements surviving in arbitrarily late strings

Proofs:

- J. H. Conway and friends, 1987 (by hand, lost)
- D. Zeilberger, 1997 (Maple)
- Litherland, 2003 (C)
- Watkins, 2006 (Haskell)

All proofs by exhaustion of cases

What is a proof?

Are these proofs?

- hand proof: case left out? too much paper?
- computer proof: buggy code? cosmic ray?

My strategy

- semantics of Haskell is my deductive system
- proof correct by construction!
- most code needn't be checked!
- key technique: Haskell lazy data as abstract interpretation

Vs deductive engine (e.g. Coq, HOL Light)

- their pros: correctness entirely formally verified
- my pros: *much* less effort; easy to experiment in Haskell interpreter

Other proofs with similar structure

In general, must:

- enumerate cases
- verify a property for each case

Examples:

- Four color theorem (Appel et al., 1976)
- Kepler's conjecture (Hales, 1998)

My method:

- *oracle strategy* simplifies showing sufficiency of enumeration
- *abstract interpretation via laziness* simplifies verifying property of each case
- This talk will illustrate both aspects

My contributions

New proof presentation strategy

- i.e. *oracle strategy* and *abstract interpretation via laziness* from previous slide
- may apply to similar proofs in other domains

Verify Conway's result

- Simple code: presented and justified in its entirety in my technical report
- Code written in a language with a simple semantics

Simplify prior proofs of Conway's result

- via my *marked sequences* (see technical report)

Talk outline

- Introduction
- Overview of Cosmological Theorem
- About Haskell and laziness
- Applying my method
- Conclusions

Cosmological Theorem

Recall:

Characterizes how strings evolve

- split strings into *elements*
- elements *decay* into other elements
- *classify* all elements surviving in arbitrarily late strings

Split strings into elements

Split string into parts that evolve independently

Formally:

- let xs_n be nth string in evolution
- xs splits into ys . zs if and only if:

  $xs_n = ys_n ++ zs_n$ for all $n \geq 0$

- (++) means append strings

Element: string that doesn't split into smaller ones

- Theorem (Conway, easy): every string splits into finitely many elements in unique way

Decision procedure for splitting?

- I'll tell you in a few minutes ...

Split strings into elements

Split up strings into parts that evolve independently

2111

1231

11121311

3112111321

13211231131211

111312211213211311221

3113112221121113122113312211

13211321322112311311222123112221

... etc.

Split strings into elements

Split up strings into parts that evolve independently

2 . 111

12 . 31

1112 . 1311

3112 . 111321

132112 . 31131211

1113122112 . 132113111221

311311222112 . 1113122113312211

13211321322112 . 3113112221231112221

... etc.

Split strings into elements

Example:

- 2111 splits into 2 . 111 which are elements
- first step: 2111 → 1231, and 2 . 111 → 12 . 31. OK!
- I claim happens for nth step, all n ≥ 0 (proof later!)

Counterexample:

- 111 *does not* split into 1 . 11:
- 111 → 31, but 1 . 11 → 11 . 21. BAD!

Analogy: factoring integer into primes?

- *Note* substring of element can be element
- e.g. 111 is element, and also 1
- so splitting into elements is context dependent

Cosmological Theorem

Recall:

Characterizes how strings evolve

- split strings into *elements*
- elements *decay* into other elements
- *classify* all elements surviving in arbitrarily late strings

Audioactive decay

Start with string

- split into elements
- do look and say
- split result into elements
- do look and say
- ... repeat ad infinitum

Audioactive decay

2 . 111

12 . 31

1112 . 1311

3112 . 111321

132112 . 31131211

1113122112 . 132113111221

311311222112 . 1113122113312211

1321132 : 1322112 . 311311222 : 12 : 3112221

... etc.

Cosmological Theorem

Recall:

Characterizes how strings evolve

- split strings into *elements*
- elements *decay* into other elements
- *classify* all elements surviving in arbitrarily late strings

Two special sets of elements:

- 92 *common elements*
- 2 infinite families of *transuranic elements*

Cosmological Theorem (Conway, proof lost): every string eventually decays into a compound of common and transuranic elements

Common elements

92 special elements

- Conway assigned them symbols H-U from chemistry
- involve only integers 1, 2, 3

Ubiquity

- Theorem (Conway): every common element eventually shows up in the decay of any *interesting* string (proved in technical report, not needed for Cosmological Theorem)

Two special cases

- empty string, 22 just repeat themselves
- call these *boring*, any other string *interesting*

Common elements example

2 . 111

12 . 31

1112 . 1311

3112 . 111321

132112 . 31131211

1113122112 . 132113111221

311311222112 . 1113122113312211

1321132 : 1322112 . 311311222 : 12 : 3112221

“holmium-silicon-erbium-calcium-antinide!”

Common elements example

2 . 111

Ca . 31

K . 1311

Ar . 111321

Cl . 31131211

S . 132113111221

P . 111312211312211

Ho : Si . Er : Ca : Sb

"holmium-silicon-erbium-calcium-antinide!"

Transuranic elements

What about integers other than 1 2 3?

- extra *transuranic elements*
- ^{n}Pu = 31221132221222112112322211n
- ^{n}Np = 1311222113321132211221121332211n

Cosmological Theorem (Conway, proof lost): Every string eventually decays into a compound of common and transuranic elements

Using the Cosmological Theorem

- How long do strings get?
  - make transition matrix on 92 common elements
  - find principal eigenvalue $\lambda = 1.3035772690...$
  - Theorem (linear algebra, not hard): length of any *interesting* string tends to $c\lambda^n$ on nth step, as $n \to \infty$
- Can also compute asymptotic relative abundance of elements
  - abundances of transuranic elements tend to 0

My proof

First step proving Cosmological Theorem: decision procedure for splitting strings into elements

This talk:

- develop correct decision procedure:
  - use *oracle strategy* to enumerate cases
  - use *abstract interpretation* to prove each case correct

See paper for the rest, leading ultimately to the proof of the Cosmological Theorem

- two more distinct uses of oracles and abstract interpretation

Splitting into elements

2111 splits into 2 and 111

But 111 doesn't split into, say, 1 and 11. Why?

- Split point is in the middle of a run of 1s
- Run 111 coded as 31 in original string
- Pieces 1 and 11 coded as 11 and 21 in split parts
- $31 \neq 1121$

Compare 2 and 111...

Splitting into elements

Compare 2 and 111...

2 . 111

12 . 31

1112 . 1311

3112 . 111321

132112 . 31131211

1113122112 . 132113111221

... etc.

Split point never lands in the middle of a run

Splitting into elements

... Split point never lands in the middle of a run

Otherwise said:

- last number of left part ≠ first number of right part, forever
- Theorem (Conway, easy): necessary and sufficient for splitting
- note last number of left part never changes

Plan: see what happens to first number of arbitrary string

Talk outline

- Introduction
- Overview of Cosmological Theorem
- About Haskell and laziness
- Applying my method
- Conclusions

Lists in Haskell

List is either:

- Empty list, written [], read “nil”
- Non-empty list, written (x:xs), read “x cons xs”
  - First element x (head)
  - List of remaining elements xs (tail)

Other syntax

- Cons associates to right: (1:2:3:[]) = (1:(2:(3:[])))
- Bracket abbreviation: [1,2,3] = (1:(2:(3:[])))

Haskell programming

Defining functions

- write equations characterizing function
- when function is called:
  - match pattern on left side of equation
  - result is right hand side of equation

Example: length of list

```
length [] = 0
length (x:xs) = length xs + 1
```

This talk: patterns always mutually exclusive

- e.g. [] and (x:xs) never both match an input

Haskell reasoning

Reasoning with functions by substitution

- can always replace (instance of) left hand side with right hand side, or vice versa
- no state, memory, etc. to screw things up
- relies on convention about mutually exclusive patterns

Example:

length (1:2:xs) = length (2:xs) + 1 = (length xs + 1) + 1 = length xs + 2

Derive properties by doing algebra

Haskell reasoning???

But can't you write inconsistent equations?

- e.g. f [] = 1 + f []

Solution:

- every Haskell type has special undefined element ⊥, read "bottom"
- have: f [] = ⊥
- therefore: ⊥ = 1 + ⊥

When running program, ⊥ means "infinite loop"

Laziness: suspending computations

What if:

g [] = 1:(g [])

Cons (:) and plus (+) work differently:

- 1+f [] evals 1 and (f []) then adds
- 1:(g []) created without eval’ing 1 and (g [])

More generally:

- (:) makes data structure, puts suspended computations in slots of structure

Classifying Haskell lists

What happens when you look for [] at end of list?

- *finite*: terminate
- *infinite*: get more and more conses forever
- *partial*: get ⊥ after seeing finitely many conses

Examples:

- *finite*, e.g. (1:(2:(3:[]))) = [1,2,3]
- *infinite*, e.g. g [] where g [] = 1 : g []
- *partial*, e.g. (1:(2:(3:⊥)))

Mutually exclusive and exhaustive

- any nonterminating expression = ⊥

Refinements of data

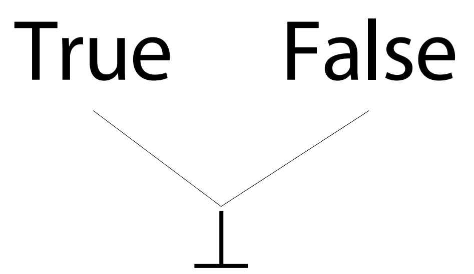
Every Haskell type has a *refinement order*:

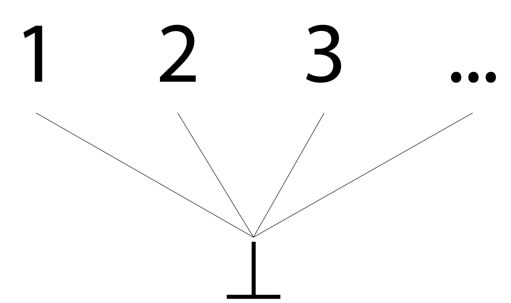
- read $x \leq y$ as “y at least as defined as x”

Pictorially ...

Integer and boolean refinements

All elements but ⊥ incomparable

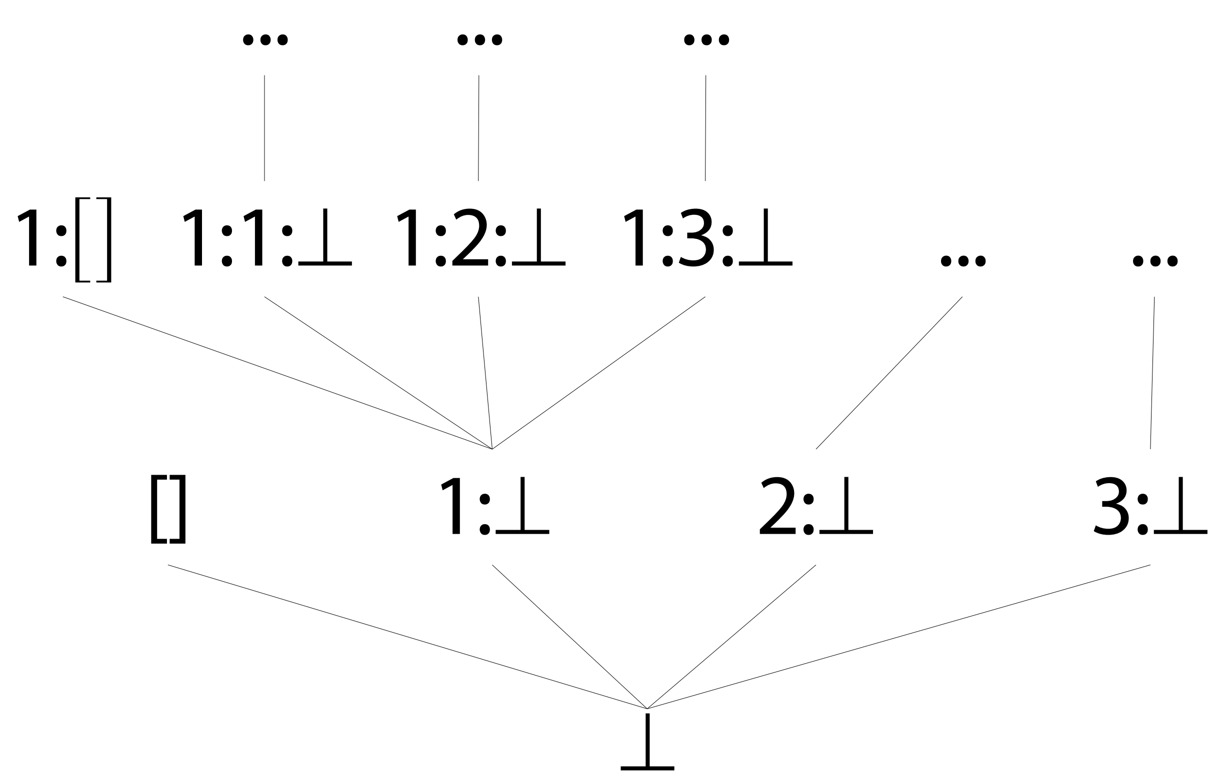




List refinements

[] and (:) incomparable; (:) monotone in both args

This talk: restrict lists to 1 2 3; all members defined for lists we consider



Talk outline

- Introduction
- Overview of Cosmological Theorem
- About Haskell and laziness
- Applying my method
- Conclusions

Using monotonicity

Fact: every definable Haskell function is monotone

- xs ≤ ys implies f xs ≤ f ys (in refinement order)
- i.e., as arg gets more defined, result gets more defined

Suppose:

- Given f such that f (1:⊥) = True

Then:

- f (1:xs) = True for any xs, by monotonicity of f

Don't need to see code for f!

Abstractly interpreting look and say

Define Haskell function *say* that does look and say

e.g. say [2,1,1,1] = [1,2,3,1]

Want *say* to be as lazy as possible

e.g. say (2:1:1:1:3:⊥) = (1:2:3:1:⊥)

e.g. say (2:1:⊥) = (1:2:⊥)

e.g. say (2:⊥) = ⊥

definition from my paper is indeed as lazy as possible

Covering all lists

Simulate f on all lists using abstract interpretation

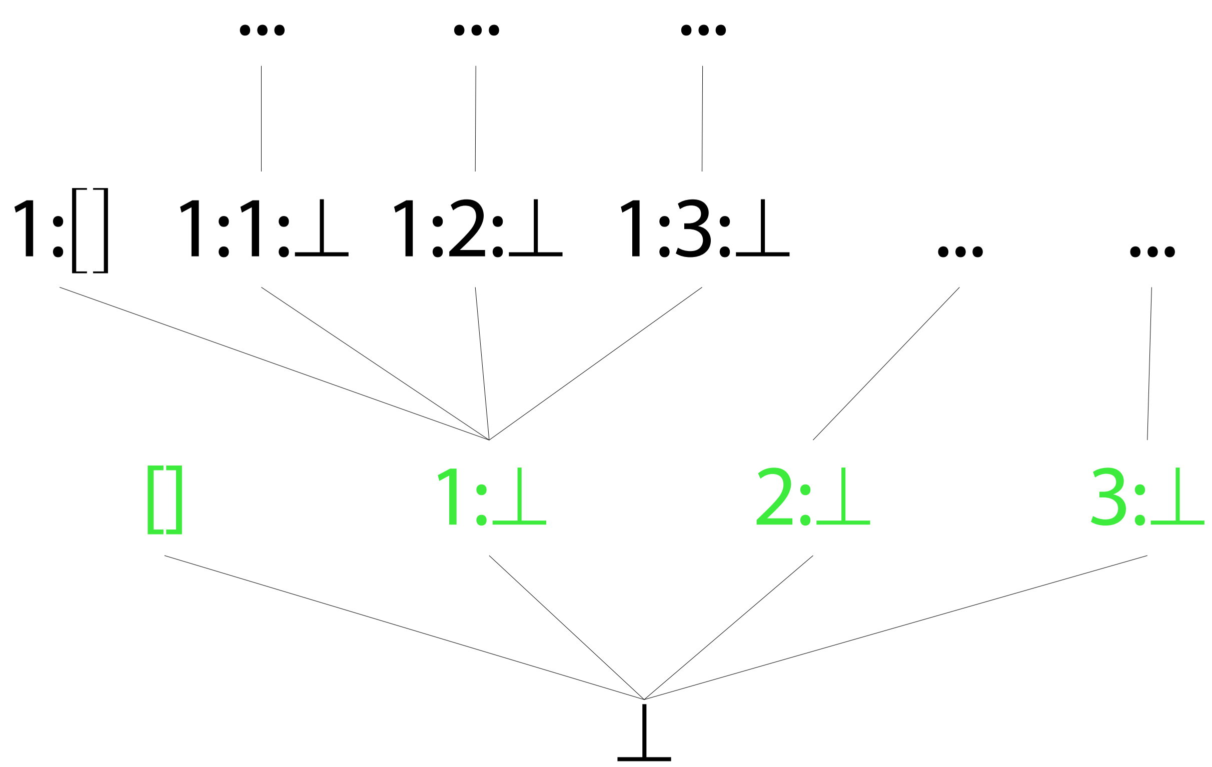
- pick (somehow) set C of partial (or finite) lists
- C must *cover* every list: $\forall xs \;\; \exists ys \in C$ such that $ys \leq xs$
- eval f on every list in C

Will explain how to pick C in a moment

Pictorially ...

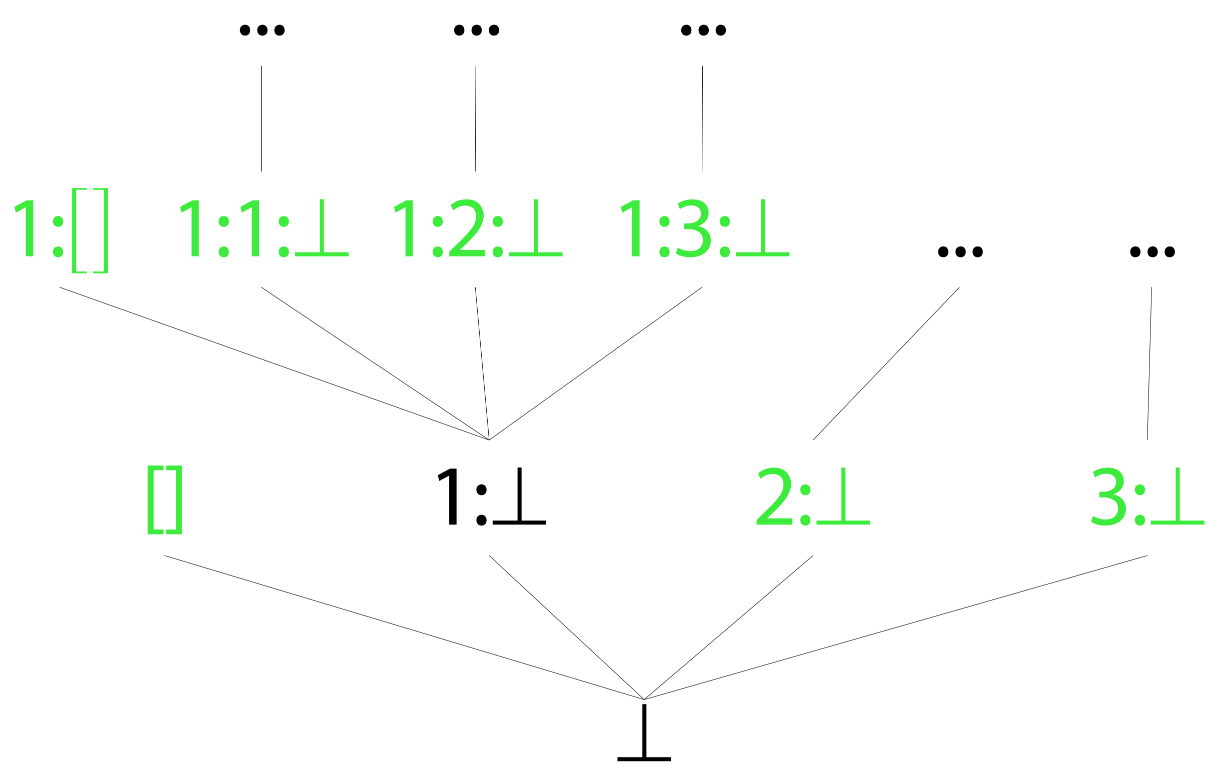
Covering all lists

A fringe in the tree of refinements



Covering all lists

A more refined cover



Applying the method

Recall: Splitting 2111...

2 . 111

12 . 31

1112 . 1311

3112 . 111321

132112 . 31131211

1113122112 . 132113111221

... etc.

2 always appears on left; want to show evolution of 111 always starts with 1 or 3

Finding the limit cycles

Pick covering set C (will say how later)

Execute the following Haskell code:

nub (map (take 20 · say^{30}) C)

nub removes duplicates from list

Constants 20, 30 chosen by trial and error

We get ...

Finding the limit cycles

? nub (map (take 20 · say^{30}) C)

[],
[31131122211311123113],
[13211321322113311213],
[13111213122112132113],
[31232112311321322112],
[11131221131211132221],
[22],
[22132113213221133112],
[22131112131221121321],
[22311311222113111231],
[22312321123113213221],
[11133112111311222112],
[22111312211312111322],
[22111331121113112221]

Finding the limit cycles

? nub (map (take 20 · say^{30}) C)

[],
[31131122211311123113],
[13211321322113311213],
[13111213122112132113],
[31232112311321322112],
[11131221131211132221],
[22],
[22132113213221133112],
[22131112131221121321],
[22311311222113111231],
[22312321123113213221],
[11133112111311222112],
[22111312211312111322],
[22111331121113112221]

Finding the limit cycles

? nub (map (take 20 · say^{30}) C) [rearranged]

[],
[11131221131211132221],
[31131122211311123113],
[13211321322113311213],
[11133112111311222112],
[31232112311321322112],
[13111213122112132113],
[22],
[22111312211312111322],
[22311311222113111231],
[22132113213221133112],
[22111331121113112221],
[22312321123113213221],
[22131112131221121321]

Finding the limit cycles

? nub (map (take 20 · say^{30}) C) [rearranged]

[],
[11131221131211132221],
[31131122211311123113],
[13211321322113311213],
[11133112111311222112],
[31232112311321322112],
[13111213122112132113],

[22],
[22111312211312111322],
[22311311222113111231],
[22132113213221133112],
[22111331121113112221],
[22312321123113213221],
[22131112131221121321]

By 30^{th} step we've reached a limit cycle!

Finding the limit cycles

? nub (map (take 20 · say^{30})) C [rearranged]

[],
[11131221131211132221],
[3113112221131112311 3],
[13211321322113311213],
[11133112111311222112],
[31232112311321322112],
[13111213122112132113],

[22],
[22111312211312111322],
[22311311222113111231],
[22132113213221133112],
[22111331121113112221],
[22312321123113213221],
[22131112131221121321]

By 30^{th} step we've reached a limit cycle!

By 32^{nd} step we've seen every starting number!

Decision procedure for splitting

By 30th step we've reached a limit cycle!

By 32nd step we've seen every starting number!

Define algorithm for starting numbers:

$\text{starts } xs = [\ \text{head } (\text{say}^n\ xs) \mid n \leftarrow [0..32]\ ]$

Define decision procedure for splitting:

$\text{splits } xs\ ys = \text{null } xs \lor \text{null } ys \lor \neg (\text{last } xs \in \text{starts } ys)$

Needed to pick C. How?

Picking a covering set

Use *oracle predicate* p to decide how far to refine

Call (cover p):

```
cover p = if p [] then [⊥] else
              [] :
              [ 1:xs | xs ← cover (λys. p (1:ys)) ] ++
              [ 2:xs | xs ← cover (λys. p (2:ys)) ] ++
              [ 3:xs | xs ← cover (λys. p (3:ys)) ]
```

Example:

cover ((== 2) · length) = { [], [1], 1:1:⊥, 1:2:⊥, 1:3:⊥, [2], ... }

Claim: if (cover p) terminates, result is covering set

– Don't have to look at p!

Putting it together

Determine appropriate oracle by experiment:

$p = ((\geq 12) \cdot \text{length} \cdot \text{say})$

Generate covering set using oracle:

$C = \text{cover } p$

Find limit cycles using covering set:

? nub (map (take 20 · say^{30}) C)

[[], [3113112221131112311 3], ...

Conclude that decision procedure is correct:

$\text{starts } xs = [\ \text{head } (\text{say}^n\ xs) \mid n \leftarrow [0..32]\ ]$

$\text{splits } xs\ ys = \text{null } xs \vee \text{null } ys \vee \neg\,(\text{last } xs \in \text{starts } ys)$

About the code

- Two more applications of the method
  - proving that a lazier version of *splits* is correct
  - finding all decay products of arbitrary strings using *splits*
- Literate Haskell program
  - 1311 lines (181 code + 1130 latex)
  - 98 LOC verified; 83 LOC in oracles, needn't be verified
- Compare:
  - Zeilberger (Maple): 2234 LOC (incl. self-documentation)
  - Litherland (C): 1650 LOC (less than half comments)

Talk outline

- Introduction
- Overview of Cosmological Theorem
- About Haskell and laziness
- Applying my method
- Conclusions

Proofs my method targets

In general, must:

- enumerate cases
- verify a property for each case

Examples:

- Four color theorem (Appel et al., 1976)
- Kepler's conjecture (Hales, 1998)

My method:

- *oracle strategy* simplifies showing sufficiency of enumeration
- *abstract interpretation via laziness* simplifies verifying property of each case

My contributions

New proof presentation strategy

- i.e. *oracle strategy* and *abstract interpretation via laziness* from previous slide
- may apply to similar proofs in other domains

Verify Conway's result

- Simple code: presented and justified in its entirety in my technical report
- Code written in a language with a simple semantics

Simplify prior proofs of Conway's result

- via my *marked sequences* (see technical report)

Questions?