The Monad.Reader Special Issue: Poetry and Fiction

by Anonymous
and Heinrich Apfelmus ⟨apfelmus@quantentunnel.de⟩
and Joachim Breitner ⟨mail@joachim-breitner.de⟩
and Neil Brown ⟨neil@twistedsquare.com⟩
and Todd Coram ⟨todd@maplefish.com⟩
and Claude Heiland-Allen ⟨claude@goto10.org⟩
and Tom Murphy ⟨amindfv@gmail.com⟩
and John D. Ramsdell ⟨ramsdell0@gmail.com⟩
and Fritz Ruehr ⟨fruehr@willamette.edu⟩
and Wouter Swierstra ⟨w.swierstra@cs.ru.nl⟩

March 15, 2011

Brent Yorgey, editor.

Contents

Editorial

by Brent Yorgey ⟨byorgey@cis.upenn.edu⟩

To be honest, I wasn't quite sure what to expect when I invited submissions to a special poetry and fiction issue of The Monad.Reader. However, I need not have worried. The final collection you now hold in your hands[1] includes several Haskell-inspired song rewrites by John Ramsdell, Joachim Breitner, and an anonymous submission from a disaffected user of GHC's rewriting features; a reflection by Wouter Swierstra on partial functions; some lovely haiku by Todd Coram and Tom Murphy; a poem written not about, but **by** a Haskell program; and a thought-provoking story about various aspects of films by Neil Brown.

I'm also pleased to be able to republish, in more fitting style, two gems which had previously only been published in wiki form: Fritz Ruehr's Seussian description of parser monads, and Heinrich Apfelmus's zipper tutorial à la classical romantic tragedy.

[1] Did you know that The Monad.Reader is specially designed to be enjoyed as a physically manifested codex? It's true. Here is a simple 6-step guide to obtaining maximum enjoyment from this and all future issues of The Monad.Reader. 1. Print double-sided. 2. Punch holes in the margin using a three-hole punch. 3. Place in a three-ring binder. 4. Procure your favorite beverage (hot or alcoholic beverage recommended). 5. Sit in your favorite comfortable location for sitting. 6. Read. Take occasional breaks from reading to sip the beverage procured in step 4.

The Poetry of Errors

I think there's a case that I missed,
For GHC seems to insist,
    That when I run main,
    it is all in vain:
`*** Exception: Prelude.head: empty list`

Wouter Swierstra

Evil Ways

You've got to change your evil ways... Rama
Before I stop respecting you.
You've got to change... Rama
And every word that I say, it's true.
You use strange syntax and typing
And offset rules
You don't mutate locations
You use strange do's
This can't go on...
Lord knows you got to change.

John D. Ramsdell

My son's nickname is Rama, so let me adopt it. I am a functional programmer, even when I use languages such as C. Scheme facilitated my development into a functional programmer, however, I appreciate the benefits of pure function programming at times. Yet when I use Haskell, I hear reminders of my Scheme past cast in the music of Santana. The words I hear are set to "Evil Ways".

Wrote it in Haskell

(sung to the tune of "Cat's in the Cradle")

Our new semester, it has just begun,
And it did seem to me like it would bring some fun.
Then we met the prof and the way he spoke
That's not German 'cause his German is broke
Seems to me that if I want to get along
I get used to his tongue. I learn to hear his tongue.

So I wrote it in Haskell in the dark pool room
It is night and I type and I toil here since noon.
I got no variables and it freaks me out
But I can do without
Somehow I can do without.

Once I did that job, had to code that thing,
And they left me the choice what I would code it in.
My buffers overflow'd when I was using C
Slowest language there is: That's PHP
Java's bloated VM always got me stuck
I'm too dumb for brainfuck, I wish I'd use brainfuck

Joachim Breitner

In the second semester of my university computer science education in Germany, we had a professor from France with a heavy accent. That course introduced, among other things, Haskell. Unfortunately, its treatment scared off more people than it attracted, and I did not like Haskell until a year later. At the time, I was experimenting with Weird Al-inspired song parodying, and this is one of the early results. More can be found on my homepage, `http://www.joachim-breitner.de/content#parodie`.

Cinematic

by Neil Brown ⟨neil@twistedsquare.com⟩

The professor cleared his throat. "Today, we will discuss programming language design, including the different features of various programming languages." Ryan, who was near the back of the lecture theatre, yawned and slumped down in his seat. The lecture crawled through an hour of discussion of object-oriented versus functional programming, static versus dynamic typing and all sorts of other details. After the lecture was over, John, Alex and Ryan left the lecture theatre together, emerging into the crisp air of winter in the American Northeast.

The three of them were a mixed group. Alex was tall and thin, European but without a noticeable accent, and somehow always stuck out from everyone else, for better or for worse. John was almost the opposite, short and quiet around strangers (but not around friends), often anonymous in larger groups. The two of them were friends, but seemed destined to disagree on almost everything, which always entertained Ryan. Ryan was originally from a forgettable little town somewhere in the Midwest, and he seemed to drift through life with a half-smile on his face.

"Okay, what lecture's next?" asked Ryan.

"Film, in half an hour – but on the other side of campus," replied Alex, pointing towards the way that they should go.

"Hopefully a bit less dry than computing then," said John, as he quickened his pace to try to catch up with the others. "Now, film is something you can have a good, proper discussion about."

"Ah, but isn't it too subjective?" Ryan said, a wilful instigator. "Isn't the problem that everyone just ends up praising their favourite films, and dismissing the films they don't like? I mean, the programming language stuff shouldn't be like that at all, it should be more objective, right?"

"Well, you just have to try to consider –" John paused, and adopted the heavy accent of their programming language professor, "the different features of various films."

"But can you decompose it like that?" asked Alex, instantly arguing with John. Ryan had observed that it was never a deliberate attempt to disagree with each

other, but simply that the two of them held opposing views on most things. Alex continued: "Surely each aspect is better or worse depending on how it fits with the other parts. I mean, you can't really judge a film's score separately from the film, what makes the difference is how well the parts fit together."

"But I think you can still decide on some aspects. You can admire the special effects on a film with poor acting and script, or you can admire the script for a low-budget indie film," interjected Ryan.

"And the nice thing is, that the best parts can be taken and tweaked and included in other films – or the more 'out-there' films can get toned down and remade for a more commercial and mainstream version," John pointed out. Alex made a face of disgust.

They reached a pedestrian crossing, and they waited there with small groups of other students from the programming languages lecture. Language names were being muttered as students discussed the lecture, with mentions of Smalltalk and Java, among many others, filling the air.

The crossing light lit up and they crossed the road, continuing on towards their film lecture.

"What did you think of that interview in the film lecture last week, with that script-writer discussing his inspiration for the film?" John asked.

Alex modulated his voice, giving the impression of a posh artist speaking in lofty tones: "Oh, I read extensively, I drew inspiration from women's liberation, the Dadaist movement and its postmodern reaction, and the struggle for democratic representation in post-colonial regimes when I was constructing the script for my masterpiece: Yet Another Rom-Com 6."

"That's the problem with all these artists – to borrow some terminology – their work is not injective; it's lossy compression," Ryan said, unusually animated. "If you ever read interviews with them, they often talk like that about all these hundreds of different background materials that they read, how they were influenced by all sorts of grand thinkers and schools of art and theories and whatever, but ultimately what they produced was a hundred-page script and that's it. Usually, you can detect hardly any of what the artist said went into the script. It may help them when writing the script, and the background and sources may appeal to other scriptwriters and film critics, but what does some random guy who watches the film care about that?"

"Maybe that's good, though," argued John. "Maybe you shouldn't need to know what went into it. If it helps somehow make the film better, great. But if I can just watch the two-hour film and not worry about the hundred hours of research that went into it, I count that as a plus. They've distilled what they wanted into a form I can comprehend, and everyone walks away happy."

"But it can be important, I think," responded Alex, disagreeing with John again. "To understand how several films were inspired by the same concept, to see that a

film works because of the precepts underlying it."

"Let's take The Crucible," suggested Ryan. "The film, I mean – I haven't seen the play. Ostensibly about the Salem witch-hunts, but really inspired by McCarthy and all that. But does it actually matter what inspired it? The film's message is still there – laid out a bit too plainly – regardless of whether the writer based it on McCarthy, Salem, or some gossiping old biddy in his local church back in the '30s. If the background helps the writer, that's all well and good, but often it just doesn't matter. I've got the film, I can understand it perfectly well, who needs to know what underpinned it?"

The three fell silent for a moment, before John asked: "So in that case, does it need all this thought to go into it? Maybe you can make really popular films without any deep thought behind them, and they would do just as well?"

"They might do well in audience numbers – but unfortunately, popularity is a poor measure of quality," Alex pointed out. "Take Star Wars, which permanently rides high in IMDB's film ratings – which are more a measure of popularity than quality. Nostalgia aside, it doesn't compare to the best films ever made. But it is at least simple: back in the 70s, Lucas took this hero's journey thing, this nice sequential model, then added bells and whistles in the form of special effects, and it – along with its immediate sequels and remastered editions – is probably still the most popular film ever made. In a way you have to admire it, and it's worth trying to understand its success, but that doesn't make it a good film, especially by modern standards."

"But don't you think that its popularity necessitates that it is, at least in some respects, a good film?" John argued. "It's accessible; by the time he remastered it, it's nicely polished – and it's something people can get to grips with. I think as a creator it's worth not losing sight of that, of not getting so lost in the art that you fail to bring everyone else with you. Maybe popularity *is* important." As John walked along, his foot accidentally kicked a small stone, sending it flying across the parking lot, dinging quietly into the bodywork of a parked car.

"Perhaps," Alex replied, rolling his eyes at John's clumsiness.

"Listen," John continued, determined to get his point across: "Take Inception –"

Alex sighed audibly.

"I know you're not the biggest fan, not a patch on some of Nolan's earlier films, *etcetera etcetera*," John said quickly, not wanting to retread earlier arguments. "But: it took a plot that, on paper, is tremendously complex, and by the time it was in the cinema it was made so easy that it seemed almost too simplistic. That's an amazing accomplishment, regardless of what you think of the other aspects of the film. You mustn't lose sight of your audience; a film without an audience is not accomplishing any purpose."

"So do you think that all films should bow to the masses? Would Pi have been better in colour with simple camera work?" Alex asked sarcastically.

"No, I think it's important for some films to be bold, to risk being unpopular, in order to collectively drive all of them forward. But I'm saying that, maybe, later on, it's better to distill some of that into a palatable film for everyone."

"Doesn't that dilute the strength of the original, though?" responded Alex.

"It's a good question," John admitted, and he had no good answer.

The conversation lapsed and they walked along in silence until John spoke again.

"So what did you think about the guy in the lecture last week who said that a film being subtitled meant it could never be considered a truly great film?" John asked, in an attempt to fill the silence and move the conversation on.

"I think he was just trying to annoy the lecturer," Ryan answered. "I mean, what a crazy point! For a start, it's all relative – if a French-speaker made the same claim, everyone would immediately decry him for being dismissive of non-French films. Preference in that sort of issue tends to just be a matter of comparing it to your native language."

"It does make a difference to how you view the film, though – being subtitled, I mean," John pointed out.

"It's the tiniest of differences, and one you should be able to get over in a few minutes," Alex said. "What is more important about films being in a different language is that they tend to arise from a different background and culture. That can make it harder to get into it. Watching a Chinese film can be strange, not because of the subtitles, but because of the totally different culture."

"But that's good, right?" suggested Ryan. "Experiencing different cultures and attitudes via the different films they produce. It leaves you more educated afterwards. I reckon it should be viewed as an ideal, to try to watch films from as many different cultures as possible. I think it helps your appreciation. There are good films from every culture."

"Do you think that some strands are better than others, though?" John asked.

"That sounds a bit xenophobic. . . " Alex said.

"No, hear me out," insisted John. "If you consider how popular films are – or, if you don't like popularity – how highly rated films are, it's fairly clear that US films are way ahead of any other type of films. Surely that must be a sign that they have an implicit advantage?"

Ryan answered quickly: "To throw your own point back at you, is that not just a language issue, that English is probably the most spoken language in the world, once you consider first and second languages?"

"But in that case, why isn't it British films or Canadian films that are so popular?" John pointed out. "Could it be that the styles of film they make, or their attitude to film-making is the reason?"

"Probably their attitude to finance – I mean, it's surely an issue of resources," Alex argued. "At some point in the past, the US films were the most popular, and that led to a greater investment in them, which perpetuated their popularity. By

now, it's the twin reasons of greater investment, and also a sort of indoctrination – US films are so ubiquitous that everyone is used to the culture behind them, so they have an amazing momentum." Then he added: "But maybe it's not really deserved."

"What if French cinema had really taken off globally, and people had gotten used to subtitles early on, how different would things be now?" suggested Ryan.

"But it's possible that they could be heralding US cinema anyway," John responded. "No-one can tell of course, but you can't necessarily hide behind history and momentum as the reason for people's preferences. It could really be intrinsic."

"That just sounds a bit culturally imperialistic to me," said Alex, and Ryan nodded.

John shrugged. "Something to think about, I'm just saying that you can't rule it out even if you'd prefer it not to be true. Like it or not, the burden now falls on films from other cultures to show that they can be successful."

As the three of them walked around the corner of the library, they could finally see their destination. "It's a shame on this film module that we don't get to actually make a film," Ryan mused as they walked on.

"The thing is, anyone can make a film nowadays," John pointed out. "The technology is easily available for anyone, so they can have a go and scratch their particular itch by making a short film."

"Those things tend to be very specific though, a one-shot short concept piece that doesn't have as much depth and expression as full-length films," Alex responded.

"If someone is satisfied with it then it's not for us to tell them to do otherwise," Ryan said calmly.

"Something I find interesting is not so much the new films, but the remixes that people make, or things like machinima," John said. "Rather than making their own footage and turning it into a film, they take existing material and re-purpose it to their own ends – it's the modern remixing mentality."

"Some of those can be pretty ghastly though. We don't want a world of Ed Woods, twisting film footage into some nonsensical mess," replied Alex.

"OK, so you can produce some pretty nasty things that way. But how much easier could it be to get started; you want to create something for you and your friends, so why not piggy back on top of someone else's efforts to get you mostly there and then tweak as you need?" Ryan said.

"True," said Alex, as they arrived at the lecture theatre. They filed in after all the other students. The professor arrived and began his lecture: "Okay everyone, now let's start talking about films."

Ryan yawned, and slumped down in his seat.

My RULES my RULES my RULES do not fire
My RULES my RULES my RULES do not fire
My RULES my RULES my RULES do not fire
We don't need no PRAGMA let the optimizer burn
BURN OPTIMIZER BURN

Anonymous

Parser Monads in the Style of Dr. Seuss

A Parser for Things
is a function from Strings
to Lists of Pairs
of Things and Strings!

Fritz Ruehr

Theseus and the Zipper

by Heinrich Apfelmus

Theseus, the well-known mythic hero, has to write a computer game featuring a labyrinth. Will he succeed in writing it in Haskell and thereby save his company from insolvency? Will his ex-girlfriend Ariadne provide the dearly needed help?

The Labyrinth

"Theseus, we have a problem," said Homer, chief marketing officer of Ancient Geeks Inc. Theseus put the Minotaur action figure$^{\text{TM}}$ back onto the shelf and nodded. "Our company is on the brink of insolvency. If we don't come up with a new product that is hip and modern, we are done for. Today's children are simply no longer interested in the ancient myth stuff, they prefer modern heroes like Spiderman or Sponge Bob."

Heroes. Theseus knew well how little he had been a hero in the labyrinth back then on Crete [1], but at least he was a human of flesh and blood, not one of these "modern heroes", these utterly unrealistic radioactive clone mutant sponges. Just what made them so successful, so appealing to the young audience? Anyway, if the pending sales problems could not be resolved, the shareholders would certainly arrange a passage over the Styx for Ancient Geeks Inc.

"Heureka! Theseus, I have an idea: we make a computer game! This is hip and modern! How about we implement your story with the Minotaur? What do you say?"

Homer was right. There had been several books, epic (and chart breaking) songs, a mandatory movie trilogy and uncountable Theseus & the Minotaur$^{\text{TM}}$ gimmicks, but a computer game was dearly missing. How could they have been so blind?

"Alright then, Theseus, your task is to program the game."

A true hero, Theseus chose Haskell as the programming language to implement the company's last hope in. Of course, exploring the labyrinth of the Minotaur

was to become one of the game's highlights. He pondered: "We have a two-dimensional labyrinth whose corridors can point in many directions. Of course, we can abstract from the detailed lengths and angles: for the purpose of finding the way out, we only need to know how the path forks. To keep things easy, we model the labyrinth as a tree. This way, the two branches of a fork cannot join again when walking deeper and the player cannot go round in circles. But I think there is enough opportunity to get lost; and this way, if the player is patient enough, he can explore the entire labyrinth with the left-hand rule."

```
data Node a = DeadEnd a
            | Passage a (Node a)
            | Fork    a (Node a) (Node a)
```

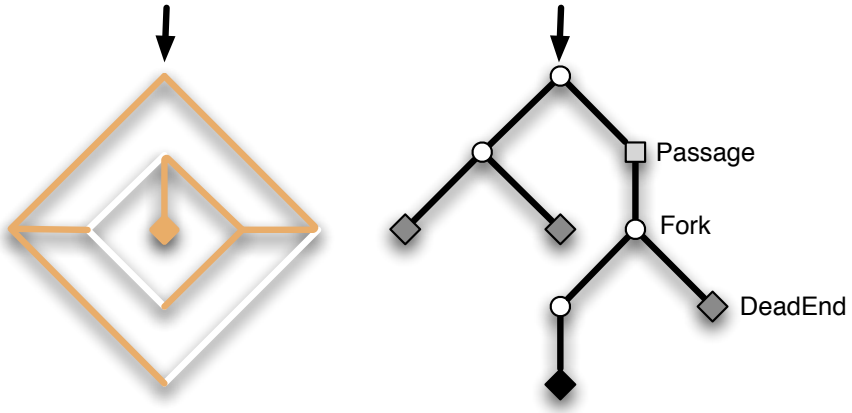

Figure 1: An example labyrinth and its representation as tree.

Theseus made the nodes of the labyrinth carry an extra parameter of type `a`. Later on, it might hold information relevant to the game, like the coordinates of the location that a node designates, the ambience around it, a list of game items that lie on the floor, or a list of monsters wandering in that section of the labyrinth. He started by writing two helper functions,

```
get :: Node a -> a
put :: a -> Node a -> Node a
```

to retrieve and change the value of type `a` stored in the first argument of every `Node` constructor.

Exercise 1. Implement `get` and `put`. One case for `get` is

```
get (Passage x _) = x
```

Exercise 2. As a concrete example, write down the labyrinth shown in the picture as a value of type `Node (Int,Int)`. The extra parameter `(Int,Int)` holds the Cartesian coordinates of a node.

"Hmm, how to represent the player's current position in the labyrinth? The player can explore deeper by choosing left or right branches, like this. . .

```
turnRight :: Node a -> Maybe (Node a)
turnRight (Fork _ l r) = Just r
turnRight _            = Nothing
```

But the player would not be able to go back if we were to throw away the current top of the labyrinth and replace it by its right or left branch. We have to keep the labyrinth intact. But then, how do we keep track of where the player is located?" Theseus pondered. "Ah, we can apply **Ariadne's trick with the thread**! We simply represent the player's position by the list of branches that his thread has taken. The labyrinth itself stays as it is."

```
data Branch = KeepStraightOn
            | TurnLeft
            | TurnRight
type Thread = [Branch]
```

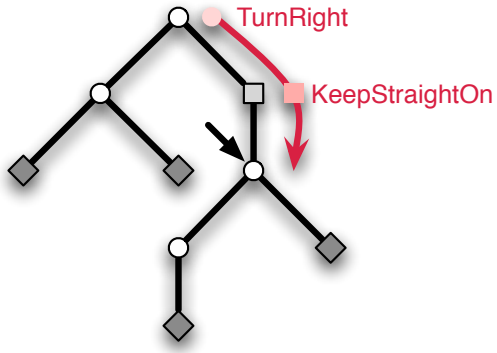

Figure 2: Representation of the player's position by Ariadne's thread.

"For example, a thread `[TurnRight,KeepStraightOn]` means that the player took the right branch at the entrance and then went straight down a `Passage` to reach their current position. With the thread, the player can now explore the labyrinth by extending or shortening it. For instance, the function `turnRight` extends the thread by appending `TurnRight` to it."

```
turnRight :: Thread -> Thread
turnRight t = t ++ [TurnRight]
```

"To access the extra data, like ambiance, items, and such, we simply follow the thread into the labyrinth."

```
retrieve :: Thread -> Node a -> a
retrieve []                    n             = get n
retrieve (KeepStraightOn:bs) (Passage _ n) = retrieve bs n
retrieve (TurnLeft      :bs) (Fork _ l r)  = retrieve bs l
retrieve (TurnRight     :bs) (Fork _ l r)  = retrieve bs r
```

Exercise 3. Write a function `update` that applies a function of type `a -> a` to the extra data at the player's position.

Theseus's satisfaction over this solution did not last long, however. "While this method works, it is very slow for large labyrinths. After all, each and every time we want to access the data at the player's position in the labyrinth, we have to follow the thread deep into the labyrinth, which takes time proportional to the length of the thread. Shouldn't there be a much faster way to do it?"

Ariadne's Zipper

While Theseus was a skillful warrior, he did not train much in the art of programming and could not find a satisfying solution. After intense but fruitless cogitation, he decided to call his former love Ariadne to ask her for advice. After all, it was she who had come up with the idea of the thread in the first place.

"Ariadne Consulting. What can I do for you?"

Our hero immediately recognized the voice.

"Hello Ariadne, it's me, Theseus."

An uneasy silence suspended the conversation. Theseus remembered well that he had abandoned her on the island of Naxos and knew that she would not appreciate his call. But Ancient Geeks Inc. was on the road to Hades and he had no choice.

"Uhm, darling... how are you?"

Ariadne retorted coolly, "Mr. Theseus, the times of *darling* are long over. What do you want?"

"Well, I uhm... I need some help with a programming problem. I'm writing a new Theseus & the Minotaur™ computer game."

She jeered, "Yet another commercial artifact with your famous name on it? And you want me, of all people, to help you?"

"Ariadne, please, I beg you, Ancient Geeks Inc. is on the brink of insolvency. The game is our last hope!"

After a short pause, she came to a decision.

"Fine, I will help you. But only if you transfer a substantial part of Ancient Geeks Inc. to me. Let's say thirty percent."

Theseus turned pale. But what could he do? The situation was desperate enough, so he agreed but only after negotiating Ariadne's share to a tenth.

After Theseus told Ariadne of the labyrinth representation he had in mind, she could immediately give advice.

"You need a **zipper**."

"Huh? What does the problem have to do with my fly?"

"Nothing, it's a data structure first published by Gérard Huet [2]."

"Ah."

"More precisely, it's a purely functional way to augment tree-like data structures like lists or binary trees with a single **focus** or **finger** that points to a subtree inside the data structure. This focus allows constant time updates and lookups at the location it points to.[1] In our case, we want a focus on the player's position."

"I already know that I want fast updates, but how do I code it?"

"Don't get impatient. You cannot solve problems by coding, you can only solve them by thinking. If you want constant time updates in a purely functional structure, you should aim at keeping your data at the topmost node or close to it.[2] After all, that's the only place you can reach if you are only allowed to walk a constant numbers of steps down into the data structure. So, the focus has to be at the top."

She continued, "Currently, the topmost node in your labyrinth is always the entrance, which is bad. Your previous idea of replacing the labyrinth by its left or right branch was much better because it ensures that the player's position is at the topmost node."

Theseus interrupted: "But then, the problem is how to go back, because all the branches that the player did not choose will get lost."

"Well, you can use my thread in order not to lose the branches."

Ariadne savored Theseus's puzzlement, but quickly continued before he could complain that he had already used her thread.

"The key is to *glue* the lost branches to the thread so that they actually don't get lost at all. The intention is that the thread and the current sub-labyrinth complement one another to make the whole labyrinth. By 'current' sub-labyrinth,

[1] Note the notion of **zipper** as coined by Gérard Huet also allows to replace whole subtrees even if there is no extra data associated with them. In the case of our labyrinth, this is irrelevant.

[2] However, sometimes, one can use **amortization** to achieve constant time even if many more nodes than just the top node are affected. An example is incrementing a number in binary representation. While incrementing say `111..11` must touch all digits to yield `1000..00`, the increment function nevertheless runs in constant amortized time (but not in constant worst case time).

I mean the one that the player stands on top of. The zipper simply consists of the thread and the current sub-labyrinth."

```
type Zipper a = (Thread a, Node a)
```

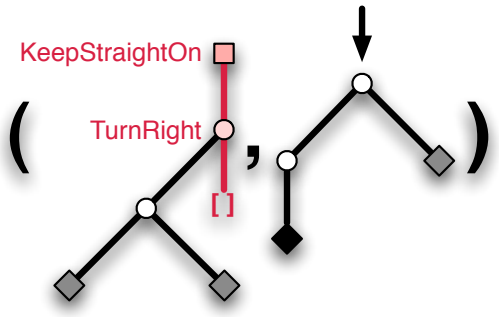

Figure 3: The zipper is a pair of Ariadne's thread and the current sub-labyrinth where the player stands. The main thread is colored red and has sub-labyrinths attached to it, such that the whole labyrinth can be reconstructed from the pair.

Theseus didn't say anything.

"You can also view the thread as a **context** in which the current sub-labyrinth resides. Now, let's find out how to define `Thread a`. By the way, `Thread` has to take the extra parameter `a` because it now stores sub-labyrinths. The thread is still a simple list of branches, but the branches are different from before."

```
data Branch a  = KeepStraightOn a
               | TurnLeft  a (Node a)
               | TurnRight a (Node a)
type Thread a  = [Branch a]
```

"Most importantly, `TurnLeft` and `TurnRight` have a sub-labyrinth glued to them. When the player chooses, say, to turn right, we extend the thread with a `TurnRight` and now attach the untaken left branch to it, so that it doesn't get lost."

Theseus interrupted, "Wait, how would I implement this behavior as a function `turnRight`? And what about the first argument of type `a` for `TurnRight`? Ah, I see. We not only need to glue the branch that would get lost, but also the extra data of the `Fork` because it would otherwise get lost as well. So, we can generate a new branch by a preliminary

```
branchRight (Fork x l r) = TurnRight x l
```

Now, we have to somehow extend the existing thread with it."

"Indeed. The second point about the thread is that it is stored **backwards**. To extend it, you put a new branch in front of the list. To go back, you delete the topmost element."

"Aha, this makes extending and going back take only constant time, not time proportional to the length length as in my previous version. So the final version of `turnRight` is

```
turnRight :: Zipper a -> Maybe (Zipper a)
turnRight (t, Fork x l r) = Just (TurnRight x l : t, r)
turnRight _               = Nothing
```

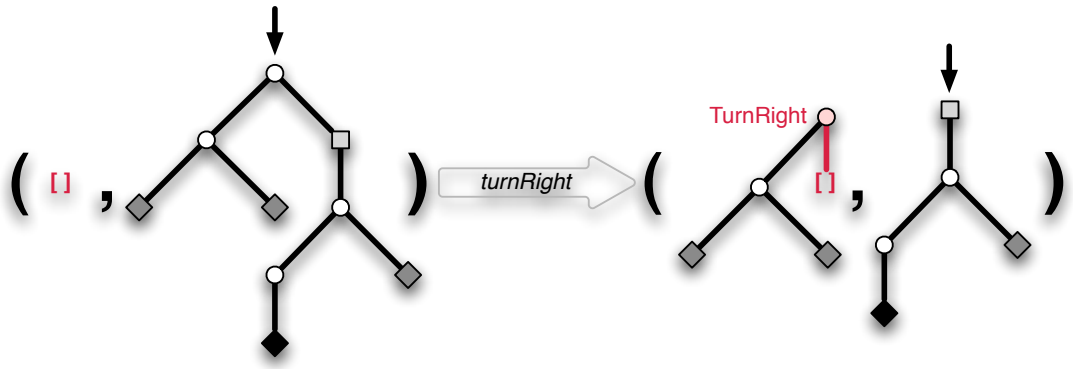

Figure 4: Taking the right branch from the entrance. Of course, the thread is initially empty. Note that the thread runs backwards, i.e. the topmost segment is the most recent.

"That was not too difficult. So let's continue with `keepStraightOn` for going down a passage. This is even easier than choosing a branch as we only need to keep the extra data:

```
keepStraightOn :: Zipper a -> Maybe (Zipper a)
keepStraightOn (t, Passage x n) = Just (KeepStraightOn x : t, n)
keepStraightOn _                = Nothing
```

Exercise 4. Write the function `turnLeft`.

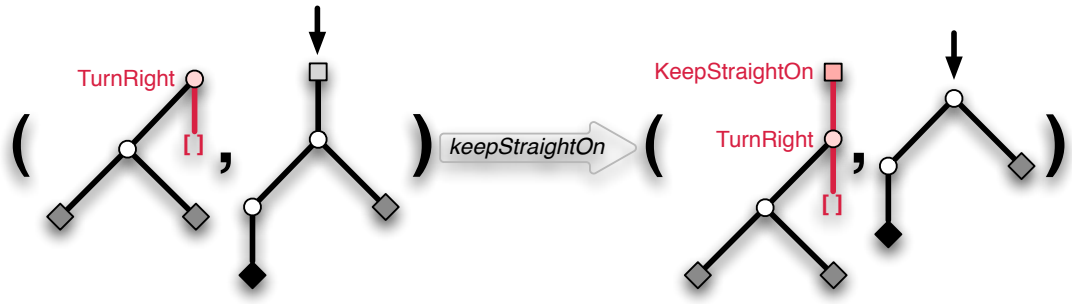

Figure 5: Now going down a passage.

Pleased, Theseus continued, "But the interesting part is to go back, of course. Let's see...

```
back :: Zipper a -> Maybe (Zipper a)
back ([]                    , _) = Nothing
back (KeepStraightOn x : t , n) = Just (t, Passage x n)
back (TurnLeft  x r    : t , l) = Just (t, Fork x l r)
back (TurnRight x l    : t , r) = Just (t, Fork x l r)
```

If the thread is empty, we're already at the entrance of the labyrinth and cannot go back. In all other cases, we have to wind up the thread. And thanks to the attachments to the thread, we can actually reconstruct the sub-labyrinth we came from!"

Ariadne remarked, "By the way, if you are not sure whether what you wrote is correct, you can do a nice test. Namely, check that each bound variable like `x`, `l` and `r` from the left hand side appears exactly once on the right hands side as well. After all, when walking up and down a zipper, we only redistribute data between the thread and the current sub-labyrinth, we never throw it away or create it anew."

Exercise 5. Now that we can navigate the zipper, code the functions `get`, `put` and `update` that operate on the extra data at the player's position.

Exercise 6. Zippers are by no means limited to the concrete example `Node a`; they can be constructed for all tree-like data types. Go on and construct a zipper for binary trees

```
data Tree a = Leaf a | Bin (Tree a) (Tree a)
```

Start by thinking about the possible branches `Branch a` that a thread can take. What do you have to glue to the thread when exploring the tree?

Exercise 7. Simple lists have a zipper as well.

```
data List a = Empty | Cons a (List a)
```

What does it look like?

Exercise 8. Write a complete game based on Theseus's labyrinth.

Heureka! That was the solution Theseus sought. Ancient Geeks Inc. was saved, even if partially sold to Ariadne Consulting. But one question remained.

"Why is it called a zipper?"

"Well, I would have called it 'Ariadne's pearl necklace'. But most likely, it's called a zipper because the thread is in analogy to the open part of a zipper and the sub-labyrinth is like the closed part. Moving around in the data structure is analogous to zipping or unzipping the zipper."

"Ariadne's pearl necklace," he echoed mockingly. "As if your thread was any help back then on Crete."

"As if the idea with the thread was yours," she replied.

"Bah, I need no thread." He defied the fact that he actually did need the thread to program the game.

Much to his surprise, she agreed. "Well, indeed, you don't need a thread. Another point of view is to literally grab the tree at the focus with your finger and lift it up in the air. The focus will be at the top and all other branches of the tree hang down. You only have to assign the resulting tree a suitable algebraic data type, most likely that of the zipper."

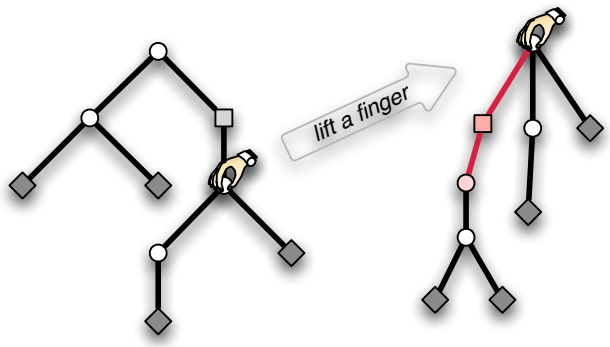

Figure 6: Grab the focus with your finger, lift it in the air and the hanging branches will form new tree with your finger at the top, ready to be structured by an algebraic data type.

"Ah." He didn't need Ariadne's thread but he needed Ariadne to tell him? That was too much.

"Thank you, Ariadne, good bye."
She did not hide her smirk as he could not see it anyway through the phone.

Exercise 9. Take a list, fix one element in the middle with your finger and lift the list into the air. What type can you give to the resulting tree?

Half a year later, Theseus stopped in front of a shopping window, defying the cold rain that tried to creep under his buttoned up anorak. Blinking letters announced

"Spider-Man: Lost in the Web"
— find your way through the labyrinth of threads —
The great computer game by Ancient Geeks Inc!

He cursed the day when he had called Ariadne and sold her a part of the company. Was it she who contrived the unfriendly takeover by WineOS Corp., led by Ariadne's husband Dionysus? Had that been her plan all along? Theseus watched the raindrops find their way down the glass window. After the production line was changed, nobody would sell Theseus & the Minotaur^{TM} merchandise anymore. He sighed. His time, the time of heroes, was over. Now came the super-heroes.

References

[1] Ian Stewart. The true story of how theseus found his way out of the labyrinth. **Scientific American**, page 137 (February 1991).

[2] Gérard Huet. The zipper. **Journal of Functional Programming**, pages 549–554 (September 1997). `http://www.st.cs.uni-sb.de/edu/seminare/2005/advanced-fp/docs/huet-zipper.pdf`.

Just (5, "Haiku")

A lone curried flower,
yielding only to the wind
with monadic kisses.

Guarding purity: chaste.
Discrete dalliances
permitted through monads.

A golden curry flavor,
strongly typed and fragrant.
It tastes of Haskell.

With strongly typed hands,
I recurse guardingly
in comprehensive repose.

Type your data.
Strictly bind them.
And they shall not go astray,
but be pure and strong.

Todd Coram

Red Lorry Yellow Lorry

Red lorry yellow lorry.
Lea dory deadly oldie.
Yield airy rarely deli.
Ye oldie deli deadly.
Dough reel redraw already.
Ell aerie really dory.
Owe really arrow deadly.
Lee eel already doughy.
D eddy really rarely.
Draw oldie erode deadly.
Heir oldie deli erode.
Awed eh already erode.
D eyrie really aerie.
Rare lorry lorry dory.
O yellow dread already.
Roll erode dodo ready.
Dread ready readied rarely.
Ode really ready yellow.
Rode rarely eyrie deadly.
All eyrie owed already.
Low eyrie dory lowly.
Lo eyrie ready eddy.
Rowed arrow deli rarely.
Doled really doughy oldie.
Yo aerie lowly deadly.
All eyrie arrow dodo.
Roared reedy yellow eyrie.
Dread yellow lorry airy.
Doled lowly dory aerie.
Led lead already eddy.
Oared dodo rarely yellow.
Real deadly eyrie eddy.
Doled readied eyrie dory.
Reel re already ready.
Roar eddy deli dairy.
Rode dairy dairy arrow.
Load role already lorry.
Reeled eyrie deli dairy.
Ore arrow eddy yellow.
Roared lowly doughy aerie.
Roared deli yellow dory.
Air eddy erode reedy.
O reedy dodo yellow.
Lo dared already reedy.
Droll lea already yellow.
Lea rarely yellow dodo.
Rode oldie aired already.
Real really d already.
Laud aerie deadly reedy.
Read oldie erode airy.
Oar eyrie rarely dodo.
Rowed dory ready aerie.
Oh yo reload already.
O dairy deli lorry.
Rode oldie readied readied.
Awed yellow lorry dairy.
O doughy yellow aerie.
Ere air already yellow.
All deli ye already.
Roe reedy reedy arrow.
Air aerie dairy rarely.
Row dory dodo yellow.
D airy arrow eyrie.
Red lorry yellow lorry.

A Haskell program (programmed by Claude Heiland-Allen)

Haiku

To recurse through lists,
Simply work on beginnings
Until it's the end

Tom Murphy