

"cs is just fancy string manipulation"
but categorical

Zach Goldthorpe

18 October 2019

Strings

What are strings?

1 `using string=char *;`

`practical` : a pointer to the head of an array of chars
(but also a dated definition)

`useless` : `char` is a set of letters, and “*” is the Kleene star
so

$$\text{string} = \text{char}^* = \bigcup_{n \geq 0} \text{char}^n$$

We're rolling with the `useless` definition.

Fix an alphabet Σ ($|\Sigma| > 1$), then our set of strings is Σ^* .

Turing Machines

What is a Turing machine?

Turing machines are defined by unrealistic and unhealthy beauty standards for computers.

So are our strings right now, to be fair...

For us, a Turing machine M will just be a function

$$M : \Sigma^* \rightarrow \Sigma^* \sqcup \{\text{"Segmentation fault (core dumped)"}\}$$

Turing Machines

What are Turing machines supposed to do?

Definitely not machine learning...

```
1 void foo(int x, int y, int n) {  
2     for (int k = 1; k <= n; ++k)  
3         printf("%s%s%.d",  
4             k%x?"":"Fizz",k%y?"":"Buzz", (k%x&&k%y)*k);  
5 }
```

They are supposed to solve problems, but for simplicity let's only look at **decision problems**.

Categories

What is a category?

Definition

A category \mathcal{C} consists of

- **objects** (e.g., X, Y, Z, \dots)
- **arrows** between objects (e.g., $f : X \rightarrow Y$)

so that

- arrows can be **composed** (e.g., $g \circ f : X \xrightarrow{f} Y \xrightarrow{g} Z$)
- there is a “**do nothing** arrow” $\text{id}_X : X \rightarrow X$ for every X

Categories

Example (sets)

We have a category **Set** where

- the objects of **Set** are **sets**
- the arrows between sets X, Y are the **functions** $f : X \rightarrow Y$

Example (vector spaces)

Similarly we have a category **Vect**_ℝ where

- the objects of **Vect**_ℝ are **real vector spaces**
- the arrows are the **linear transformations** $T : V \rightarrow W$

Category of Fancy String Manipulation

Let's define the category **CS** where

- the objects of **CS** are **sets of strings**
hence subsets $L \subseteq \Sigma^*$
- the morphisms $L \rightarrow S$ are Turing machines M so that

$$M(x) \in S \iff x \in L$$

(co)Recognition

"You're discriminating against code I write!"

$\overrightarrow{\text{CS}}$ "QA engineer isn't paid enough": $x \in L \implies M(x) \in S$ and
 $x \notin L \implies M(x) \notin S$ or Segmentation fault

$\overleftarrow{\text{CS}}$ "it's a feature not a bug": $x \notin L \implies M(x) \notin S$ and
 $x \in L \implies M(x) \in S$ or Segmentation fault

$\overleftrightarrow{\text{CS}}$ "I test in production": $M(x) \in S \iff x \in L$ if $M(x)$ runs

All of These Categories are Weird

Consider the empty set of strings \emptyset . What are arrows $L \rightarrow \emptyset$?

$L \longrightarrow$ Segmentation fault

$\bar{L} \longrightarrow$ (whatever you want)

Consider the set of all strings Σ^* . What are arrows $L \rightarrow \Sigma^*$?

$L \longrightarrow$ (whatever you want)

$\bar{L} \longrightarrow$ Segmentation fault

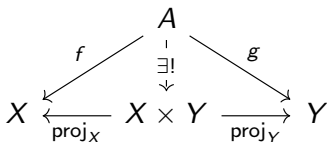
\mathbf{CS} , $\overleftarrow{\mathbf{CS}}$, $\overrightarrow{\mathbf{CS}}$, $\overleftrightarrow{\mathbf{CS}}$ all don't have initial and terminal objects.

Products

The (categorical) product of two **sets** (objects in **Set**) is something we all know:

$$X \times Y = \{(x, y) \mid x \in X; y \in Y\}$$

In general:



Products

So how about in **CS**?

What you probably think: for strings α, β , let $\alpha \bowtie \beta$ be some (fixed, computable) way of encoding the ordered pair (α, β) , then for $L, S \subseteq \Sigma^*$ set

$$L \times S := \{\alpha \bowtie \beta \mid \alpha \in L; \beta \in S\}$$

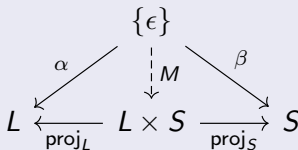
This is great, but it's missing *nuance*.

Products

Theorem (you're kinda right)

Let $\emptyset \neq L, S \subsetneq \Sigma^*$. If $L \times S$ exists, then for $\alpha \in L$ and $\beta \in S$, there must be a unique $\alpha \bowtie \beta \in L \times S$.

Proof (sketch).



then $\alpha \bowtie \beta = M(\epsilon)$. □

Products

How would we write $\text{proj}_L : L \times S \rightarrow L$?

```

1 string projL(string alpha, string beta) {
2     return alpha;
3 }
```

Foolish! If $\alpha \in L$ and $\beta \notin S$, then $\alpha \bowtie \beta \notin L \times S$, but $\text{proj}_L(\alpha \bowtie \beta) = \alpha \in L$, so this is **not** an arrow $L \times S \rightarrow L$.

If $\beta \notin S$, then any choice we make for $\text{proj}_L(\alpha \bowtie \beta)$ with $\alpha \in L$ ruins the necessary uniqueness of the tupling Turing machine. Guess **CS** is hopeless... (its variants too)

We Just Needed More *Nuance*

“[R]emember to look up at the stars and not down at your feet.”

—Stephen Hawking

What starts with an ‘h’ and ends in ‘ope’? **Homotope!**

Say two Turing machines $M, N : L \rightarrow S$ are (homotopy) **equivalent** if

$$M(x) = N(x) \quad \forall x \in L \quad (\text{including segfaults})$$

CS is just slightly weak

We should really think of **CS** (and its variants) as a **(2, 1)-category!**

Definition

A $(2, 1)$ -category \mathcal{C} consists of

- objects (e.g., X, Y, Z, \dots)
- arrows between objects (e.g., $f : X \rightarrow Y$)
- **equivalences** between arrows (e.g., $h : f \simeq g : X \rightarrow Y$)

so that

- arrows can be composed (up to equivalence)
- there is a “do nothing arrow” (up to equivalence)

What's the Point?

*“Arrow composition [et cetera] is defined up to equality in **CS**, so haven't we done nothing at all?”*

Yes, we've done nothing!

THE END

Weaker Limits

Definition (terminal object)

An object 1 of a category is terminal if there is a unique arrow

$$X \dashrightarrow 1$$

I mentioned **CS** does not have a terminal object (nor does its variants).

Definition (2-terminal object)

An object 1 of a $(2, 1)$ -category is terminal if there is a unique arrow

$$X \dashrightarrow 1$$

up to (unique) equivalence.

So, when **CS** is a $(2, 1)$ -category, does it have a terminal object?

Terminal Object

Set (as an ordinary category) has a terminal object: $\{*\}$

Vect $_{\mathbb{R}}$ also has a terminal object in the same way: $\{0\}$

As a $(2, 1)$ -category, does $\overleftrightarrow{\mathbf{CS}}$ have a terminal object? $\emptyset!$

Any arrow $L \rightarrow \emptyset$ must segfault in L , and hence are all equivalent to

```

1 void bar(void) {
2     for (;;); // the compiler is REALLY bad
3 }

```

This also shows $\overleftarrow{\mathbf{CS}}$ has \emptyset as its terminal object.

What about $\overrightarrow{\mathbf{CS}}$ and \mathbf{CS} ? Maps $L \rightarrow \emptyset$ generally do not exist.

Terminal Objects in **CS**

The terminal object would have to be a **singleton** (by its uniqueness).

Let $\{\top\}$ be our candidate terminal object for **CS** (and $\overrightarrow{\text{CS}}$).
By post-composing with

```
1 string baz(string l) {  
2     return l==TOP ? l : BOT;  
3 }
```

we need only consider arrows $L \rightarrow \{\top\}$ sending:

$$L \longrightarrow \top$$

$$\bar{L} \longrightarrow \perp$$

Can you **decide** if **CS** has a terminal object?
Can you **recognise** a terminal object for $\overrightarrow{\text{CS}}$?

Terminal Objects in **CS**

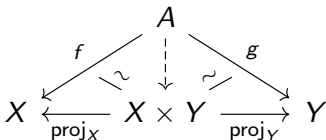
The Halting Problem shows that **CS** does not have a terminal object.

However, the full $(2, 1)$ -subcategory of decidable languages in **CS** does!

Likewise, the full $(2, 1)$ -subcategory of recognisable languages in $\overrightarrow{\mathbf{CS}}$ has the same terminal object.

Products in CS

A product of objects in a $(2, 1)$ -category is a mouthful:



Theorem

Let $\emptyset \neq L \neq \Sigma^*$ be decidable, and $S \neq \Sigma^*$. Then, the following are equivalent:

- i) S is decidable
- ii) $L \times S$ exists in **CS**
- iii) $L \times S$ exists and is decidable in **CS**

Edge Cases

Theorem

Let $\emptyset \neq L \neq \Sigma^*$ be decidable, and $S \neq \Sigma^*$. Then, the following are equivalent:

- i) S is decidable
- ii) $L \times S$ exists in **CS**
- iii) $L \times S$ exists and is decidable in **CS**

The only product with Σ^* that exists is with Σ^* , or else the projection maps do not work.

If $S \neq \Sigma^*$, then $\emptyset \times S = \emptyset$ no matter what S was.

Proof Sketch

If L, S are decidable, and we know $l_0 \in L$ while $l_1 \notin L$, then the projections are given by:

```
1 string projL(string x, string y) { // TODO: projS
2     return memberL(x) && memberS(y) ? x : ELL_1;
3 }
```

It's clear how to decide if $x \in L \times S$ as well.
If $L \times S$ is decidable, then we can decide S :

```
1 bool memberS(string y) {
2     return memberL(projL(ELL_0, y));
3 }
```


Conclusion

- the structure of **CS** is too weak for ordinary category theory
- it's no coincidence that the objects of **CS** are sets of strings
- you need fancy string manipulation (Turing machines) to study **CS**

THE END