"cs is just fancy string manipulation"

but categorical

Zach Goldthorpe

18 October 2019

Strings

What are strings?

```
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

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

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

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

Turing Machines

What are Turing machines supposed to do?

Definitely not machine learning...

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

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

Categories

What is a category?

Definition

A category $\mathscr{C}$ consists of

- objects (e.g., $X, Y, Z, \ldots$)
- arrows between objects (e.g., $f : X \to Y$)

so that

- arrows can be composed (e.g., $g \circ f : X \xrightarrow{f} Y \xrightarrow{g} Z$)
- there is a "do nothing arrow" $\mathrm{id}_X : X \to X$ 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 $f : X \to Y$

Example (vector spaces)

Similarly we have a category $\mathbf{Vect}_{\mathbb{R}}$ where

- the objects of $\mathbf{Vect}_{\mathbb{R}}$ are real vector spaces
- the arrows are the linear transformations $T : V \to W$

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 $L \to S$ are Turing machines M so that

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

(co)Recognition

"You're discriminating against code I write!"

$\overrightarrow{\mathbf{CS}}$ "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`

$\overleftarrow{\mathbf{CS}}$ "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`

$\overleftrightarrow{\mathbf{CS}}$ "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 $\varnothing$. What are arrows $L \to \varnothing$?

$$L \longrightarrow \texttt{Segmentation fault}$$

$$\overline{L} \longrightarrow \text{(whatever you want)}$$

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

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

$$\overline{L} \longrightarrow \texttt{Segmentation fault}$$

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

Products

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

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

In general:

$$\begin{tikzcd} & A \arrow[dl, "f"'] \arrow[d, dashed, "\exists!"] \arrow[dr, "g"] & \\ X & X \times Y \arrow[l, "\mathrm{proj}_X"] \arrow[r, "\mathrm{proj}_Y"'] & Y \end{tikzcd}$$

Products

So how about in **CS**?

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

$$L \times S := \{\alpha \between \beta :|: \alpha \in L; \beta \in S\}$$

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

Products

Theorem (you're kinda right)

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

Proof (sketch).

$$
\begin{tikzcd}
 & \{\epsilon\} \arrow[dl, "\alpha"'] \arrow[d, dashed, "M"] \arrow[dr, "\beta"] & \\
L & L \times S \arrow[l, "\mathrm{proj}_L"] \arrow[r, "\mathrm{proj}_S"'] & S
\end{tikzcd}
$$

then $\alpha \between \beta = M(\epsilon)$. $\square$

Products

How would we write $\mathrm{proj}_L : L \times S \to L$?

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

Foolish! If $\alpha \in L$ and $\beta \notin S$, then $\alpha \between \beta \notin L \times S$, but $\mathtt{projL}(\alpha \between \beta) = \alpha \in L$, so this is not an arrow $L \times S \to L$.

If $\beta \notin S$, then any choice we make for $\mathrm{proj}_L(\alpha \between \beta)$ 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 $M, N : L \to S$ are (homotopy) equivalent if

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

CS is just slightly weak

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

Definition

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

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

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

As a $(2,1)$-category, does $\overleftrightarrow{\mathbf{CS}}$ have a terminal object? $\varnothing$!
Any arrow $L \to \varnothing$ must segfault in L, and hence are all equivalent to

```c
void bar(void) {
    for (;;); // the compiler is REALLY bad
}
```

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

What about $\overrightarrow{\mathbf{CS}}$ and **CS**? Maps $L \to \varnothing$ 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 $\overrightarrow{\mathbf{CS}}$).

By post-composing with

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

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

$$L \longrightarrow \top$$

$$\overline{L} \longrightarrow \bot$$

Can you decide if **CS** has a terminal object?

Can you recognise a terminal object for $\overrightarrow{\mathbf{CS}}$?

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:

$$\begin{tikzcd}
 & A \arrow[dl, "f"'] \arrow[dr, "g"] \arrow[d, dashed] & \\
X & X \times Y \arrow[l, "\mathrm{proj}_X"] \arrow[r, "\mathrm{proj}_Y"'] \arrow[ul, phantom, "\sim"] \arrow[ur, phantom, "\sim"] & Y
\end{tikzcd}$$

Theorem

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

1. *S is decidable*
2. *$L \times S$ exists in* **CS**
3. *$L \times S$ exists and is decidable in* **CS**

Edge Cases

Theorem

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

1. *S is decidable*
2. *$L \times S$ exists in* **CS**
3. *$L \times S$ 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 $\varnothing \times S = \varnothing$ no matter what S was.

Proof Sketch

If L, S are decidable, and we know $\ell_0 \in L$ while $\ell_1 \notin L$, then the projections are given by:

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

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

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

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