"cs is just fancy string manipulation"
but categorical

Zach Goldthorpe

18 October 2019
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 $\Sigma$ ($|\Sigma| > 1$), then our set of strings is $\Sigma^*$. 

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

$$M : \Sigma^* \rightarrow \Sigma^* \sqcup \{"Segmentation fault (core dumped)"\}$$
What are Turing machines supposed to do? *Definitely not machine learning*...

```c
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.
What is a category?

**Definition**

A category \( \mathcal{C} \) 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" \( \text{id}_X : X \to X \) for every \( X \)
Categories

Example (sets)
We have a category $\mathbf{Set}$ where
- the objects of $\mathbf{Set}$ are sets
- the arrows between sets $X, Y$ are the functions $f : X \to Y$

Example (vector spaces)
Similarly we have a category $\mathbf{Vect}_R$ where
- the objects of $\mathbf{Vect}_R$ are real vector spaces
- the arrows are the linear transformations $T : V \to W$
Let’s define the category $\textbf{CS}$ where

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

$$M(x) \in S \iff x \in L$$
"You’re discriminating against code I write!"

CS “QA engineer isn’t paid enough”: \( x \in L \implies M(x) \in S \) and
\[
x \notin L \implies M(x) \notin S \text{ or } \text{Segmentation fault}
\]

CS “it’s a feature not a bug”: \( x \notin L \implies M(x) \notin S \) and
\[
x \in L \implies M(x) \in S \text{ or } \text{Segmentation fault}
\]

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 $L \rightarrow \emptyset$?

$$\begin{align*}
\text{L} & \rightarrow \text{Segmentation fault} \\
\overline{\text{L}} & \rightarrow \text{(whatever you want)}
\end{align*}$$

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

$$\begin{align*}
\text{L} & \rightarrow \text{(whatever you want)} \\
\overline{\text{L}} & \rightarrow \text{Segmentation fault}
\end{align*}$$

CS, ĈS, ĈS, ĈS all don’t have initial and terminal objects.
The (categorical) product of two sets (objects in \textbf{Set}) is something we all know:

\[ X \times Y = \{(x, y) : x \in X; y \in Y\} \]

In general:
So how about in **CS**?
What you probably think: for strings $\alpha, \beta$, let $\alpha \parallel \beta$ be some (fixed, computable) way of encoding the ordered pair $(\alpha, \beta)$, then for $L, S \subseteq \Sigma^*$ set

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

This is great, but it’s missing *nuance*. 
Theorem (you’re kinda right)

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

Proof (sketch).

then $\alpha \downarrow \beta = M(\epsilon)$. 

\[
\begin{array}{c}
\{\epsilon\} \\
\downarrow \\
\alpha \\
\downarrow M \\
L \\
\downarrow \text{proj}_L \\
L \times S \\
\downarrow \text{proj}_S \\
S \\
\downarrow \\
\beta
\end{array}
\]
How would we write $\text{proj}_L : L \times S \to L$?

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

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

If $\beta \notin S$, then any choice we make for $\text{proj}_L(\alpha \not\in L \times S)$ with $\alpha \in L$ ruins the necessary uniqueness of the tupling Turing machine. Guess CS is hopeless... (its variants too)
"[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., \(X, Y, Z, \ldots\))
- 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

$$X \longrightarrow 1$$

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

$$X \longrightarrow 1$$

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: \( \{ * \} \)

**\( \text{Vect}_\mathbb{R} \)** also has a terminal object in the same way: \( \{ 0 \} \)

As a \((2,1)\)-category, does \( \text{← →} \) \( \text{CS} \) have a terminal object? \( \emptyset \)!

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

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

This also shows \( \text{←} \) \( \text{CS} \) has \( \emptyset \) as its terminal object.

What about \( \text{→} \) \( \text{CS} \) and \( \text{CS} \)? Maps \( L \rightarrow \emptyset \) generally do not exist.
Terminal Objects in $\textbf{CS}$

The terminal object would have to be a singleton (by its uniqueness).
Let $\{\top\}$ be our candidate terminal object for $\textbf{CS}$ (and $\overrightarrow{\textbf{CS}}$).
By post-composing with

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

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

$\begin{align*}
L & \rightarrow \top \\
\overline{L} & \rightarrow \bot
\end{align*}$

Can you decide if $\textbf{CS}$ has a terminal object?
Can you recognise a terminal object for $\overrightarrow{\textbf{CS}}$?
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 \(\overrightarrow{CS}\) has the same terminal object.
Products in \textbf{CS}

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

\[
\begin{align*}
A & \xrightarrow{f} X \\
A & \xleftarrow{g} Y \\
X & \xleftarrow{\sim} X \times Y \\
Y & \xrightarrow{\sim} 
\end{align*}
\]

\textbf{Theorem}

\begin{enumerate}
\item \(S\) is decidable
\item \(L \times S\) exists in \textbf{CS}
\item \(L \times S\) exists and is decidable in \textbf{CS}
\end{enumerate}

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

(iii) $L \times S$ exists and is decidable in $\text{CS}$

The only product with $\Sigma^*$ that exists is with $\Sigma^*$, 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 $\ell_0 \in L$ while $\ell_1 \notin L$, then the projections are given by:

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

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

```c
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