Theoretical Computer Science for the Working Category Theorist

Noson S. Yanofsky
Brooklyn College, CUNY

Applied Category Theory Seminar
University of Maryland
May 7, 2020

Abstract

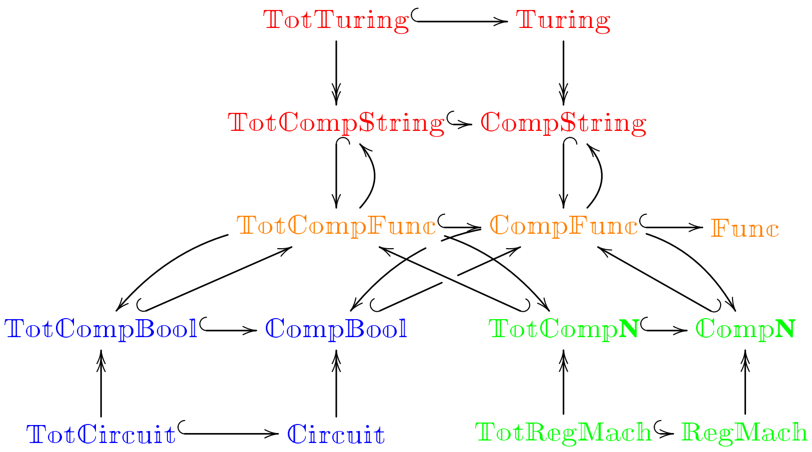
Abstract: This talk is a preview of a forthcoming book in the Applied Category Theory series of Cambridge University Press. The book uses basic category theory to describe all the central concepts and prove the main theorems of theoretical computer science. Category theory, which works with functions, processes, and structures, is uniquely qualified to present the fundamental results of theoretical computer science. We will meet some of the deepest ideas and theorems of modern computers and mathematics, e.g., Turing machines, unsolvable problems, the P=NP question, Kurt Gödel's incompleteness theorem, intractable problems, cryptographic protocols, Alan Turing's Halting problem, and much more. I will report on new things I learned about theoretical computer science and category theory while working on this project.

Outline of Talk

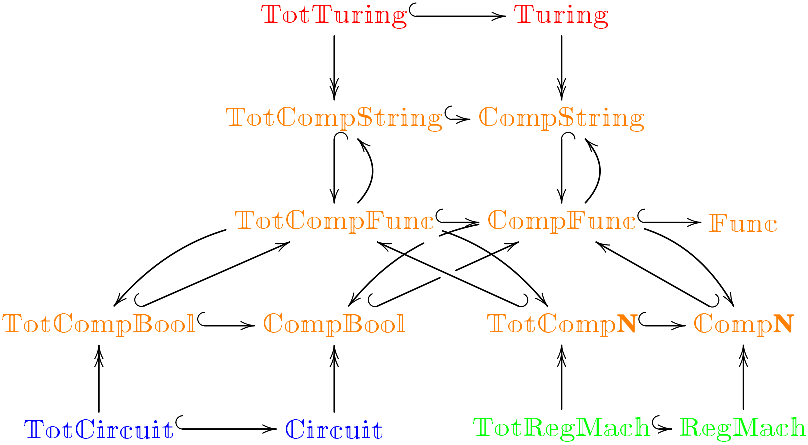
“The Big Picture” of models of computation

$$
\begin{tikzcd}
 & \mathbb{TotTuring} \arrow[r, hook] \arrow[d, two heads] & \mathbb{Turing} \arrow[d, two heads] & \\
 & \mathbb{TotCompString} \arrow[r, hook] \arrow[d, hook] & \mathbb{CompString} \arrow[d, hook] & \\
 & \mathbb{TotCompFunc} \arrow[r, hook] \arrow[u, bend right] \arrow[dl, bend right] \arrow[dr, bend left] & \mathbb{CompFunc} \arrow[r, hook] \arrow[u, bend right] \arrow[dl, bend right] \arrow[dr, bend left] & \mathbb{Func} \\
\mathbb{TotCompBool} \arrow[r, hook] \arrow[ur, hook] & \mathbb{CompBool} \arrow[ur, hook] & \mathbb{TotComp}\mathbf{N} \arrow[r, hook] \arrow[ul, hook] & \mathbb{Comp}\mathbf{N} \arrow[ul, hook] \\
\mathbb{TotCircuit} \arrow[r, hook] \arrow[u, two heads] & \mathbb{Circuit} \arrow[u, two heads] & \mathbb{TotRegMach} \arrow[r, hook] \arrow[u, two heads] & \mathbb{RegMach} \arrow[u, two heads]
\end{tikzcd}
$$

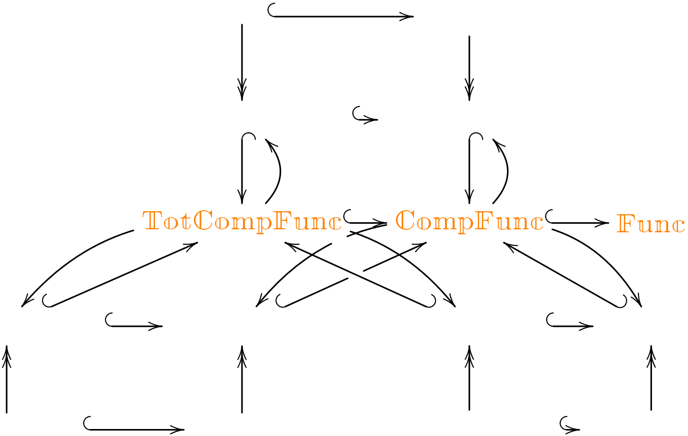
"The Big Picture" of models of computation



"The Big Picture" of models of computation



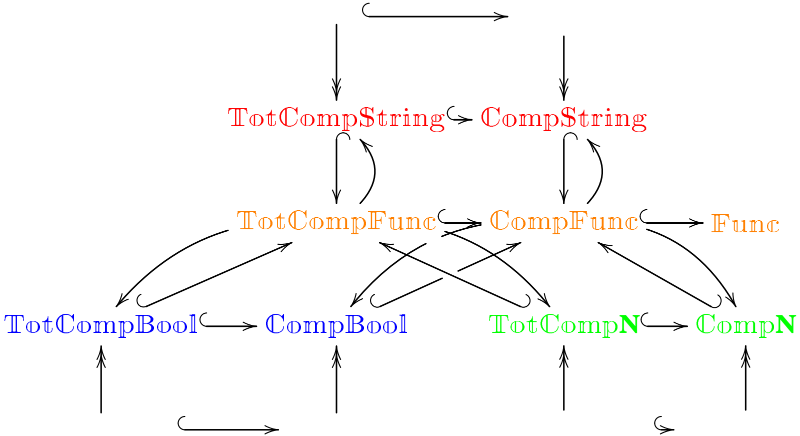
"The Big Picture" of models of computation



Categories of functions

	$\mathbb{TotCompFunc}$	$\mathbb{CompFunc}$	$\mathbb{Func}$
Objects	all types	all types	all types
Morphisms	total computable functions	computable functions	all functions
Structure	symmetric monoidal category	symmetric monoidal category	symmetric monoidal category

“The Big Picture” of models of computation



Categories of functions

	$\mathbb{TotCompString}$	$\mathbb{CompString}$
Objects	powers of *String*	powers of *String*
Morphisms	total computable functions	computable functions
Structure	symmetric monoidal category	symmetric monoidal category

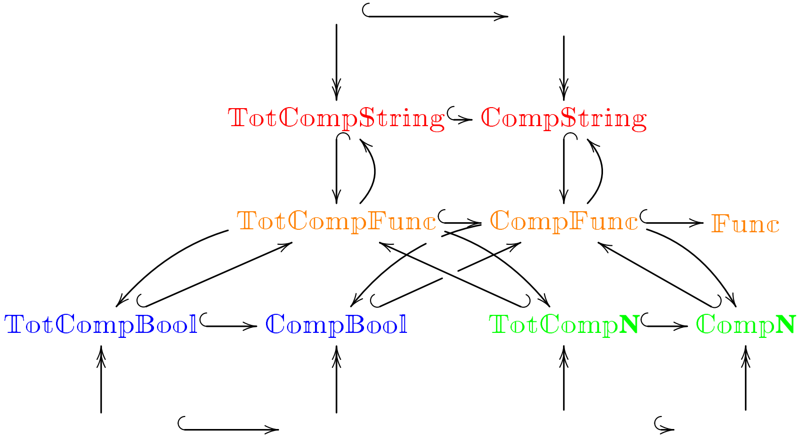
Categories of functions

	$\mathbb{TotComp}\mathbf{N}$	$\mathbb{Comp}\mathbf{N}$
Objects	powers of Nat	powers of Nat
Morphisms	total computable functions	computable functions
Structure	symmetric monoidal category	symmetric monoidal category

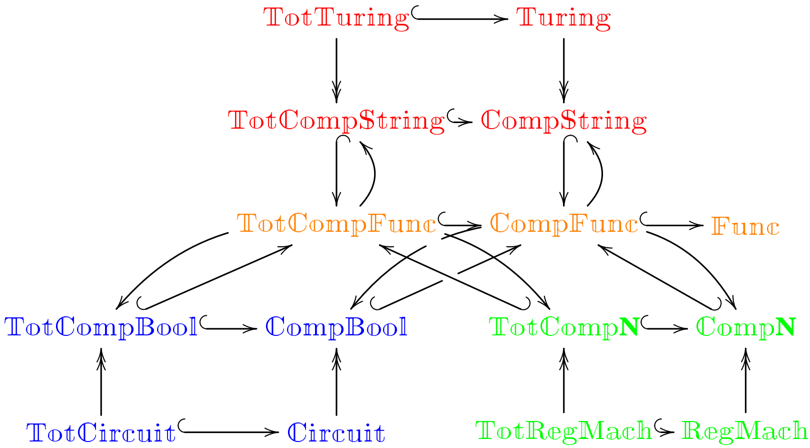
Categories of functions

	$\mathbb{TotCompBool}$	$\mathbb{CompBool}$
Objects	powers of $Bool^*$	powers of $Bool^*$
Morphisms	total computable functions	computable functions
Structure	symmetric monoidal category	symmetric monoidal category

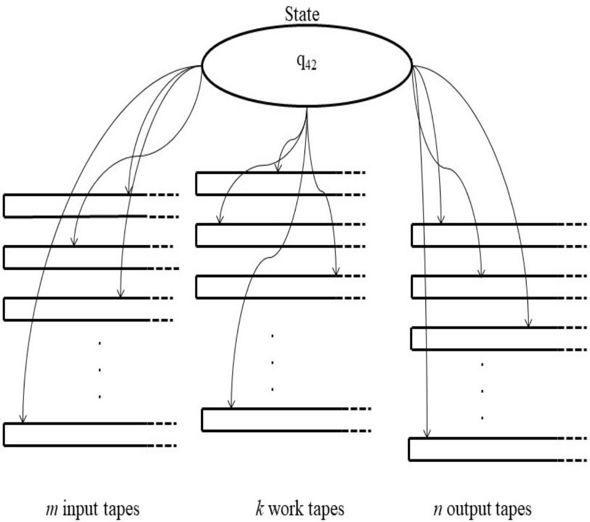
“The Big Picture” of models of computation



"The Big Picture" of models of computation



Turing Machines



Categories of Turing Machines

	$\mathbb{TotTuring}$	$\mathbb{Turing}$
Objects	$\mathbf{N}$	$\mathbf{N}$
Morphisms	total Turing machines	Turing machines
Structure	symmetric monoidal bicategory	symmetric monoidal bicategory

Register Machines

Register machines are methods for manipulating natural numbers. These machines are basically programs in a very simple programming language where variables can only hold natural numbers. The programs use three different types of variables, namely: X_1,X_2,X_3,... called “input variables;” Y_1, Y_2, Y_3, ... called “output variables;” and W_1, W_2, W_3, ... called “work variables.” Register machines employ only the following types of operations on any variable Z:

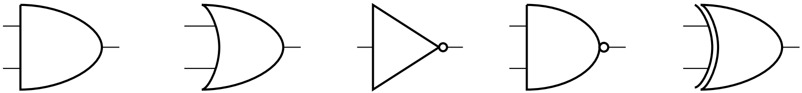
$$Z = Z + 1 \qquad Z = Z - 1 \qquad \texttt{If } Z \neq 0 \texttt{ goto } L, \tag{1}$$

where L is some line number. A program is a list of such statements for various variables. The values in the output variables at the end of an execution are the output of the function. There exist certain register machines for which some of the input causes the machine to go into an infinite loop and have no output values. Other register machines halt for any input.

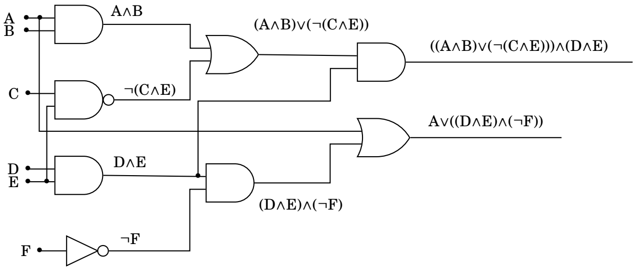
Categories of Register Machines

	$\mathbb{TotRegMach}$	$\mathbb{RegMach}$
Objects	$\mathbf{N}$	$\mathbf{N}$
Morphisms	total register machines	register machines
Structure	symmetric monoidal bicategory	symmetric monoidal bicategory

Circuits



These gates generate all logical circuits such as



This circuit has six inputs and two outputs.

Categories of circuits

	$\mathbb{TotCircuit}$	$\mathbb{Circuit}$
Objects	$\mathbf{N}$	$\mathbf{N}$
Morphisms	total circuit families	circuit families
Structure	braided monoidal bicategory	braided monoidal bicategory

Conclusion:
The more "physical" your models are, the less structure there is in the collection of all such models.

Functors between models of computation

$$
\begin{tikzcd}
 & \mathbb{Turing} \arrow[d] \arrow[ddl, bend right] \arrow[ddr, bend left] & \\
 & \mathbb{CompFunc} & \\
\mathbb{Circuit} \arrow[uur, bend right] \arrow[ur] \arrow[rr, bend right] & & \mathbb{RegMach} \arrow[uul, bend right] \arrow[ul] \arrow[ll, bend right]
\end{tikzcd}
$$

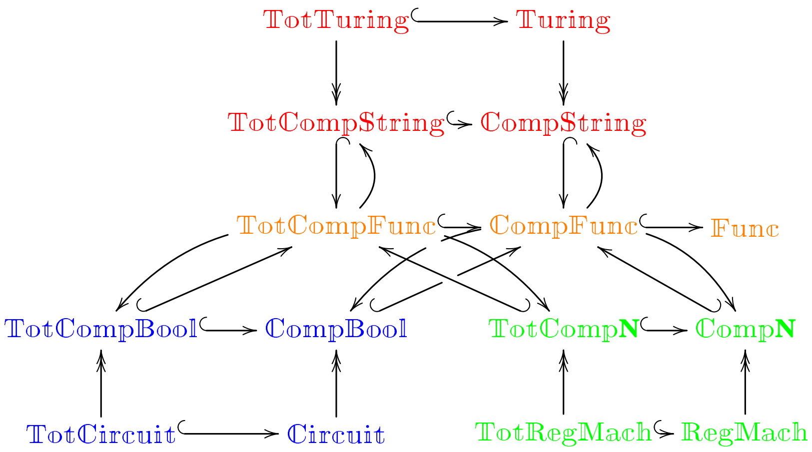
Functors between models of computation and logical formulas

$$
\begin{tikzcd}
& \mathbb{TotTuring} \arrow[dd, "L_t"'] \arrow[ddl, bend right] \arrow[ddr, bend left] & \\
& & \\
& \mathbb{Logic} & \\
\mathbb{TotCircuit} \arrow[uuur, bend right] \arrow[ur, "L_c"] \arrow[rr, bend right] & & \mathbb{TotRegMach} \arrow[uuul, bend left] \arrow[ul, "L_r"'] \arrow[ll, bend right]
\end{tikzcd}
$$

Enumerations

$$
\begin{tikzcd}
\coprod_{Seq,Seq'} Hom_{\mathbb{CompFunc}}(Seq,Seq')) \arrow[dr, two heads] & & \coprod_{m,n} Hom_{\mathbb{RegMach}}(m,n)) \arrow[dl, "\sim"'] \\
\coprod_{m,n} Hom_{\mathbb{Turing}}(m,n) \arrow[r, tail] & d(\mathbf{N}) & \coprod_{m,n} Hom_{\mathbb{Logic}}(m,n)) \arrow[l] \\
\coprod_{m,n} Hom_{\mathbb{PRComp}}\mathbf{N}(Nat^m,Nat^n)) \arrow[ur, two heads] & & \coprod_{m,n} Hom_{\mathbb{Circuit}}(m,n)) \arrow[ul, "\sim"]
\end{tikzcd}
$$

"The Big Picture" of models of computation



Computability Theory: what can and cannot be computed.

$$\begin{tikzcd}
\mathbb{TotTuring} \arrow[r, hook] \arrow[d, two heads] & \mathbb{Turing} \arrow[d, two heads] & \\
\mathbb{TotCompString} \arrow[r, hook] \arrow[d, hook, shift right] & \mathbb{CompString} \arrow[d, hook, shift right] & \\
\mathbb{TotCompFunc} \arrow[r, hook] \arrow[u, bend right] & \mathbb{CompFunc} \arrow[r, hook] \arrow[u, bend right] & \mathbb{Func}
\end{tikzcd}$$

Computability Theory: what can and cannot be computed.

$$\begin{tikzcd}
\mathbb{TotTuring} \arrow[r, hook] \arrow[d] & \mathbb{Turing} \arrow[d] & \\
\mathbb{TotCompFunc} \arrow[r, hook] & \mathbb{CompFunc} \arrow[r, hook] & \mathbb{Func}
\end{tikzcd}$$

Computability Theory: what can and cannot be computed.

$$\begin{tikzcd}
\mathbb{TotTuring} \arrow[r, hook] \arrow[d] & \mathbb{Turing} \arrow[d] & \\
\mathbb{TotCompFunc} \arrow[r, hook] & \mathbb{CompFunc} \arrow[r, hook] & \mathbb{Func}
\end{tikzcd}$$

Computability theory determines if a given morphism in $\mathbb{Func}$ is in $\mathbb{CompFunc}$ or in $\mathbb{TotCompFunc}$. Another way of looking at this is to consider the following functors

$$\begin{tikzcd}
\mathbb{TotTuring} \arrow[rr, hook] \arrow[dr, "D"'] & & \mathbb{Turing} \arrow[dl, "Q"] \\
& \mathbb{Func} &
\end{tikzcd}$$

and ask if a particular morphism in $\mathbb{Func}$ is in the image of Q, or in the image of D, or neither.

The Halting Problem

There is a total morphism in $\mathbb{Func}$

$$Halt : Nat \times Nat \longrightarrow Bool$$

defined as

$$Halt(x,y) = \begin{cases} 1 & : \text{if Turing machine } y \text{ on input } x \text{ halts.} \\ 0 & : \text{if Turing machine } y \text{ on input } x \text{ does not halt.} \end{cases}$$

Theorem

Halt is not a morphism in $\mathbb{TotCompFunc}$.

Proving the undecidability of the Halting problem

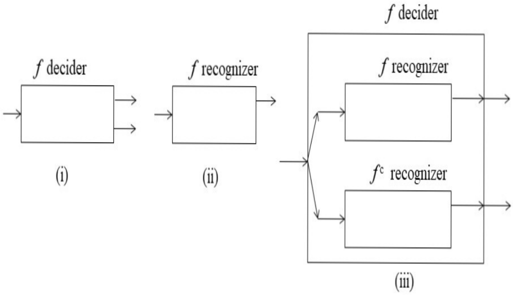


Figure: (i) a decider, (ii) a recognizer, and (iii) a decider built out of two recognizers

Proving the undecidability of the Halting problem

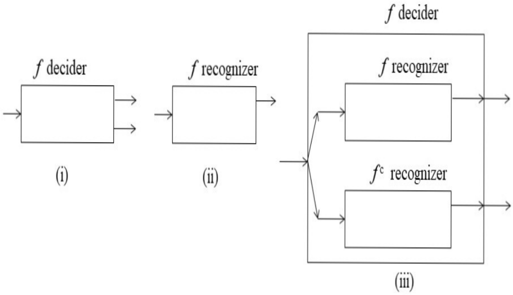


Figure: (i) a decider, (ii) a recognizer, and (iii) a decider built out of two recognizers

$$\begin{tikzcd} Seq \arrow[r, "\Delta"] & Seq \times Seq \arrow[r, "f \times f"] \arrow[rr, bend left, "f \times f^c"] & Bool \times Bool \arrow[r, "id \times NOT"] & Bool \times Bool \arrow[r, "Parallel"] & Bool \end{tikzcd} .$$

Other Undecidable Problems

The **Nonempty domain problem** asks if a given (number of a) Turing machine will have a nonempty domain. There is a total morphism in $\mathbb{Func}$ called $Nonempty : Nat \longrightarrow Bool$ which is defined as follows

$$Nonempty(y) = \begin{cases} 1 & : \text{if Turing machine } y \text{ has a nonempty domain} \\ 0 & : \text{if Turing machine } y \text{ has an empty domain.} \end{cases}$$

We show that the Halting problem reduces to the Nonempty domain problem as in

$$\begin{tikzcd} Nat \times Nat \arrow[rr, "h"] \arrow[dr, "Halt"'] & & Nat \arrow[dl, "Nonempty"] \\ & Bool. & \end{tikzcd}$$

h is in $\mathbb{TotCompFunc}$. If $Nonempty$ was also in that category, then so would $Halt$. Conclusion: $Nonempty$ is not in $\mathbb{TotCompFunc}$.

Other Undecidable Problems

A **reduction** is a way of discussing the relation between two decision problems. Let $f\colon Seq \longrightarrow Bool$ and $g\colon Seq' \longrightarrow Bool$ be two functions in $\mathbb{Func}$. We say that f is **reducible** to g or f **reduces** to g if there exists an $h\colon Seq \longrightarrow Seq'$ in $\mathbb{TotCompFunc}$ such that

$$\begin{tikzcd}
Seq \arrow[rr, "h"] \arrow[dr, "f"'] & & Seq' \arrow[dl, "g"] \\
& Bool &
\end{tikzcd}$$

commutes. We write this as $f \leq g$. .

A categorical way to view reducibility is to form the comma category of the following two functors

$$\mathbb{TotCompFunc} \xhookrightarrow{Inc} \mathbb{Func} \xleftarrow{Const_{Bool}} \mathbb{1}.$$

Other Undecidable Problems

(i) The nonempty domain problem.

(ii) The empty domain problem.

(iii) The equivalent program problem.

(iv) The printing 42 problem.

(v) Rice's theorem. Any nontrivial, semantic property of Turing machines is undecidable.

(vi) Gödel's Incompleteness Theorem. For any consistent logical system which is powerful enough to deal with basic arithmetic, there are statements that are true but unprovable. That is, the logical system is incomplete.

(vii) The Entscheidungsproblem is unsolvable.

Complexity Theory

Complexity theory studies what can be computed efficiently.
First we have to see how to measure computable functions.

(i) $\mathbb{T}\mathrm{ot}\mathbb{C}\mathrm{omp}\mathbb{F}\mathrm{unc} \xrightarrow{\mu_{D,Time}} Hom_{\mathbb{S}\mathrm{et}}(\mathbf{N}, \mathbf{R}^*)$.

Complexity Theory

Complexity theory studies what can be computed <u>efficiently</u>.
First we have to see how to measure computable functions.

(i) $\mathbb{T}\mathbb{o}\mathbb{t}\mathbb{C}\mathbb{o}\mathbb{m}\mathbb{p}\mathbb{F}\mathbb{u}\mathbb{n}\mathbb{c} \xrightarrow{\mu_{D,Time}} Hom_{\mathbb{S}\mathbb{e}\mathbb{t}}(\mathbf{N}, \mathbf{R}^*)$.

(ii) $\mathbb{T}\mathbb{o}\mathbb{t}\mathbb{C}\mathbb{o}\mathbb{m}\mathbb{p}\mathbb{F}\mathbb{u}\mathbb{n}\mathbb{c} \xrightarrow{\mu_{D,Space}} Hom_{\mathbb{S}\mathbb{e}\mathbb{t}}(\mathbf{N}, \mathbf{R}^*)$.

(iii) $\mathbb{T}\mathbb{o}\mathbb{t}\mathbb{C}\mathbb{o}\mathbb{m}\mathbb{p}\mathbb{F}\mathbb{u}\mathbb{n}\mathbb{c} \xrightarrow{\mu_{N,Time}} Hom_{\mathbb{S}\mathbb{e}\mathbb{t}}(\mathbf{N}, \mathbf{R}^*)$.

(iv) $\mathbb{T}\mathbb{o}\mathbb{t}\mathbb{C}\mathbb{o}\mathbb{m}\mathbb{p}\mathbb{F}\mathbb{u}\mathbb{n}\mathbb{c} \xrightarrow{\mu_{N,Space}} Hom_{\mathbb{S}\mathbb{e}\mathbb{t}}(\mathbf{N}, \mathbf{R}^*)$.

Complexity Classes

We use the measures to find complexity classes.
For every subset S of $Hom_{\mathbb{Set}}(\mathbf{N}, \mathbf{R}^*)$, there is a pullback.

$$\begin{tikzcd}
\mathbb{DTIME}(S) \arrow[r, hook] \arrow[d] & \mathbb{TotCompFunc} \arrow[d, "\mu_{D,Time}"] \\
S \arrow[r, hook] & Hom_{\mathbb{Set}}(\mathbf{N}, \mathbf{R}^*).
\end{tikzcd}$$

Similar pullbacks with the other measures form $\mathbb{DSPACE}(S)$, $\mathbb{NTIME}(S)$, and $\mathbb{NSPACE}(S)$ subcategories.

Complexity Classes

There are relations between the complexity measures.

(i) Space vs Time.

$$\begin{tikzcd} \mathbb{TotCompFunc} \arrow[rr, bend left, "\mu_{D,Space}"] \arrow[rr, bend right, "\mu_{D,Time}"'] \arrow[rr, phantom, "\Downarrow"] & & Hom_{\mathbb{Set}}(\mathbf{N}, \mathbf{R}^*) \end{tikzcd} .$$

(ii) Deterministic vs Nondeterministic

(iii) Subsets vs Sets.

Complexity classes and their inclusion functors.

We can see all three of these “dimensions” in one diagram. Given $T \subseteq S \subseteq Hom_{\mathbb{Set}}(\mathbf{N}, \mathbf{R}^*)$ we have

$$
\begin{tikzcd}
 & \mathbb{DSPACE}(S) \arrow[rr, hook] & & \mathbb{NSPACE}(S) \\
\mathbb{DTIME}(S) \arrow[ur, hook] \arrow[rr, hook] & & \mathbb{NTIME}(S) \arrow[ur, hook] & \\
 & \mathbb{DSPACE}(T) \arrow[uu, hook] \arrow[rr, hook] & & \mathbb{NSPACE}(T) \arrow[uu, hook] \\
\mathbb{DTIME}(T) \arrow[uu, hook] \arrow[ur, hook] \arrow[rr, hook] & & \mathbb{NTIME}(T). \arrow[uu, hook] \arrow[ur, hook] &
\end{tikzcd}
$$

Complexity classes and their inclusion functors.

Some common subsets of $Hom_{\mathbb{Set}}(\mathbf{N}, \mathbf{R}^*)$ that are closed under addition are *Poly* consisting of all functions that are polynomial or less, *Const* consisting of all constant functions, *Exp* consisting of all functions that are exponential or less, *Log* consisting of all functions that are logarithmic or less. These subsets are included in each other as

$$Const \hookrightarrow Log \hookrightarrow Poly \hookrightarrow Exp \hookrightarrow Hom_{\mathbb{Set}}(\mathbf{N}, \mathbf{R}^*).$$

Complexity classes and their inclusion functors.

$$
\begin{tikzcd}
 & \mathbb{DSPACE}(\mathit{Exp}) \arrow[rr, hook] & & \mathbb{NSPACE}(\mathit{Exp}) \\
\mathbb{DTIME}(\mathit{Exp}) \arrow[ur, hook] \arrow[rr, hook] & & \mathbb{NTIME}(\mathit{Exp}) \arrow[ur, hook] & \\
 & \mathbb{DSPACE}(\mathit{Poly}) \arrow[uu, hook] \arrow[rr, hook] & & \mathbb{NSPACE}(\mathit{Poly}) \arrow[uu, hook] \\
\mathbb{DTIME}(\mathit{Poly}) \arrow[uu, hook] \arrow[ur, hook] \arrow[rr, hook] & & \mathbb{NTIME}(\mathit{Poly}) \arrow[uu, hook] \arrow[ur, hook] & \\
 & \mathbb{DSPACE}(\mathit{Log}) \arrow[uu, hook] \arrow[rr, hook] & & \mathbb{NSPACE}(\mathit{Log}) \arrow[uu, hook] \\
\mathbb{DTIME}(\mathit{Log}) \arrow[uu, hook] \arrow[ur, hook] \arrow[rr, hook] & & \mathbb{NTIME}(\mathit{Log}) \arrow[uu, hook] \arrow[ur, hook] & \\
 & \mathbb{DSPACE}(\mathit{Const}) \arrow[uu, hook] \arrow[rr, hook] & & \mathbb{NSPACE}(\mathit{Const}) \arrow[uu, hook] \\
\mathbb{DTIME}(\mathit{Const}) \arrow[uu, hook] \arrow[ur, hook] \arrow[rr, hook] & & \mathbb{NTIME}(\mathit{Const}). \arrow[uu, hook] \arrow[ur, hook] &
\end{tikzcd}
$$

Decision Problems

The reductions used in basic complexity theory are called **polynomial reduction**. Let $f: Seq \longrightarrow Bool$ and $g: Seq' \longrightarrow Bool$ be two decision problems in $\mathbb{TotCompFunc}$. We say that f is **polynomial reducible** to g if there is an $h: Seq \longrightarrow Seq'$ in $\mathbb{DTIME}(Poly)$ such that

$$\begin{tikzcd} Seq \arrow[rr, "h"] \arrow[dr, "f"'] & & Seq' \arrow[dl, "g"] \\ & Bool & \end{tikzcd}$$

commutes. We write this as $f \leq_p g$.

To form the category $\mathbb{Decision}$ of decision problems and polynomial reductions, consider the comma category of

$$\begin{tikzcd} \mathbb{DTIME}(Poly) \arrow[r, hook, "Inc"] & \mathbb{TotCompFunc} & \mathbb{1} \arrow[l, "Const_{Bool}"'] \end{tikzcd}$$

Decision Problems

There are two subcategories of $\mathbb{TotCompFunc}$ that are of interest: $\mathbb{DTIME}(\mathit{Poly})$ and $\mathbb{NTIME}(\mathit{Poly})$. These are all deterministic polynomial computable functions, and all nondeterminisitic polynomial functions, respectively. They sit in the diagram

$$\begin{tikzcd}
 & \mathbb{TotCompFunc} & \\
\mathbb{DTIME}(\mathit{Poly}) \arrow[ur, hook, "\mathit{Inc}"] \arrow[r, hook, "\mathit{DInc}"] \arrow[dr, hook, "\mathit{Id}"'] & \mathbb{NTIME}(\mathit{Poly}) \arrow[u, hook] & \mathbb{1} \arrow[ul, "\mathit{Const}_{\mathit{Bool}}"'] \arrow[l, "\mathit{Const}_{\mathit{Bool}}"'] \arrow[dl, "\mathit{Const}_{\mathit{Bool}}"] \\
 & \mathbb{DTIME}(\mathit{Poly}). \arrow[u, hook] &
\end{tikzcd}$$

These inclusions induce:

$$\mathbb{P} \hookrightarrow \mathbb{NP} \hookrightarrow \mathbb{Decision}.$$

Decision Problems

A morphism from one decision problem f to another decision problem g means g is as hard or harder than f. The hardest problems in a complexity class is a problem that every problem maps to it. Such problems are called **complete**. The collection of all complete problems in a complexity class is the subcategory of weak terminal objects of that category.

Lawvere-Cantor Diagonalization Theorem

Theorem

Let $\mathbb{A}$ be a category with a terminal object and binary products. Let y be an object in $\mathbb{A}$. If $\alpha : y \longrightarrow y$ is a morphism in $\mathbb{A}$ and α does not have a fixed point, then for every object a and for every $f : a \times a \longrightarrow y$, there exists a morphism $g : a \longrightarrow y$ such that g is not representable in f.

$$
\begin{tikzcd}
 & a \times a \arrow[r, "f"] & y \arrow[dr, "\alpha"] & \\
a \arrow[ur, "\Delta"] \arrow[rrr, "g"'] & & & y.
\end{tikzcd}
$$

Lawvere-Cantor Diagonalization Theorem

The contrapositive is also important.

Theorem

Let $\mathbb{A}$ be a category with a terminal object and binary products. Let y be an object in $\mathbb{A}$. If there exists an object a, and a morphism $f : a \times a \longrightarrow y$ such that every morphism $g : a \longrightarrow y$ is representable in f, then every $\alpha : y \longrightarrow y$ has a fixed point.

$$
\begin{tikzcd}
 & a \times a \arrow[r, "f"] & y \arrow[dr, "\alpha"] & \\
a \arrow[ur, "\Delta"] \arrow[rrr, "g"'] & & & y.
\end{tikzcd}
$$

Conclusion:

Almost every instance of self reference and diagonalization proof falls into this simple format.

Applications

(i) There does not exist an onto function from the set of natural numbers, $\mathbf{N}$, to the powerset of natural numbers, $\mathscr{P}(\mathbf{N})$.

(ii) The unsolvability of the Halting problem (again).

(iii) There is a computable function that is not primitive recursive.

(iv) There is a total unary function that is not computable.

(v) The recursion theorem.

Applications

(i) There does not exist an onto function from the set of natural numbers, **N**, to the powerset of natural numbers, $\mathscr{P}(\mathbf{N})$.
(ii) The unsolvability of the Halting problem (again).
(iii) There is a computable function that is not primitive recursive.
(iv) There is a total unary function that is not computable.
(v) The recursion theorem.
(vi) The space hierarchy theorem: There are computable functions which can be computed in Space(h) but not in less space.
(vii) The time hierarchy theorem: There are computable functions which can be computed in Time(h) but not in much less time.
(viii) The Baker-Gill-Solovay theorem: There exists oracles A and B such that $P^A = NP^A$ and $P^B \neq NP^B$.
(ix) (Ladner's Theorem: If $P \neq NP$ then there are an infinite number of complexity classes between them.)

Other Fields of Theoretical Computer Science

(i) Formal Language Theory
(ii) Cryptography
(iii) Kolmogorov Complexity Theory
(iv) Algorithms

Cryptography

Alice wants to communicate with Bob. The encoders and the decoders must be total and computabile, i.e., morphisms in $\mathbb{TotCompFunc}$. We shall work with some subcategory with the same objects called $\mathbb{Easy}$. Any morphism not in $\mathbb{Easy}$ will be in **Hard**. Encoders $e\colon SeqA \longrightarrow SeqB$ are in $\mathbb{Easy}$. The intended receiver of the secret message should be able to easily decode the message. In other words, the decoders $d\colon SeqB \longrightarrow SeqA$ are also in $\mathbb{Easy}$. It should not be hard to decode, rather, it should be hard *to find the right decoder*. The notion of a trapdoor function will be helpful. It is hard to find the right decoder. However, with the right key, it will be easy to find the right decoder.

Cryptography

Definition

A **cryptographic protocol** *that encodes data of type SeqA into data of type SeqB consists of*

(i) *a set of "encoder" functions,* $Enc \subseteq Hom_{\mathbb{E}\mathrm{asy}}(SeqA, SeqB)$,

(ii) *a set of "decoder" functions,* $Dec \subseteq Hom_{\mathbb{E}\mathrm{asy}}(SeqB, SeqA)$,

(iii) *an "inverter" function* $INV : Enc \longrightarrow Dec$ *in* **Hard** *such that for all* $e \in Enc$*, there is a* $d = INV(e)$ *that satisfies* $d \circ e = Id_{SeqA}$,

(iv) *a "key" function* $KEY : Enc \longrightarrow Seq$ *in* **Hard** *such that for all* $e \in Enc$*, there is a* $k_e = KEY(e)$*, and*

(v) *a "trapdoor" function* $TRP : Seq \longrightarrow Dec$ *in* $\mathbb{E}\mathrm{asy}$ *satisfying*

Cryptography

$$\begin{tikzcd} & Seq \arrow[dr, "TRP"] & \\ Enc \arrow[ur, "KEY"] \arrow[rr, "INV"'] & & Dec \end{tikzcd}$$

i.e., for every $e \in Enc$ there is a “key” $k_e \in Seq$ such that $TRP(k_e) = INV(e)$.

Cryptography

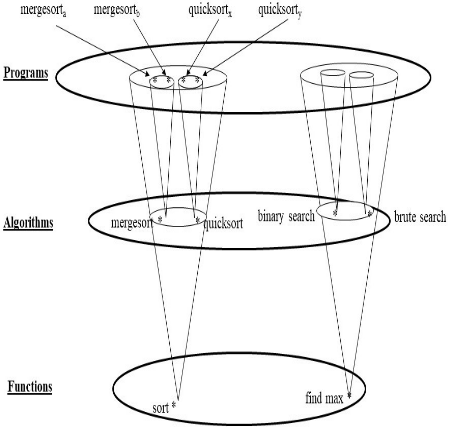
$$\begin{tikzcd}
& Seq \arrow[dr, "TRP"] & \\
Enc \arrow[ur, "KEY"] \arrow[rr, "INV"'] & & Dec
\end{tikzcd}$$

i.e., for every $e \in Enc$ there is a “key” $k_e \in Seq$ such that $TRP(k_e) = INV(e)$.

Conclusion:

Almost every major cryptographic protocol falls into this simple format.

Algorithms



$$\text{Program} \twoheadrightarrow \text{Algorithm} \twoheadrightarrow \text{CompFunc}$$

Lessons Learned

(i) Rather than considering the set of isolated decision problems, we worked with the category of decision problems and their reductions. They are comma categories. In complexity theory, these categories form complexity classes.

(ii) Complete problems for a complexity class are weak terminal objects in the category of decision problems. They form a subcategory of the decision problems.

(iii) There are a few major results (e.g., Halting is undecidable, SAT is NP-Complete, the recursion theorem, etc.) and many corollaries are derived from those results. This follows from the fact that decision problems are a comma category where it is easy to express reductions.

(iv) Adjoint functors <u>did not</u> play a major role in the tale that we told. In computer science, there are many equivalent ways of making constructions. This is not conducive to universal properties.

Lessons Learned

(v) We, however, did use Kan extensions as ways of finding complicated minimization and maximazation functors.

(vi) We defined a functor L_t from the symmetric monoidal bicategory of total Turing machines to the symmetric monoidal bicategory of families of sequences of logical formula. This functor was used to describe the workings of a Turing machine with logical formulas. The functor L_t and extensions of the functor were used in the proofs of the following theorems that relate computation and logic:

  (i) Gödel's Incompleteness Theorem.
  (ii) The unsolvability of the Entscheidungsproblem.
  (iii) The Cook-Levin Theorem.

Lessons Learned

The following “density relation” arose many times.

$$\begin{tikzcd}
\mathbb{A} \arrow[rr, hook, "Inc"] \arrow[dr] & & \mathbb{B} \arrow[ll, bend right, "F"'] \arrow[dl] \\
& \mathbb{C}. &
\end{tikzcd}$$

where both triangles commute and $F \circ Inc = Id_{\mathbb{A}}$ but, in general $Inc \circ F \neq Id_{\mathbb{B}}$. What this means is that for every b in $\mathbb{B}$, $F(b)$ is not the same as an a in $\mathbb{A}$, but is the same in relation to the functors to $\mathbb{C}$.

Weakening of reflective equivalence.

Lessons Learned

(i) Every Turing machine is equivalent to a Turing machine in $\mathbb{Turing}(1,1)$.

(ii) Every logic circuit has an equivalent NAND logic circuit.

(iii) Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

(iv) Savitch's theorem: Every $\mathbb{NPSPACE}$ computation has an equivalent $\mathbb{PSPACE}$ computation.

(v) Every $\mathbb{NP}^A$ computation is equivalent to a $\mathbb{NPSPACE}$ computation where A is PSPACE-complete.

(vi) Every $\mathbb{PSPACE}$ computation is equivalent to a $\mathbb{P}^A$ computation where A is PSPACE-complete.

(vii) Every nondetermistic finite automaton has an equivalent deterministic finite automaton[1].

[1] This relationship between nondeterminism and determinism is not universal. Here are two examples where it fails. (i) Not every nondeterministic pushdown automaton is equivalent to a deterministic pushdown automaton. (ii) If $\mathbb{P} \neq \mathbb{NP}$ then not every nondeterministic polynomial algorithm has an equivalent deterministic polynomial algorithm.

Lessons Learned

While our aim was to be as categorical as possible, we found that twice we had to go outside the bounds of categories:

(i) Enumerations of models of computation or of computable functions are not functorial. They neither respect sequential composition nor parallel composition.

(ii) Complexity measures of models of computation or of computable functions are not functorial. They neither respect sequential composition nor parallel composition.

The End

Thank You!