Category Theory for Software Modeling and Design

Angeline Aguinaldo [1, 2]

[1]University of Maryland, College Park

[2]Johns Hopkins University Applied Physics Laboratory

October 29, 2020

About Me

Angeline Aguinaldo

University of Maryland, College Park
- Computer Science
- 3rd Year Ph.D. Student

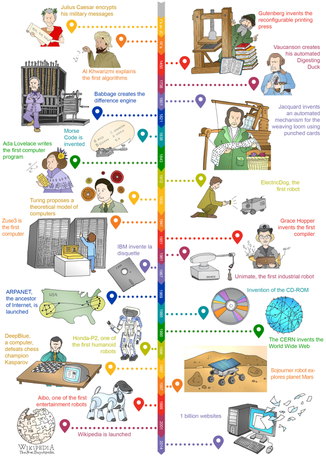
Johns Hopkins University Applied Physics Laboratory (JHUAPL)
- Software Engineer
- 2017 - Present

Drexel University
- B.S. Biomedical Engineering
- M.S. Electrical Engineering
- Graduated 2017

Measuring software design complexity

"People have been programming computers for more than 80 years now and yet software design is still basically a black art"

John Ousterhout



Measuring software design complexity

Some research questions:

- How do we know what we have built is unique?
- How do we measure complexity of software design?
  - How modular have we made our system?
  - Are there opportunities for parallelization?

To start, we need a representation of systems that can support these questions.

Current approaches to modeling software design complexity

- **Based on control flow graph**. McCabe, T. J., 1976, "A Complexity Measure," IEEE Trans. Softw. Eng., 2(4), pp. 308–320.
- **Based on function inputs/outputs and lines of source code**. Albrecht, A. J., and Gaffney, J. E., Jr., 1983, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Trans. Softw. Eng., SE-9(6), pp. 639–648.
- **Based on number of operators in source code**. Halstead, Maurice H. Elements of Software Science. New York: Elsevier, 1977. Print.
- **UML software architecture notation**. J. Rumbaugh, et al, The Unified Modeling Language Reference Manual, Addison-Wesley, 1999.

Decomposing a Software Problem

"Software design involves inventing software concepts to model concepts in a problem space."

Jack Reeves, 1992

Decomposing a Software Problem

Example Software Project:
"I want to a program that can tell me how many animals are in this image"

"Oh, I want it to be able to tell me what type of animals are in it"

"And I want it to tell me the location of these animals on Earth"

Defining data models and concepts in our software

What's an image?

Def. An *image*, $i \in I$, is defined as a $N \times M \times 3$ matrix with matrix elements, $\{z \mid 0 \geq z \geq 255 \in \mathbb{Z}\}$.

What's an animal and what data comes with being an animal?

Def. *Animal names*, N, is a set {"cat", "dog", "elephant", "cow"}.

Def. *Locations*, L, refers to the set of geographic coordinates $\{(x, y) \mid x \in \text{Longitude (Deg)}, y \in \text{Latitude (Deg)}\}$.

Def. *Image ID*, D, is the set of unique identifier for each image, I.

Def. *Animals*, A, is the set $\{(n, l, d) \mid n \in N, l \in L, d \in D\}$.

Defining data models and concepts in our software

We need a function that will give us a set of images (of animals) from a source image.

Def. $\phi : I \to I$ that takes an element from the set of images and returns a set of images.

image_001.png

Defining data models and concepts in our software

We need a function that will translate an image to an animal.

Def. $\rho : I \to A$ that maps images to animals.

➡ ("cow",
(38.98583, -76.93733),
"image_001.png")

We might consider ρ as consisting of multiple maps.

$\rho_1 : I \to N$ $\rho_2 : I \to L$ $\rho_3 : I \to D$

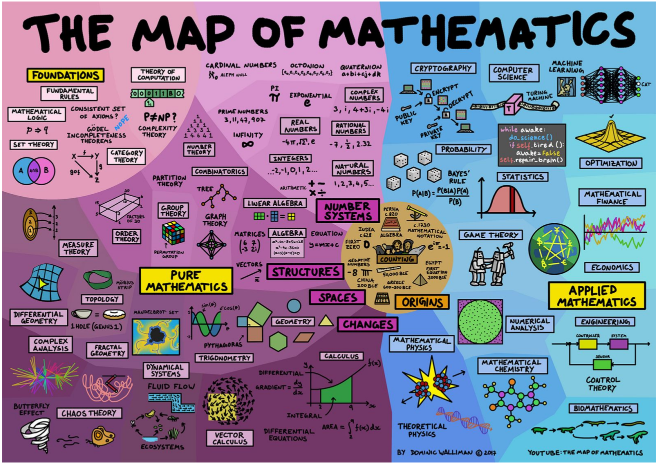
So ρ can be thought of as the product of these maps.

$$\rho = \rho_1 \times \rho_2 \times \rho_3$$

To go from an image to a set of animals, I compose

$$\rho \circ \phi : I \to A$$

What is Category Theory?



"Map of Mathematics", *Domain of Science*, YouTube

What is Category Theory?

- Samuel Eilenberg and Saunders Mac Lane introduced the concepts of categories, functors, and natural transformations from 1942-45 in their study of algebraic topology.
- Category theory is intended to be a unifying framework to describe all mathematics, specifically the functions, transforms, or morphisms that preserve structure.
- Applied category theory is a growing field in which mathematicians and engineers use concepts from category theory to model structures found in the real-world. Examples include hetergeneous sensor fusion[1], software design[2], biology and music[3].

(1) M. M. Kokar, K. Baclawski, and H. Gao, "Category theory-basedsynthesis of a higher-level fusion algorithm: An example," in2006 9thInternational Conference on Information Fusion, 2006, pp. 1–8
(2) S. P. Kovalyov, "Category-theoretic approach to software systems de-sign,"Journal of Mathematical Sciences, vol. 214, pp. 814–853, 2016.
(3) Wong JY, McDonald J, Taylor-Pinney M, Spivak DI, Kaplan DL, Buehler MJ. Materials by Design: Merging Proteins and Music. Nano Today. 2012 Dec 1;7(6):488-495. doi: 10.1016/j.nantod.2012.09.001. PMID: 23997808; PMCID: PMC3752788.

Category

A category $\mathbb{C}$ is a class of objects $A, B, C, \ldots$ and a sets of morphisms, or arrows, $f, g, \ldots$. For every ordered pair A, B of objects there is a set $Hom_{\mathbb{C}}(A, B)$ of morphisms from A to B. These objects and arrows satisfy the following axioms:

- For every object, there exists an *identity arrow*.

$$\forall A \in \mathbb{C},\ id_A : A \to A \tag{1}$$

- The arrows are *composable* where the tail of an arrow exactly equals the head of the previous arrow.

$$f : A \to B, \quad g : B \to C \implies g \circ f : A \to C \tag{2}$$

- The composition of arrows is *associative*.

$$\begin{aligned} f : A \to B, \quad g : B \to C, \quad h : C \to D \\ \implies (h \circ g) \circ f = h \circ (g \circ f) \end{aligned} \tag{3}$$

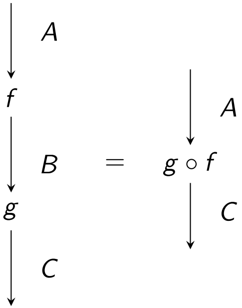
- Identity arrows act as a left and right *unitor* of composition.

$$f : A \to B \implies id_B \circ f = f = f \circ id_A \tag{4}$$

Diagrammatic Syntax

Category theory provides a algebraic system for functions and their compositions.

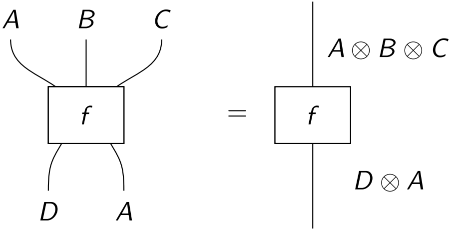
$$
\begin{tikzcd}
A \arrow[d, "f"] \arrow[dd, bend right=60, "g \circ f"'] \\
B \arrow[d, "g"] \\
C
\end{tikzcd}
$$



Conventional syntax for relationships between inputs/outputs and functions (**left**). Poincare dual representation, where functions are nodes and inputs/outputs are arrows (**right**).

Symmetric Monoidal Categories

Additional mathematical structure (tensor product $\otimes$) can be added to support multiple inputs and multiple outputs. This diagram is often referred to as a *string diagram* or *wiring diagram*.



Symmetric Monoidal Categories

A symmetric monoidal category, $\mathbb{M}$, is a category with:

- A *unit object*

$$I \in \mathbb{M} \tag{5}$$

- A functor, called the *tensor product*, which is the product of $\mathbb{M}$ with itself

$$\otimes : \mathbb{M} \times \mathbb{M} \to \mathbb{M} \tag{6}$$

- with the associative isomorphism

$$a_{X,Y,Z} = (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z) \tag{7}$$

- and a left and right *unitor* isomorphisms

$$\rho_l : I \otimes X \to X \qquad \rho_r : X \otimes I \to X \tag{8}$$

- and a braiding isomorphism

$$B_{X,Y} : X \otimes Y \to Y \otimes X \tag{9}$$

- such that the braiding isomorphism obeys the following identity (symmetric)

$$B_{Y,X} \circ B_{X,Y} = I_{X \otimes Y} \tag{10}$$

Functors

A functor $F : \mathbb{X} \to \mathbb{Y}$, where $\mathbb{X}$ and $\mathbb{Y}$ are categories, maps both

- **Objects**. An object in $\mathbb{X}$ $\to$ some object in $\mathbb{Y}$.
- **Arrows**. Arrow(s) between two objects in $\mathbb{X}$ $\to$ arrow(s) between the corresponding objects in $\mathbb{Y}$ such that,

$$F(id_X) = id_{FX} \tag{11}$$

$$F(g \circ f) = Fg \circ Ff \tag{12}$$

where, f and g are composable arrows in $\mathbb{X}$.

How can these structures help us?

Software Concept	Software Questions	Mathematical Translation
Modularity	"Can I use tool A in place of tool B?"	Let $\mathbb{C}$ category with objects A, B, C, D and arrows $f : A \to B, g : C \to D$ and $h_1 : B \to C$, $h_2 : B \to C$. Does $g \circ h_1 \circ f \stackrel{?}{=} g \circ h_2 \circ f$ In other words, are h_1 and h_2 in $Hom(B, C)$? Are these expressions equal up to isomorphism?
Interoperability	"Can I pass data from system A to system B?"	Let $s_1 : A \to C, s_2 : C \to E$ be arrows in $\mathbb{C}$. Does $s_2 \circ s_1 \in C$?
Performance	"Can I run subcomponent A in parallel with subcomponent B?"	Let $f : A \to B, g : C \to D, id_A, id_B, id_C, id_D$ be arrows in $\mathbb{C}$. Are the following equivalent up to isomorphism? $(f \otimes id_C) \circ (id_D \otimes g)$ $\stackrel{?}{=} (id_A \otimes id_C) \circ (f \otimes g) \circ (id_B \otimes id_D)$ $\stackrel{?}{=} (id_A \otimes g) \circ (f \otimes id_D)$

Robot Programming and Category Theory

Bloomberg via Getty Images. Robots weld car body components for vehicles at a BMW assembly plant in Greer, S.C., May 10, 2018.

RoboCat Framework

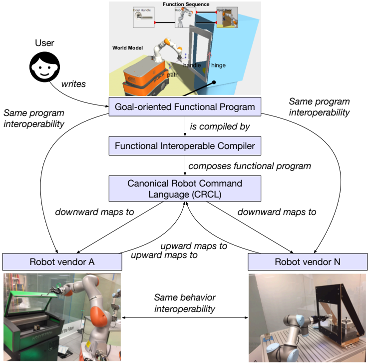
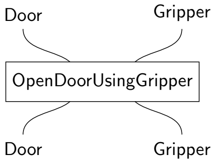


Figure: RoboCat framework consists of a goal-oriented functional programming environment, a functional interoperable compiler, and the mapping to the Canonical Robot Command Language (CRCL) and robot-specific APIs

Aguinaldo, B. Pollard, A. Canedo. G. Quiros, W. Regli. "RoboCat: A category theoretic framework for robotic interoperability using goal-oriented programming." 2020. In Submission.

Physical Representation, $\mathbb{C}_1$

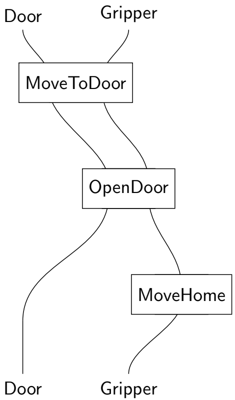
$\mathbb{C}_1$ is a symmetric monoidal category that refers to the physical representation of the world. This consists of objects, $S^1 = \{\texttt{Door}, \texttt{Gripper}, \ldots\}$. The arrows are the set of all possible commands that would enlist these two resources.



(13)

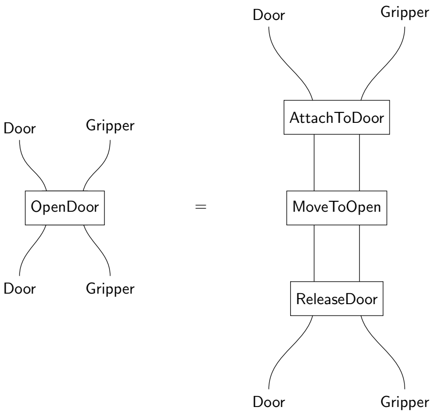
Software Representation, $\mathbb{C}_2$

$\mathbb{C}_2$ is a symmetric monoidal category that refers to the informational or virtual resources. This consists of objects, $S^2 = \{\texttt{Door}, \texttt{Gripper}\}$ and arrows where each arrow represents the skills needed to complete the desired action.



(14)

Software Representation, $\mathbb{C}_2$



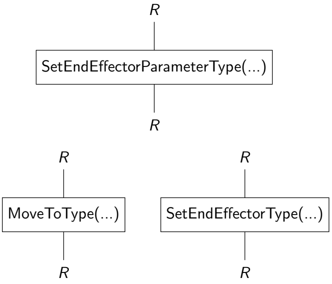
(15)

In $\mathbb{C}_2$, we can define relations that encode semantic equivalences between skills and composition of skills.

Robot Commands, $\mathbb{C}_3$

```
<CRCLCommand xsi:type="MoveToType">
  <MoveStraight>true</MoveStraight>
    <EndPosition>
      <Point>
        <X>8.25</X> <Y>1</Y> <Z>0.5</Z>
      </Point>
      <XAxis>
        <I>1</I> <J>0</J> <K>0</K>
      </XAxis>
      <ZAxis>
        <I>0</I> <J>0</J> <K>-1</K>
      </ZAxis>
    </EndPosition>
</CRCLCommand>


<CRCLCommand
    xsi:type="SetEndEffectorParametersType">
  <ParameterSetting>
    <ParameterName>mode</ParameterName>
    <ParameterValue>grip</ParameterValue>
  </ParameterSetting>
</CRCLCommand>
<CRCLCommand xsi:type="SetEndEffectorType">
  <Setting>1.0</Setting>
</CRCLCommand>
```



(16)

$\mathbb{C}_3$ is a symmetric monoidal category that refers to the category of CRCL commands. This is a category with one object per robot platform, $R \in S^3$, and arrows where arrows are fully parameterized CRCL commands.

Functors

By designing the functions and their interfaces (i.e. categories) first, we have isolated the parts of the code that will be responsible for handling variations needed for interoperability, i.e. the functors F and G.

$F : \mathbb{C}_1 \to \mathbb{C}_2$. This is the human-performed task of developing a 3D model of the physical environment.

$G : \mathbb{C}_2 \to \mathbb{C}_3$. This maps

- **Objects.** Software objects to robot platforms
- **Arrows.** Follow Table 1 (next slide)

Functor G

C_2 Arrow	C_3 Arrow	Pseudocode

MoveToDoor	MoveTo(...)	`let` *pose* `= GetGripperNaturalPose(Gripper)`
		`let` *position* `= GetDoorHandlePosition(Door)`
		`BuildMoveToCRCL(`*pose*`,` *position*`)`

MoveToOpen	MoveTo(...)	`let` *pose* `= GetGripperCurrentPose(Gripper)`
		`let` *position* `= GetDoorOpenPosition(Door)`
		`BuildMoveToCRCL(`*pose*`,` *position*`)`

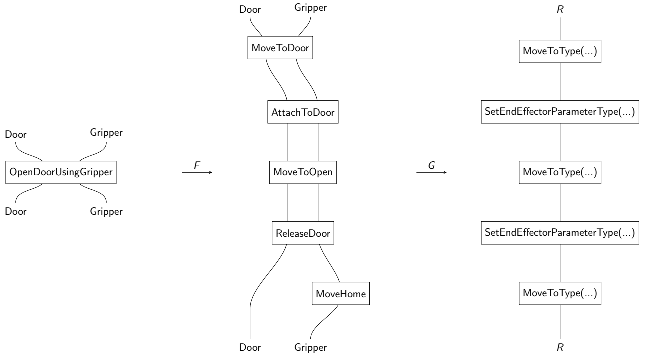
MoveHome	MoveTo(...)	`let` *pose* `= GetGripperNaturalPose(Gripper)`
		`let` *position* `= GetGripperHomePosition(Door)`
		`BuildMoveToCRCL(`*pose*`,` *position*`)`

AttachToDoor	SetEndEffector(...) $\circ$
	SetEndEffectorParameter(...)
		`let` *endEffectorMode* `= GetGripperAttachMode(Gripper)`
		`BuildSetEndEffectorParameterCRCL(`*endEffectorMode*`)`
		`let` *endEffectorSetting* `= GetGripperAttachSetting(Gripper)`
		`BuildSetEndEffectorCRCL(`*endEffectorSetting*`)`

ReleaseDoor	SetEndEffector(...) $\circ$
	SetEndEffectorParameter(...)
		`let` *endEffectorMode* `= GetGripperAttachMode(Gripper)`
		`BuildSetEndEffectorParameterCRCL(`*endEffectorMode*`)`
		`let` *endEffectorSetting* `= GetGripperReleaseSetting(Gripper)`
		`BuildSetEndEffectorCRCL(`*endEffectorSetting*`)`

Table: Functor G that maps objects and arrows in $\mathbb{C}_2$ to those in $\mathbb{C}_3$

Full Model



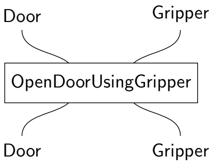
(17)

$\in C_1$ $\in C_2$ $\in C_3$

Full example of string diagram for the robot program: open door using gripper

String Diagrams for Resource Tracking

String diagrams are useful for resource tracking at any "slice" or time resolution



$$(id_d \otimes id_g) \circ p \circ (id_d \otimes id_g)$$

t	expression	shorthand
1	`Door` $\otimes$ `Gripper`	D $\otimes$ G
2	`OpenDoorUsingGripper`	p
3	`Door` $\otimes$ `Gripper`	$\mathrm{id}_d \otimes \mathrm{id}_g$

Table: Linear syntax representation

Benefits of Category Theoretic Representation

This method provides:

- a systematic way to **compartmentalize** the translation from physical objects to programming language syntax, and then, to robot command APIs in your robot program
- an **algebra of functions** that produce rigorous program
- a system for **tracking semantic relationships**

Future Work for Robotic Programming

Explore advantages of symmetric monoidal category structure as a representation for robot programs

- Metrics that characterize the **level of interoperability** or **similarity** between robot program definitions via pattern matching on linear expressions
- Validation model for **detecting failure** during execution time, leveraging
  - Precise expression of temporal resource utilization
  - Strictly functional paradigm
- Exploit sliding and concatenating of string diagrams to produce **optimized process plans** for single and multi-agent workcells

Resources for Applied Category Theory

Tool Support

- Catlab (applied category theory computer algebra library)
- idris-ct (verified category theory library)

Blogs

- GraphicalLinearAlgebra (Blog about string diagram algebra. Very gentle and casual expositions.)
- John Baez (Physics-focused applications of category theory. Math heavy but includes motivational text.)
- Bartosz Milewski (Introduction to Haskell through category theory)

Texts

- Eilenberg, S., & MacLane, S. (1945). General Theory of Natural Equivalences. Transactions of the American Mathematical Society, 58(2), 231-294. doi:10.2307/1990284
- Lawvere, F., & Schanuel, S. (2009). Conceptual Mathematics: A First Introduction to Categories (2nd ed.). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511804199
- Fong, B., Spivak, D.I. (2018). Seven Sketches in Compositionality: An Invitation to Applied Category Theory. arXiv: Category Theory. https://arxiv.org/pdf/1803.05316.pdf
- Eugenia Cheng. How to Bake Pi: An Edible Exploration of the Mathematics of Mathematics. 2015. Basic Books.

Acknowledgements

- Spencer Breiner (NIST)
- Eswaran Subrahmanian (NIST)
- Fred Proctor (NIST)
- Patrick Eisen (Siemens)
- Christof Budnik (Siemens)
- John Nolan (UMD)
- John Kanu (UMD)
- Mukulika Ghosh (UMD)

This work was funded by Advanced Robotics for Manufacturing (ARM).

ARM
ADVANCED ROBOTICS FOR MANUFACTURING

Feel free to reach out to me with questions or interest.

Angeline Aguinaldo
aaguinal@cs.umd.edu