Contents

1 Introduction [draft]

to write

Contents

XXX

1.1 Introduction

1.1.1 A brief history of *Homo sapiens*

The word *intelligence*, from latin *intellego*, comes from Proto-Indo-European *$*h_1$entér* ("between") and **leǵ* ("to gather"), and we can translate it as *the ability to gather things together* to obtain some goal; this, for us, is the essence of intelligence. These *things* to gather could be abstract, such as pieces of evidence to achieve a conclusion, or physical, like ingredients to prepare a tasty meal, or the parts to create machines.

Intelligence is not unique to *homo sapiens*; other animals can reason, build, and use language. Some animals excel at things that *sapiens* cannot do. The sapienses will never experience the richness of the smells of things and processes in the world like *canis* does. The sapienses will never have the same spatial awareness of an octopus with eight arm-legs and neurons distributed all along its body. But sapienses developed a trick no other animal did. They mastered **abstraction and compositionality**.

Societies Sapiens de-composed the process of survival. Once, a single sapiens had the ability to survive by themselves, or in a very small pack of sapiens. Eventually they figured out that it was much more efficient to divide up the work, so that some could specialize in hunting, some in gathering, some in fighting, some in rearing children.

About 10,000 years ago, sapiens invented **agriculture**; it was a momentous change, as it was the first time that they could change the world around them and bend nature to their will. Up to that moment, it was the other way around: as they moved beyond Africa, sapienses adapted *to* the environment; bodies optimized to run after prey in savannahs became optimized to fish in tropical seas or to heard cattle on the Alps.

More specialization. Today only 1% work at food production. In fact, you could take most sapiens and put them in the most fruit-rich plains, the most prey-rich savannah, and, alone, they would die in days.

Writing With a sedentary society, while the grains and the rice grew by themselves in the field, they found the time to invent writing. They managed to de-compose thought into a sequence of symbols, which could be written on clay, and re-composed back by the receiver to reconstruct the original thought. Writing is a teleportation device; previously you had to be face-to-face, or at least at a hear's distance, to communicate. Writing is also a time machine:

Money abstraction of resources

Industrial de-composition becomes a science

Machines Repetitions

Thinking machines Lately, *homo sapiens* has cultivated and inclination to create machines that could help them *think*. Firstly, it was about counting Then, it was about **remembering** Then, it was about **communication** Then, it was about **coordination**

Bunch of different problems

and we need to tell them how to do this intelligence trick, not just solve single problems, but see how solutions to smaller problems can be composed to solve bigger problems. Retorics

Machines making machines

1.1.2 Compositional and computational knowledge

Knowledge must be compositional and computational.

1.2 Compositional thinking for engineers

The thesis of this book is that most engineering fields would benefit from knowing and using the language of applied category theory to address the design and analysis of complex systems.

1.2.1 Systems and components

What is a "system"?

Here is a great quote[1]:

> A system is composed of components;
> a component is something you understand.

The first part of the quote, "A system is *composed* of *components*", is plain as day as much as it is tautological. We could equally say: "A system is *partitioned* in *parts*".

The second part, "a component is something you understand", is where the insight lies: we call "system" what is too complex to be understood naturally by a human.

Aiken referred to computer engineering, but we find exactly the same sentiment expressed in other fields. In systems engineering, Leveson puts it as "complexity can be defined as intellectual unmanageability" [**leveson12engineering**].

We will be content of this anthropocentric and slightly circular definition of systems and complexity: "systems" are "complex" and "components" are "simple".

Whether something is a complex system also depends on the task that we need to do with it. One way to visualize this is to imagine a "phenomenon" as a high-dimensional object that we can see from different angles (Fig. 1.1). For each task, we have a different projection. The decomposition of the system in components can be different according to the task. For example, a system that might be easy to simulate could be very difficult to control.

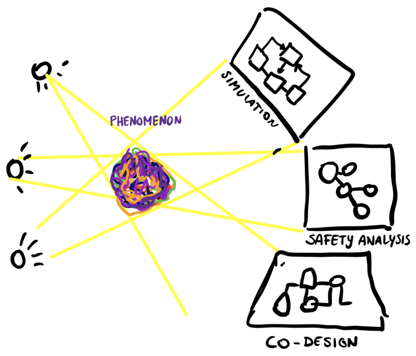

Figure 1.1: Engineers live in a Plato's cave with multiple light sources. A certain phenomenon can be illuminated from different angles and look very different, very simple or complex.

Make better figure

1.2.2 What ACT can do for you

Many of the topics we will discuss here can be catagorized under the umbrella of applied category theory.

describe how ACT can help

[1]This quote is by Howard Aiken (1900-1973), creator of the MARK I computer, as quoted by Kenneth E. Iverson (1920-2004), creator of programming language APL, as quoted in [**McIntyre1999Role**], but ultimately sourceless and probably apocryphal.

1.2.3 What ACT cannot do

This is a way of thinking about thinking

describe limitations of ACT

1.3 Guidebook

1.3.1 Atlas of abstractions

Broader overview of the abstractions that we will develop in this book, including posets, lattices, etc. up to operad.

1.3.2 Book organization

To write.

1.4 Acks

1.4.1 Contributors

To write.

1.4.2 Sponsors

To write.

2 Workbook [draft]

Contents

XXX

2.1 hello

Exercise 1 (`TestFiniteSetOperations`). Given two finite sets, compute the union and intersection.

```
class FiniteSetOperations(ABC):
    @abstractmethod
    def union(self, s1: FiniteSet, s2: FiniteSet) -> FiniteSet:
        ...
    @abstractmethod
    def intersection(self, s1: FiniteSet, s2: FiniteSet) -> FiniteSet:
        ...
```