Concurrency and Topology

Brad McCoy and Gage O'Neill

CSCI 460 Project

December 12, 2019

1 Introduction

In computation, multiple processes often share resources. This is called concurrent programming. In some cases executions of concurrent programs can lead to deadlock. Deadlock occurs when one process awaits a resource held by another waiting process, resulting in the program never terminating. Recall the conditions for deadlock:

- Hold and wait mechanisms
- Circular wait queue
- Mutual exclusion
- No preemption allowed

Consider two processes sharing two resources; we can plot both processes on an axis and mark when each process acquires and releases a resource. This gives a geometric region that will lead to deadlock, known as the fatal region. See the figure below:

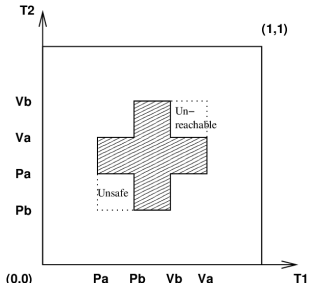

Figure 1: Swiss Flag

Verifying a concurrent program involves running all possible inputs to determine if a given input will not result in deadlock. Often there are an infinite number of possible inputs making verification impossible.
A possible solution to this problem uses tools from Algebraic Topology. Instead of attempting to run an infinite number of tests we define an equivalence on sets of inputs.

Then the authors prove that if one representative of the equivalence class does not result in deadlock then no other member of the class results in deadlock. This implies that if one checks a single member from each equivalence class then one has determined which inputs do not result in deadlock - highly useful when investigating concurrent operations.
The goal of the document is to give non-topologist a feeling that directed homotopy is useful in studying concurrency. It is also a glimpse at how advanced topics in geometry can be applied to concurrent programming. The layout of this paper is as follows: in section 2 we give a introduction to homotopy and the definition of directed homotopy, and in section 3 we explain how these definitions are useful in giving a description of the fatal region. Additionally, we revisit the Dining Philosophers Problem in section 4 and finally end with a discussion.

2 History of Homotopy

In this section, we introduce many definitions that enable us to define an equivalence of paths. The most important definition is that of a topological space. For our purposes one can think of a topological space as a rubber blob that can stretch and bend. A fundamental question in the field of topology is to determine if one can deform one space into another. Determining which spaces are the same is often counter intuitive. See the figure below for a nice example of the two torus:
A useful strategy to determine if two spaces are different is by showing they belong to different homotopy classes. A *homotopy class* is an equivalence relation on topological spaces, where the equivalence is path homotopies. A path *homotopy* between two paths $f(x)$ and $g(x)$ from x_0 to x_1 in X is a continuous function from the closed two dimensional interval into any topological space, X with four properties:

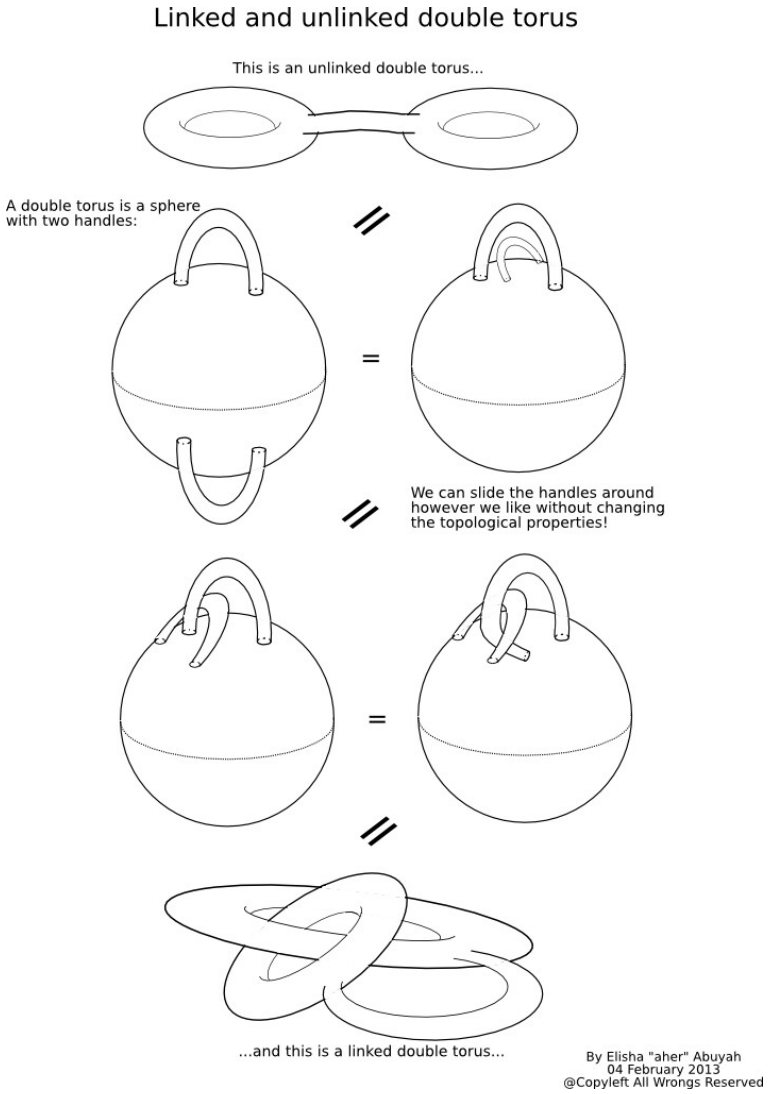

Figure 2: Two torus from [1]

$$H : I \times I \to X$$

$$H(x,0) = f(x), \qquad H(x,1) = g(x)$$
$$H(0,\cdot) = x_0, \qquad H(1,\cdot) = x_1.$$

It is fascinating that this tool is finding modern applications in concurrent programming.

3 Homotopy and Concurrency

Here we show how the set of all possible inputs into a concurrent program can be described as a topological space.
For concurrent processes time is always increasing for both processes so here our homotopies include the property that all paths must be increasing in all directions. We call these *dipaths*. The space of all dipaths together with the compact open topology gives us a topological space that represents the set of all possible executions of a concurrent program. The main payoff of studying the topology of the space of directed paths is that for any two paths, f and g, in the cubical complex have a homotopy between them. Then, if f does not result in deadlock then neither does g for the entire trace of the homotopy does not result in deadlock. The implication is that we can not check a single path from each homotopy class. Instead of the impossible task of checking an infinite number of inputs we can verify a program with a finite number of inputs.
Consider the swiss flag example introduced in Section 1. All paths that travel above the plus sign are homotopic and all paths that travel below the plus sign are homotopic. Thus, for this configuration of processes verification requires checking two particular inputs. This simplification is quite involved and is the discussion of [3].
To give some intuition towards their argument, the space is decomposed into smaller cells called cubical complexes. By examining the boundary of these cells one can determine if that cell can be collapsed without changing the connectivity of the space. This is done by using the past link of a vertex with respect to another vertex to keep track of connectivity. After performing all possible collapses it is possible to compute the homotopy type of the space.
The authors of [3] also provide an alternative approach using what are called cubical complexes, but argue that directed collapsing is stronger.

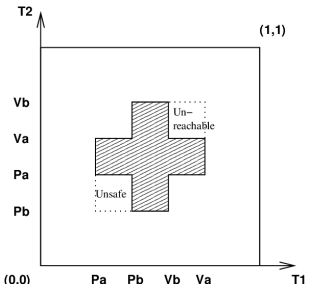

Figure 3: Swiss Flag

In practice, the number of processes sharing resources is more than two. For an example of three processes see the figure below:

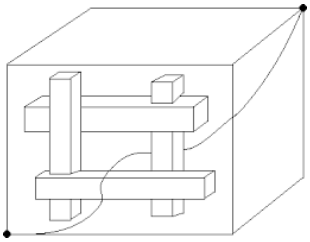

Figure 4: 3-D

The topology extends with minimal effort to higher dimension, as the definition of the space of dipaths does not depend on dimension.

4 Another Look at the Dining Philosophers Problem

Recall the example introduced in Section 1, illustrating two processes sharing two resources or a simple version of the classic Dining Philosophers problem. To further illustrate what execution paths and fatal regions can look like in higher dimensions, Figure 5 was generated using Python's NumPy library, seen below:

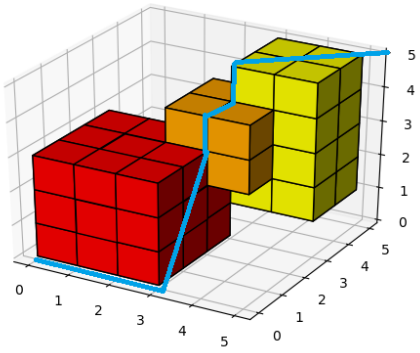

Figure 5: 3 Philosophers and 3 Resources

Here, we see the fatal region represented as the *euclidean cubical complex*, or a union of cubes, in red, orange, and yellow - each color representing a distinct resource. In three dimensions, there are three axes representing each philosopher over time as they make use of each resource. This figure is meant to help visualize how the fatal region can grow significantly by adding just one additional process and resource.

5 Discussion

Verifying concurrent programs offers reassurance that processes that share resources can coexist. This reassurance is achieved by using tools that were developed to answer

theoretical questions in mathematics.
Interestingly, none of the resources that we found gave a specific application of how verification has been used in a "real world" application. Even overviews of the topic such as [2] do not go into detail of how this is useful. It would be nice to see how the idea of verification is being used in practice.
Nevertheless, more research is being done in this area. In [3] the authors are curious as to how to perform the collapsing of cubical complexes more efficiently among other things. The ultimate goal could be to be able to analyze more complex concurrent programs using topology and homotopy to verify that there exists successful modes of execution - significantly reducing the work that goes into verifying a program.

References

[1] Elisha Abuyah. Image of linked 2-torus, 2013. https://www.flickr.com/photos/79364035@N04/8444518698.

[2] Eric Goubault Lisbeth Fajstrup, Martin Rauben. Algebraic topology and concurrency, 2006. Theoretical Computer Science, pages 241-271.

[3] Stefania Ebli Lisbeth Fajstrup Brittany Fasy Catherine Ray Nicole Sanderson Elizabeth Vidaurre Robin Belton, Robyn Brooks. Towards directed collapsibility, 2019. https://arxiv.org/pdf/1902.01039.pdf.