COASTL: Creating a Software Package for Signal Temporal Logic Processing and Design-by-Contract Operations

Pratham Gandhi, gandhip@horacemann.org
Horace Mann School, Class of 2020
University of Southern California, Department of Electrical Engineering

Introduction

Currently, much of system design is based around trial and error tweaking techniques, where a system is put through a series of tests and adjustments in the design are made according to test results. This, however, is extremely costly and time-consuming. A solution to improve the design process via a rigorous design discipline can increase productivity and guarantee correctness. Contract-based design is a system design approach which provides both horizontal and vertical integration capabilities. This allows us to be able to decompose and compose requirements for sub-components in the early stages of design, and tell if a design is flawed or not without spending resources on development. This is better than iterative testing, which only reveals faults after a prototype has been developed. As stated earlier, the capability to decompose and compose requirements is key to the superiority of contract-based design, and thus, a toolkit which effectively achieves these operations needs to be created before this scheme can be widely implemented.

Background

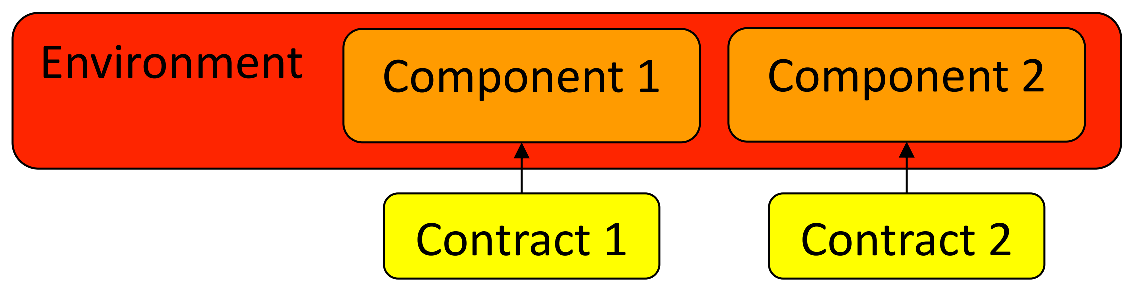

Figure 1: Environment-Component-Contract Structure

Contracts are descriptions of components which are intentionally abstract and serve the purpose of distinguishing the responsibilities of a component from that of its environment.

Signal temporal logic (STL) is a modal temporal logic by which you can formally specify the responsibilities of components. Unlike linear temporal logic (LTL) which can only reason about binary variables, STL allows you to reason about continuous variables. The modality aspect of STL comes in the fact that modifiers of STL expressions allow them to reason about a variable at different times (timesteps). For example, x must be greater than or equal to 3 eventually between time 0 and 10, "F[0,10]($3 \leq x$)", y must be less than or equal to 1 for all times 10 to 20, "G[10,20]($y \leq 1$)."

Approach

STL Parsing

Functionality to parse *Signal Temporal Logic* expressions such as "G[0,10]((x<=5)&&((y<=7)||(~(10<=y))))" into a *tree structure* is essential. It is implemented by finding unique expressions encased in parentheses, and constructing nodes, as shown in Fig. 2.

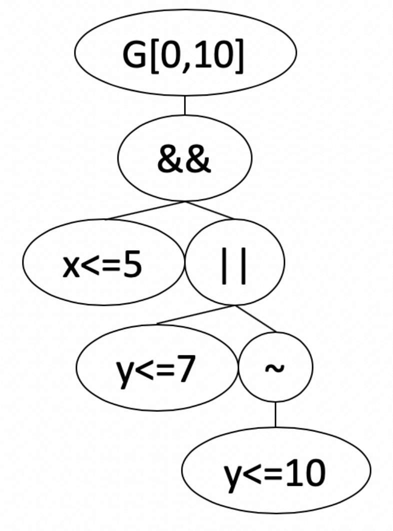

Figure 2: STL Tree

Constraints & Solution

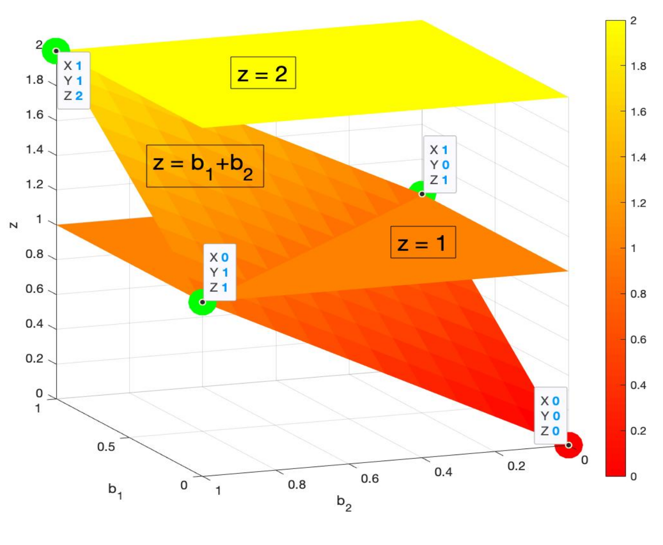

Figure 3: "OR" Constraints

For every logical operator, a set of Boolean *constraints* are imposed in order to enforce the correctness of the logic for any solution. An O(n) STL tree traversal produces constraints. These constraints become part of an optimization problem which leads to solutions for every binary variable. The expression "$b_0 = b_1 \cup b_2$" produces constraints $b_0 == 1$, $b_0 \leq b_1 + b\_2$, and $b_1 + b_2 \leq 2 * b_0$, visualized in Fig. 3.

At Atomic Predicate (AP) nodes, we impose two additional constraints which allow the synthesis of values for the continuous variables reasoned about at that node. For the AP expression $x \leq 3$, whose truth is represented by b, the constraints $(b-1) * M \leq 3 - x$ (Fig. 4.1) and $3 - x \leq b * M - \epsilon$ (Fig. 4.2) are imposed. M, ϵ are maximum, minimum bounds for the optimization problem (default to $10^4, 10^{-4}$). Fig. 4 shows possible solutions (Fig. 4.3) given all combinations of binary and continuous values of relevant variables. The larger optimization problem with every constraint is solved using Gurobi, a linear optimization solver.

Contract Operations

Given two contracts $C1 = [A_1, G_1]$ and $C2 = [A_2, G_2]$, various operations can be carried out in order to combine the requirements in meaningful ways.

Approach (cont.)

Saturation of a contract is necessary before performing any operations:

$$C_1' = [A_1, G_1 \cup (\neg A_1)] = [A_1', G_1']$$

Conjunction of contracts to join requirements for one systems:

$$C_{conj} = [A_1' \cup A_2', G_1' \cap G_2']$$

Composition of contracts to join requirements for multiple systems in one environment:

$$C_{comp} = [(A_1' \cap A_2') \cup \neg(G_1' \cap G_2'), G_1' \cap G_2']$$

Examples

Simple Numerical Synthesis

Consider the following contracts:

```
c1 = contract('T', '1<=x<=3')
c2 = contract('x<=5', '2<=x+y<=10')
```

To find a solution where all the assumptions are met and the solution is guaranteed by *c1* and *c2*, we can perform the conjunction operation ($c1 \wedge c2$), saturate the resulting contract and solve the guarantees.

```
(conjunction(c1,c2).synthesize()).get_synthesized_vars()
>> 1,0
```

Autonomous Vehicle Control

Consider the example of synthesizing a control system for an autonomous farming vehicle. The vehicle must navigate two rows of crops and then return to its starting position along a predetermined route in a 20 minute (equivalent to 20 timestep) cycle. We create the following contracts:

```
c1 = ('T','F[0,3]((x==55)&&(y==27))')
c2 = ('T','F[4,6]((x(1)==5)&&(x(2)==13))')
c3 = ('T','F[7,10]((x(1)==55)&&(x(2)==5))')
c4 = ('T','F[11,15]((x(1)==55)&&(x(2)==35))')
c5 = ('T','F[16,20]((x(1)==5)&&(x(2)==35))')
c6 = ('T',
'G[0,20](~((10<=x(1)<=50)&&(26<=x(2)<=28)))')
c7 = ('T',
'G[0,20](~((10<=x(1)<=50)&&(12<=x(2)<=14)))')
```

The vehicle's maximum velocity and acceleration are constrained to be 0.5 m/s and and 1.67 m/s^2, respectively. Solving the optimization problem produced by guarantees from the saturated contract created by `conjunction(c1,c2,c3,c4,c5,c6,c7)` produces the results shown in Fig. 5.

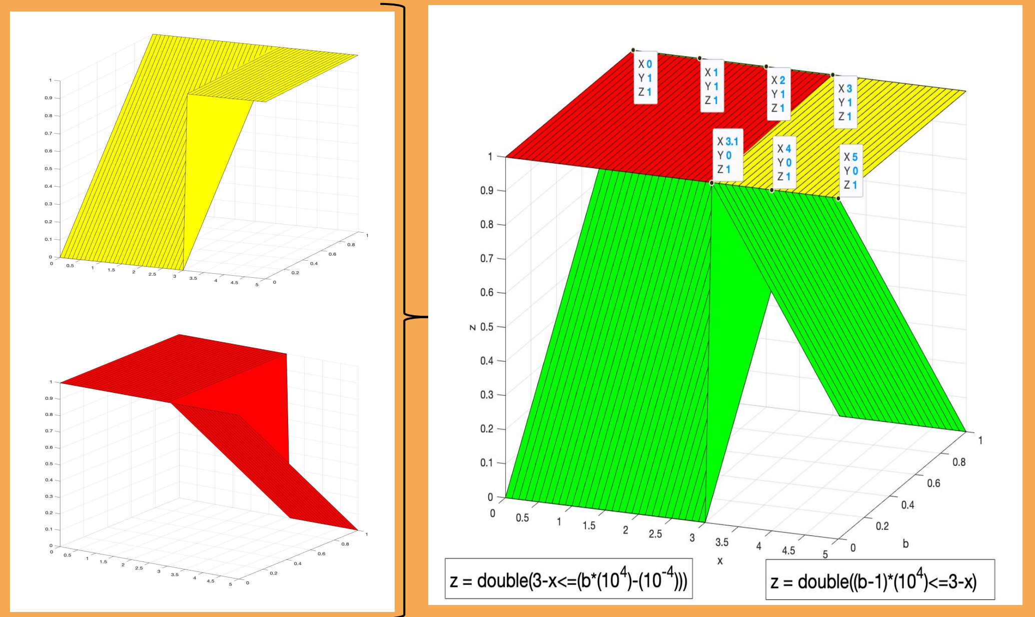

Figure 4: AP Constraints; Top Left (4.1), Bottom Left (4.2), Right (4.3)

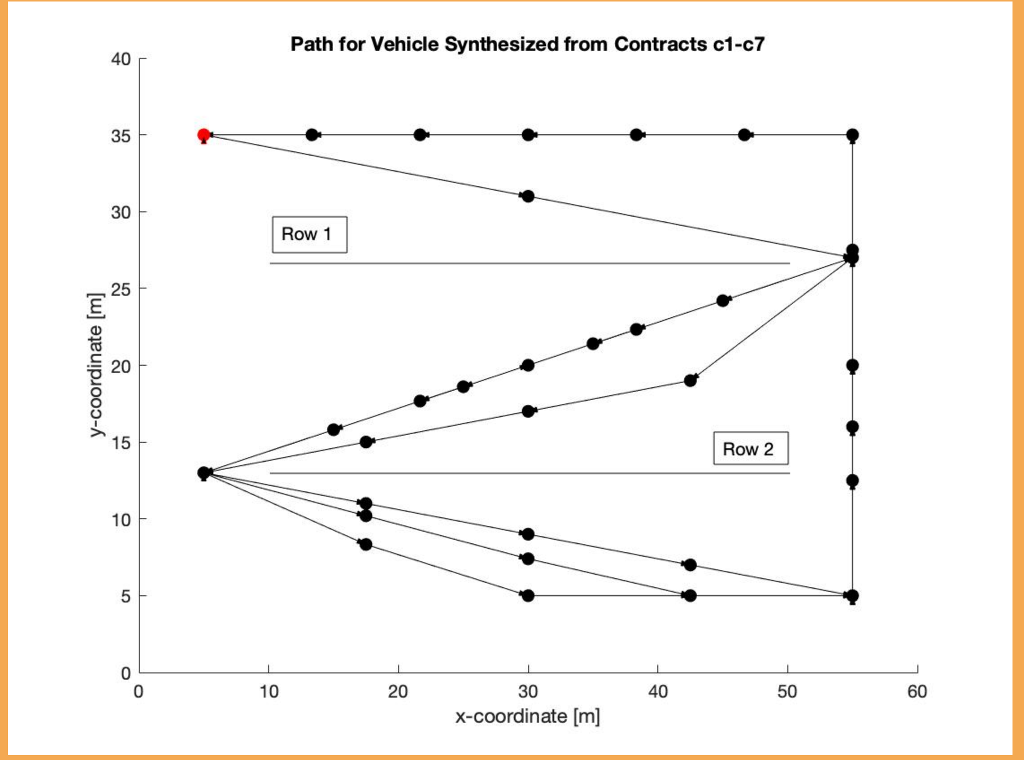

Figure 5: Autonomous Vehicle Control from Contracts

Conclusion

"coastl" is a Python tool which computes operations on contracts for design-by-contract system design schemes. The tool takes a unique but effective approach in parsing STL expressions of requirements into a tree structure, making operations simpler and faster down the line. The trees are easily manipulated to execute contract-level operations. Constraints are derived from the trees, and values for the continuous variables considered in the logical expressions are synthesized.

Acknowledgements

Thank you to Chanwook Oh and Prof. Pierluigi Nuzzo for their guidance and hands-on help in formulating and carrying out this project. Additional thanks to Dr. Megan Harrold and Dr. Katie Mills for their mentorship throughout.