COASTL: A Software Package for Contract Operations and Signal Temporal Logic Processing

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

USC Viterbi School of Engineering
Ming Hsieh Department of Electrical Engineering

SHINE
Summer High School Intensive in Next-Generation Engineering

Introduction

The design of complex systems is mostly based on costly trial and error techniques, where a system is put through a series of tests and adjustments.

Contract-based design is one of the most promising system design approaches to increase productivity and guarantee correctness:

- Allows principled decompositions and compositions of requirements in the early stages of design enabling identification of design flaws without spending large resources on development stages
- Needs a toolkit which effectively computes these operations to enable its wide adoption

This project develops a Python package which allows users to input contracts and perform operations with them. The package will output the resulting contract as well as be able to generate system behaviors that satisfy the contract. Contract requirements will be specified by strings of logic input by the user.

Background

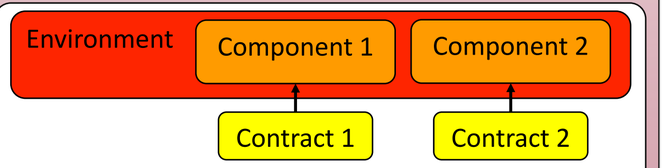

Figure 1: Environment-Component-Contract Structure

- A **Contract** is a rigorous description of a component's requirements written in formal logics and can distinguish the responsibilities of a component from that of its environment using **Assumptions** and **Guarantees**.
- **Signal temporal logic (STL)** is a timed logic which can formally specify the requirements of components in terms of time.
- Unlike linear temporal logic (LTL) which can only reason about binary variables, STL allows you to reason about continuous variables.
- ex) *x* greater than or equal to 3 eventually between time 0 and 10 — "F[0,10](3 ≤ x)"

Approach

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.

- *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:

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

STL Parsing

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

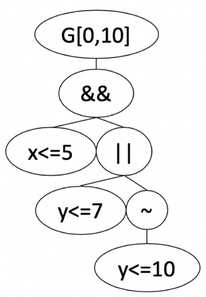

Figure 2: STL Tree

Translating Contracts into Mixed Integer Constraints

- Boolean *constraints* enforce correctness of logical operators.
- STL tree traversal produces constraints which become part of an optimization problem
- Solving optimization problem provides solutions for every binary variable.

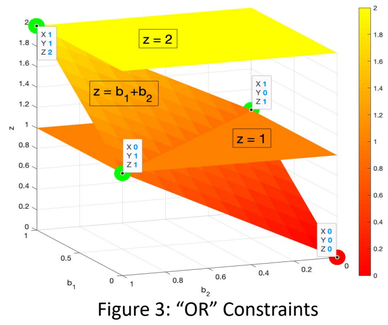

Figure 3: "OR" Constraints

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.

- For Atomic Predicate (AP) expression $x \leq 3$, truth is represented by b, constraints $(b-1) * M \leq 3 - x$ and $3 - x \leq b * M - \epsilon$ are imposed to find continuous x
- M, ϵ are maximum, minimum bounds for optimization problem (default to $10^4, 10^{-4}$)

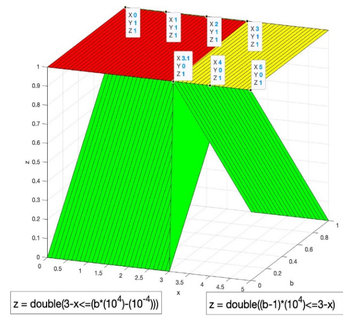

Figure 4: AP Constraints

- Fig. 4 shows possible solutions for some different combinations of binary and continuous values of relevant variables. The larger optimization problem with every constraint is solved using Gurobi, a linear optimization solver.

Examples

Simple Contract Operations

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

Perform conjunction ($c1 \wedge c2$), saturate the resulting contract, and solve the guarantees to find a solution guaranteed by *c1* and *c2:*

```
(conjunction(c1,c2).synthesize()).get_vars()
>> ['x': 1,'y': 9]
```

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)))')
```

Examples (cont.)

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.

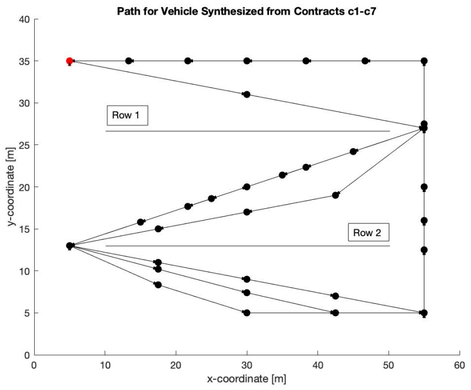

Figure 5: Autonomous Vehicle Control from Contracts

Conclusion

COASTL is a Python tool which computes operations for design-by-contract system design:

- Parses STL expressions of requirements into a tree structure, making the computation of contract operations simpler and faster
- Facilitates translation of STL contracts into mixed integer constraints and automatic generation of system behaviors that satisfy the contracts

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.