Reactive Synthesis 2024

Project Description

The goal of this project is to provide hands-on experience with formalizing requirements as specifications in LTL, and using state-of-the-art reactive synthesis tools to synthesize implementations from these specifications. To this end, you are given a high-level description of the desired behavior of a system, i.e., a *controller*, and the *environment* in which it operates. Your task is, based on that description, to formalize the *requirements for the controller* and the *assumptions on the environment* in the form of a temporal logic specification, and to use a synthesis tool to find an implementation for the controller. Furthermore, you are required to document your reasoning and the justification of choices you make during the design of the formal specification, and your experience with using the synthesis tool.

1 Scenario Description

Urban air mobility needs systems in place to coordinate the traffic of unmanned aerial vehicles (UAV). We divide the air space into cells, which we call *controlled traffic regions (CTR)*. Each CTR can be occupied by at most one UAV at any given time. On the ground, *helipads* are available for the UAVs to land on. The helipads are serviced by *ground robots* for charging the UAVs and for loading and unloading cargo on/off the UAVs.
There are three types of UAV: *emergency vehicle*, *passenger transport* and *cargo transport*. These are assigned priorities, where emergency vehicles have the highest priority and cargo transport the lowest. Each UAV has an id by which it can be uniquely identified.

Controlled Traffic Regions

Each *CTR has a controller* to regulate air traffic inside the zone. UAVs request entry into the CTR, and only one vehicle is allowed to be into the zone at any time. The controller grants permission to enter based on the previously mentioned priority. Should an emergency vehicle enter a CTR, a warning is issued to controllers of neighboring CTRs. A UAV can leave a CTR by moving to a neighboring CTR or by landing on a helipad.

Helipads

UAVs can land upon helipads, each offering space for exactly one UAV. Before landing on an empty helipad the UAV has to request a permission to land. The *helipad controller* has to grant the landing permission according to the priority. Before granting landing permissions, the helipad controller has to ensure that the helipad is free from ground robots and persons. Once landed, a UAV can remain on the helipad for an arbitrary amount of time. While the UAV is on the helipad, the helipad controller is in charge of requesting the service ground robots if necessary.

Charging Robots

A charging robot recharges UAVs on helipads upon request. Once a charging robot goes to a helipad and starts charging the UAV, it has to charge it until it is fully recharged. A single charging robot is responsible for servicing 4 helipads, and can service at most one of them at any given time.

Cargo Robots

A cargo robot transports, loads and unloads cargo. Cargo can be found in UAVs that have landed on helipads, and in the warehouse. Upon request by a helipad controller, the cargo robot should go to the helipad, unloads the cargo from the UAV, go to the warehouse and put the cargo there. Similarly, upon request, a cargo robot takes cargo from the warehouse, transports it to the helipad, and loads it into the UAV. A cargo robot can only transport a single piece of cargo at the time, i.e. if it is already transporting cargo it has to unload it until reloading cargo again. A single cargo robot is responsible for servicing 4 helipads.

Overall

The traffic management system being designed then consists of the controllers for

1. Charging Robots,
2. Cargo Robots,
3. Helipads (which grant landing permissions and ensures safe landing), and
4. CTRs (which regulate traffic inside a zone).

The UAVs are part of the external environment. Consider a scenario with **at least 1 emergency vehicle, 2 passenger vehicle and 2 cargo transport vehicle**.

2 Project Task Description

Your task is to provide temporal logic specifications for the controllers, reflecting the high-level description given above. Formulate the specification of each of the controllers individually. Remember to document your process
Investigate first how these controllers could interact, what inputs they depend on and what outputs they produce. Make sure not to over-specify the desired behavior of the controllers, i.e. do not describe a specific concrete implementation via the specification. Instead, specify *what the controller should do* not *how it does it.*

Important: Documentation of your process is mandatory, see Task 5 for more information on that. The controller specifications and synthesized implementations should be in the formats listed in Section 3. Please see Section 4 for all further submission information.

Task 1: Charging Robot

Complete the specification of the Charging Robot controller, from the unfinished specification file `charging_robot_sample.tlsf`.
The environment issues service requests to a specific helipad via the input `call_from`, and provides information about the battery status with `charging_complete`. The controller decides the robot's movement and when to begin and stop charging. Where the robot moves to is modeled by the output `service_location`, and the output `deliver_power` controls the charging process. Note that the robot controller should be able to handle requests from four different helipads, so `call_from` and `service_location` are parameterized with $n = 4$.
From the description given, we can identify these requirements:

- if charging is requested, the robot will handle the request
- once the robot begins servicing a helipad, it has to complete the charging

- the robot can service at most one helipad at any given time

Your task is to translate these into a formal specification to synthesize a correct controller. Consider if assumptions on its environment may be necessary to obtain a realizable specification.

Task 2: Cargo Robot

Write a specification for the Cargo Robot controller from scratch, according to its high-level description. You can make use of ideas from the previous task, but be mindful of the differences in the requirements.

Task 3: Helipad

Write a specification for the Helipad controller. The controller handles permissions for UAVs to land based on their priority. It also interacts with the service robots, issuing requests and communicating the status of a landed vehicle.

Task 4: CTR Controller

Write a specification for the CTR controller, which grants and revokes entry permissions to UAVs according to the stated priority.

Task 5: Project Report

While developing the specifications and synthesizing the controllers from these specifications, you are expected to document and explain your process in getting there. Write a short document (approx. 2 pages, PDF or in plain text) to describe:

- the choices you made in modeling the controllers,
- which assumptions you introduced and why,
- the time the synthesis tool took for synthesizing each of the controllers,
- any problems you faced, and any shortcomings of your submission.

(BONUS) Task 6: System Extension

Once you are satisfied with your solution to the above, you can try to accomplish the following bonus task: Think of an interesting and meaningful extension either to one of the four controllers or add another component to the combined traffic management system. Define its relation to the existing functionality, and describe its intended behavior. Argue why this extension makes sense. Then, write a formal specification, synthesize the controller and document your process as before. The bonus task is not necessary for passing the project, but may earn you a bonus (see Section 4).

Helpful Comments

- When writing a specification for a controller, start by determining what are the inputs provided by the environment in which this controller operates, and what are the outputs of the controller.
- One challenge when writing specifications is to avoid contradictions that make the specification unrealizable. Hence, it can be helpful to check the realizability of the specification during the course of extending it, in order to find contradictions early on.

- As reactive synthesis is a hard problem, even state-of-the-art tools suffer from scalability issues and might take some time to produce results. It is therefore important that you are careful when stating requirements and assumptions, in order to avoid over-specification.
- In the course of designing a realizable specification, especially if you encounter scalability issues, it might be helpful to temporally reduce the number of UAVs in the environment of the system.

3 Tools and File Formats

Write your specifications in the TLSF format[1]. TLSF specifcation are semantically LTL but included a lot of syntactic sugar allowing more structured, concise specifications with better readability. We provide an example specification file in TLSF in the CMS.
For doing synthesis recommend two options:

Install the tools yourself Get and build SyFCo[2] which allows to transform TSLF specifaction into different formats (e.g. bare-bones LTL). Then get the latest release of Strix[3], which is a synthesis tool for LTL specifications. It can be used to determine the realizability of a specification, and if possible, generate a Mealy Machine or AIGER circuit. You can determine how to use Strix by calling `strix -h`. To use Strix with TLSF specifications and SyFCo, you can use an existing wrapper script [4].

Use docker Alternativley, we provide a Dockerfile that wraps the installation and usage of the tools above. You can find a ZIP file in the CMS that contains the Dockerfile and some wrapper scripts. To build the Dockerimage, run

```
docker build -t reactive-synthesis-lecture-2024 .
```

in the folder where the `Dockerfile` is. Inside the build Dockerimage the is a script called `synthesize` in the top level folder which takes the same arguments as `strix`, reads a TLSF file from STDIN and outputs the synthesis result on STDOUT. In order to make the the usage easier we provide two wrapper scripts. Run

```
run-docker-real.sh <YOUR-TLSF-FILE>
```

to check if a specification is realizable. If it is you can run

```
run-docker.sh <YOUR-TLSF-FILE>
```

to generate the AIGER circuit.

4 Submission Instructions and Evaluation

You can work on the project alone or in a team of two. In either case, make a (1 or 2-person) team on your personal status page in CMS by **September 24, 2024**. You must submit your solution by **September 27, 2024 at 17:59** on the personal status page in the CMS. A submission has to contain the following:

- The specifications of the four controllers in the TLSF format.

[1] `https://arxiv.org/pdf/1604.02284.pdf`
[2] `https://github.com/reactive-systems/syfco`
[3] `https://strix.model.in.tum.de/`
[4] `https://github.com/meyerphi/strix/blob/main/scripts/strix_tlsf.sh`

- The controllers synthesized from your specifications in AIGER format.
- A document describing your process and reasoning, justifying the choices you made. This document must also include the names and matriculation numbers of all team members.

Please only submit one solution per team.

To pass the project submission, your submission must provide a solution to the tasks 1-5 as outlined above. If, in addition, you successfully fulfill the requirements of the bonus task 6, you will be awarded one grade step improvement of your exam grade (for instance, from 2.7 to 2.3), provided you have a passing grade in the exam.