CS 333: Safe and Interactive Robotics — Fall 2017

Lecture 5: Reactive Synthesis for LTL

Scribes: Karen, Vikranth

Reactive synthesis for LTL

The idea of reactive synthesis is to synthesis (construct) a strategy for a reactive system that satisfies the given specification given all possible inputs. That is given all possible inputs, we want a strategy f such that the corresponding outputs satisfy the LTL specification. Mathematically speaking, this can be written as $f(2^I)^* = 2^O$ and a winning strategy is one where the output behavior satisfies the specification.

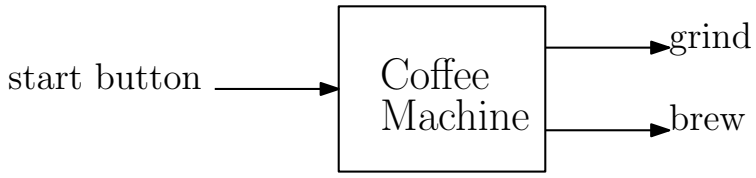
There are different notions of how you may satisfy a specification.

Definition 1 (Satisfiability) *Satisfiability means that there exists an input sequence and output sequence such that the specification is satisfied. Mathematically, this can can be written as:* $\exists \hat{i} = i_0, i_1, \ldots$ *and* $\exists \hat{o} = o_0, o_1, \ldots$ *s.t.* $\rho = \hat{i} \cup \hat{o} \models \phi$

Definition 2 (Realizability) *Realizability means that for all input sequence, there exists an output sequence such that the specification is satisfied. Mathematically, this can can be written as:* $\forall \hat{i} = i_0, i_1, \ldots,$ $\exists \hat{o} = o_0, o_1, \ldots$ *s.t.* $\rho = \hat{i} \cup \hat{o} \models \phi$

The Coffee Machine Example

Suppose you have a coffee machine and the specification is the following: If you press the start button, then the system will immediately start grinding for the next two cycles and then it will brew for the next two cycles after. Then the coffee will be produced. For this specification, our input is $I = \{start\,button\}$ and the outputs are $O = \{grind, brew\}$. Our reactive system is illustrated below.



In terms of LTL notation, this can be written as

$$\varphi = G\,(button \Rightarrow Grind \wedge XGrind \wedge XX(\neg Grind \wedge Brew) \wedge XXX(\neg Grind \wedge Brew))$$

Suppose we press the start button once, then we will have the following sequence of inputs/outputs:

$$\begin{pmatrix} button \\ grind \\ brew \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

φ is satisfied by this sequence of inputs/outputs.

Suppose now that we accidentally press the start button twice

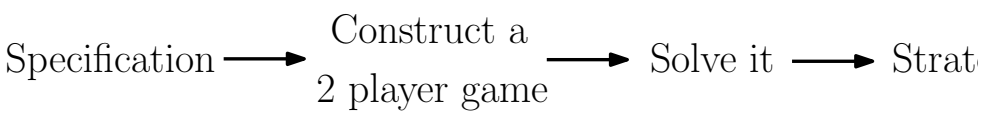
$$\begin{pmatrix} button \\ grind \\ brew \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ ? \\ ? \end{pmatrix}$$

It is not clear what the outputs should be and for any output, we do not satisfy the specification given the input. Hence, φ is not realizable.

Thus this is an example of a satisfiable system since there exists an input-output sequence where the specification is met, but this is not a realizable system because the specification is not met for all input sequences.

When a system is not realizable, we obtain a counter strategy which fails and this helps us in improving our specification.

As such, let us try change the specification and approach this problem with a different perspective. We can take the specification and convert it to a state machine.



The difficulty here is transforming the specification into a 2-player game. Here we will describe in detail how this can be done by doing a simple example. In general, the steps of obtaining a 2-player game are:

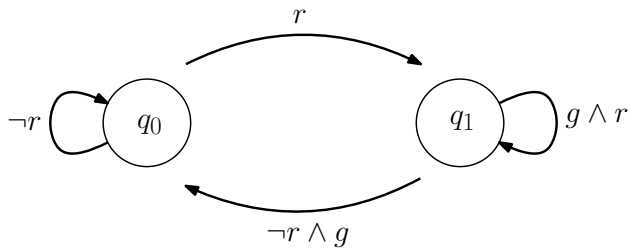
1. Convert the specification to a non-deterministic Büchi automaton (exponential in size of spec)
2. Determinize it (exponential in size of spec)
3. Obtain the game

So it's doubly exponential algorithm in the size of specification.

Example: $\varphi_1 : G(r \rightarrow Xg)$ where:

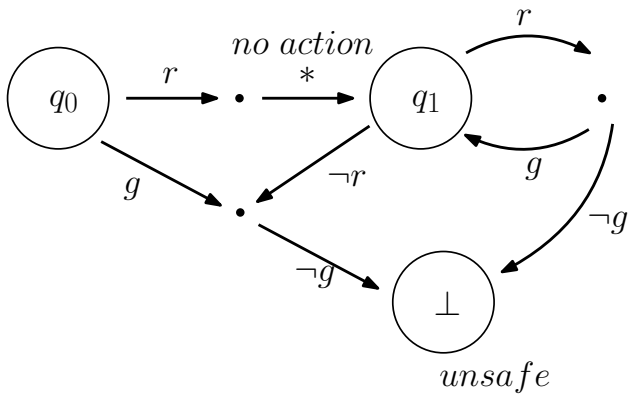
- Input $I = r$, request or no request
- Output $O = g$, granted or not granted

Step 1 Construct a Büchi automaton



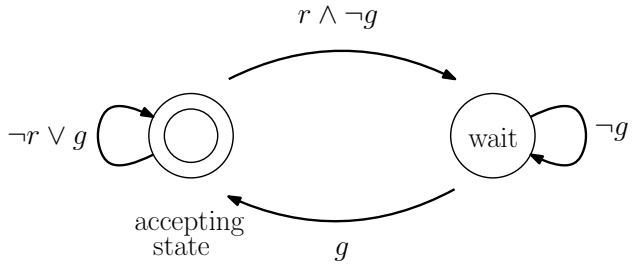
Step 2 It is already deterministic. No further conversion is required. Algorithms like subset construction and safraless method can be used for this.

Step 3 Convert it to a 2-player game. Introduce intermediate states and an unsafe state. Assuming the environment acts first (input), we then reach an intermediate state and then the system acts and transitions to the next state. This is shown in the figure below. This can be viewed as a reachability problem where we do not want to reach the unsafe / failure state.



Here system can control whether it grants a request or not. So, a simple strategy to always avoid the unsafe state is to always grant the request :

Another example is $\varphi_2 : G(r \rightarrow Fg)$ and the Büchi automate is given below.



There are different ways to formulate the 2-player game. Here are three types of games we could use:

Safety — Reachability game

$$\text{Safe states } (S) \subseteq \text{all states } (V)$$

Player 0 wins the play $\pi = p_0 p_1 p_2 ...$ if $p_i \in S \ \forall i$

Büchi game

$$\text{Accepting states } (F) \subseteq \text{ all states}(V)$$

Let $Inf(\pi)$ be the set of all states which are visited infinitely often. Then the condition for winning is $Inf(\pi) \cap F \neq \emptyset$ which means you will visit accepting states infinitely often.

Parity game

Pairs of good and bad states (G,B)

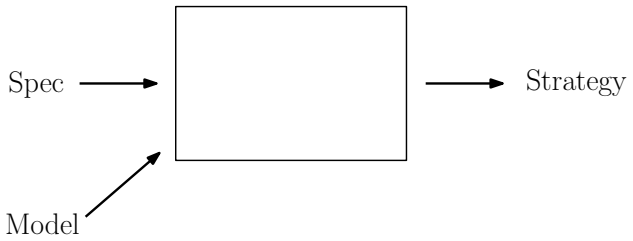
$$F = \{(B_1, G_1), (B_2, G_2), ...\}$$

condition is

- $Inf(\pi) \cap G_i \neq \emptyset$, visits G_i infinitely often
- $Inf(\pi) \cap B_i = \emptyset$, visits B_i maximum of finitely often

Any LTL can be converted to non-deterministic Büchi automaton. And any LTL can be converted to deterministic parity automaton.

If we have a model (e.g., robot), what needs to be done?
In practice, we often do not have a model ready and so we cannot write the model as part of LTL. This increases the size of specification and solving it is exponential computational time in size of specification.



Instead, we can write the model as an automaton and when we get a strategy, we can check of satisfiability of the product of automata (specification and model).

Other types of Temporal Logic

GR(1): Generalized reactivity(1)

$$\varphi : \varphi_e \rightarrow \varphi_g$$

where φ_e, φ_g is the environment and guaranteed states.

This is for discrete space. Commonly used in task planning. Tulip converts GR(1) specs to control strategies (developed in Caltech). LTLMOP, Linear Temporal Logic Mission Planner is developed in Cornell.

Signal Temporal Logic

This is used for a continuous timed systems (when the sequence of states is a signal). This can be used for real valued variables. For example, we can have

$$G_{[5,10]}(r \rightarrow F_{[7,11]}g).$$

This specification states that globally, between 5 to 10 seconds, when there is a request, it will eventually be granted between 7 to 11 seconds. Here, we have have a notion of intervals and thus there is no notion of next here. This helps us in writing specifications for real life systems. But, we do not have machinery to solve them. One of the ways is to use LTL tools on discretized space (but computation blows up exponentially). Another approach is to write the specification as an optimization problem and use mixed integer programming to solve it. This loses guarantees from Temporal logic. An example of an optimization problem would be:

$$\min J(\xi(t)) \;\; s.t. \;\; \xi(t) \models G_{[5,10]}(\xi(t) > 10) \wedge ...$$

where ξ is trajectory and J is a cost function. This optimization problem says that we want to minimize the cost of the trajectory ξ such that ξ satisfy the STL specification of always being above 10 between 5 to 10 seconds.