Lecture 14
Reactive Systems Introduction

Armando Solar-Lezama and Xiaokang Qiu
with some slides from Adam Chlipala
Example 1: vending machine

Takes 3 coins to get a coke

If you put 3 coins, you can select a drink (as long as you don’t cancel in the middle)

After you select your drink you get your drink

If you press cancel, you get your money back
Example 2: ignition button

- Engine starts and stops with button push
- If engine is off, it stays off until I push
  - If I never push it stays off forever
- If engine is on, it stays on until I push
  - If I never push it stays on forever
What is a program

Environment ⟷ System

Environment → Program ← System

Input → Program → Output

Inputs ⟷ Outputs
Key Questions for reactive systems

How do we specify their behavior?
- Pre/post conditions are not adequate
- Even concrete scenarios are infinite
- Environment may be reactive too

Models vs Programs?
- Finite state models of reactive systems
- Key tradeoffs in synthesizing models vs. synthesizing programs

How do you synthesize them?
- Taking advantage of finite state models for synthesis
Behavior specification

```
while(true) {
    read inputs;
    make decisions;
    update state;
    write outputs;
}
```

Environment  

System  

Inputs

Preconditions

Postconditions

Inputs

Outputs

Environment

System
Behavior specification

Option 1: shoehorn it into the functional paradigm

Problems:
- Environment itself is reactive (future inputs depend on current outputs)
- Need to write predicates on infinite sequences of values
- Standard verification machinery does not work for non-terminating programs
Predicates on infinite sequences of values

“If you put 3 coins, you can select a drink (as long as you don’t cancel in the middle)”

Inputs: what can the environment (user) do to the machine
- Put a coin (coin)
- Select a drink (select)
- Press cancel (cancel)
- Nothing ()

Outputs: what can the machine do
- Nothing ()
- Produce a drink (drink)
Predicates on infinite sequences of values

“If you put 3 coins, you get a drink”

• Input: $in_t: \{Input\}$  Output: $out_t: \{Output\}$

\[
\begin{align*}
in_{t_1} &= (coin) \\
\forall t_1, t_2, t_3 \quad t_1 < t_2 < t_3 &\Rightarrow \land in_{t_2} = (coin) \Rightarrow \exists t_4 > t_3 \quad out_{t_4} = (drink) \\
&\land in_{t_3} = (coin)
\end{align*}
\]

Inputs: what can the environment (user) do to the machine
- Put a coin (coin)
- Select a drink (select)
- Press cancel (cancel)
- Nothing ()

Outputs: what can the machine do
- Nothing ()
- Produce a drink (drink)
Predicates on infinite sequences of values

“If you put 3 coins, you can select a drink (as long as you don’t cancel in the middle)”
- Input: $in_i: \{Input\}$  Output: $out_i: \{Output\}$

Problem: Formulas are too complex to handle with the technology we have explored so far

Inputs: what can the environment (user) do to the machine
- Put a coin (coin)
- Select a drink (select)
- Press cancel (cancel)
- Nothing ()

Outputs: what can the machine do
- Nothing ()
- Produce a drink (drink)
Step 1: Properties of a single step

Atomic propositions: Set of predicates that relate to the state at a given moment in time

- coin, select, cancel, drink can be used as atomic propositions in the example
- Shorthand for $in_i = (\text{coin})$ or $out_i = (\text{drink})$ on a given step $i$
- Can stand for arbitrarily complex properties

State predicates

- Boolean predicates over atomic propositions
- Can also be established as true or false on a given step
- If $p \in \{\text{AP}\}$ then $p$ is a state formula
- if $f$ and $g$ are state formulas, so are $(f \text{ and } g)$, $(\text{not } f)$, $(f \text{ or } g)$
- Ex. $(\neg \text{cancel} \wedge \text{coin})$
Step 2: Properties over sequences of steps

Linear Temporal Logic (LTL)

Let $\pi$ be a sequence of steps

- $\pi := s_0, s_1, s_2, \ldots$
- Let $\pi_i$ be a subsequence starting at $i$

LTL allows us to define predicates over sequences of steps
Sequences of steps = paths

Path formulas
- a state formula $p$ is also a path formula
  - $p(\pi_i) := p(s_i)$
- boolean operations on path formulas are path formulas
  - $f$ and $g(\pi_i) := f(\pi_i)$ and $g(\pi_i)$
- path quantifiers
  - $G f(\pi_i) := \text{globally } f(\pi_i) = \forall k \geq i \ f(\pi_k)$ (may abbreviate as $\blacksquare$)
  - $F f(\pi_i) := \text{eventually } f(\pi_i) = \exists k \geq i \ f(\pi_k)$ (may abbreviate as $\Diamond$)
  - $X f(\pi_i) := \text{next } f(\pi_i) = f(\pi_{i+1})$ (may abbreviate as $\bigcirc$)
  - $f U g(\pi_i) := f \text{ until } g = \exists k \geq i \ g(\pi_k) \land \forall j: i \leq j < k \ f(\pi_j)$

Given a formula $f$ and a path $\pi$,
- if $f(\pi)$ is true, we say that $\pi \models f$
Examples

If you submit your homework (submit) you eventually get a grade back (grade)
- $G (\text{submit} \Rightarrow F \text{ grade})$

You should get your grade before you submit the next homework
- $G (\text{submit} \Rightarrow X (\neg \text{submit U grade}))$
  - What’s wrong with $G (\text{submit} \Rightarrow (\neg \text{submit U grade}))$?

If assignment $i$ was submitted before drop date, you should get your grade before drop date
- $(G (\text{submit}_i \Rightarrow F \text{ dropDate})) \Rightarrow ((G (\text{grade}_i \Rightarrow F \text{ dropDate})))$
  - and $G (\text{submit} \Rightarrow F \text{ grade})$
Review of Temporal Logic

Engine starts and stops with button push

• If engine is off, it stays off until I push
  • If I never push it stays off forever
• If engine is on, it stays on until I push
  • If I never push it stays on forever

\[ G \text{off} \Rightarrow \text{off} \lor \text{push} \]
\[ G (\text{off} \Rightarrow (\text{off} \lor \text{push} \lor G \text{off})) \]
\[ G (\text{on} \Rightarrow (\text{on} \lor \text{push} \lor G \text{on})) \]