15-424/15-624/15-824 Recitation 5
Event and Time-Triggered Control
15-424/15-624/15-824 Foundations of Cyber-Physical Systems

Today we will first talk about converting last week's Ping Pong model from event-triggered to time-triggered, then we will work through a differential invariant proof.

1 Ping-Pong Time(-Triggers)

Recall last week we built an event-triggered model of two people playing ping-pong. An **event-triggered controller** is a controller that performs an action whenever an event occurs. Events are typically chosen carefully so that they are both observable (i.e., a sensor can be used to detect the event without much delay) and useful (i.e., they help establish the post-condition).

Here is a correct event-triggered model for ping-pong:

$v \geq 0 \land l \leq x \leq r \land l < r \rightarrow$
[{
  $\{?x = r; v := -1 \cup ?x = l; v := 1 \cup ?(x \neq l \land x \neq r)\}$;
  $\{x' = v \& l \leq x \land x \leq r\} \cup \{x' = v \& x \leq l \lor x \geq r\}$
}$*$]$(l \leq x \leq r)$

The easiest proof of safety for this system simply solves differential equations after choosing a good loop invariant.

2 The Time-Triggered Controller

The problem with event-triggered models is that they can be unrealistic: in reality we don't learn about events as soon as they happen. Instead, it takes some time for us to detect events. Since many bugs in real implementations are due to timing mistakes, we can catch more bugs when we build a more detailed model that accounts for time. As a metaphor, consider what happens if you asked Piazza questions last night before Theory 2 was due. Say it takes you an hour to finish typesetting once Brandon answers your question (based on the continuous dynamics of your keyboard). In an event-triggered model, Brandon immediately detected a Piazza question and answered it, so you could ask a question at 11pm and that would be safe. In reality, Brandon is busy sometimes, but checks Piazza at least every 2 hours (we say system delay is $T = 2$hrs. Asking questions at 11pm is *not safe* because Brandon might be busy, but asking them at 9 is safe becauses you know that he will respond by 11. By modeling response time explicitly, we arrive at a safer system design.

Next, we will transform the above ping-pong model to a time-triggered model. In a time-triggered system the environment no longer tells us when a particular event happens. Instead, a time-triggered system checks for events at regular time intervals.

```
v >= 0 & l <= x & x <= r & l < r & T > 0 ->
[ {{?(x=l); v := 1;}
 ++{?(x=r); v := -1;}
 ++{?(x!=l & x != r);}}
 t:= 0;
 {x'=v, t'=1, t <= T}
](l <= x & x <= r)
```

However, simply adding a time variable and its dynamics is not sufficient. Why? We are now solving a harder control problem. In an event-triggered model, we designed a controller that reacts safely if it gets warned about danger *exactly* when it needs to react. In the time-triggered model, we need to design a controller that reacts safely *even* if it only hears about impending danger time T before the danger really

happens. This means we need to design a more conservative controller, one that asks "If I waited time T to start breaking, would it be too late?". Any time the answer is "yes" the controller needs to break right away.

Similarly, in the ping-pong model, we need to think ahead! We can't just change the velocity when we reach $x = l$ or $x = r$, because our sensors aren't good enough to notify us exactly when $x = l$ or $x = r$!. Instead we only get to run the controller every T time, so the question we need to ask is: "Would I hit the wall within the next T time?". If so, we need to hit the ball immediately. Since the ball evolves continuously with constant velocity, the position at time T is just $x + v \cdot T$. Our controller can just compare $x + v \cdot T$ with l and r to determine if it's time to hit the ball. Additionally, we need to know that the ball will take longer than time T to cross the table, otherwise we might never get a chance to hit the ball back after our opponent hits it. We can just write this as $l + T < r$ since the velocity always has magnitude 1.

```
v^2 = 1 & l <= x & x <= r & l + 2 * T < r & T > 0 ->
[{{{?(x + v*T < l); v := 1;}
 ++{?(x + v*T > r); v := -1;}
 ++{?(l <= x + v*T & x + v*T <= r);}}
 t:= 0;
 {x'=v, t'=1, t <= T}
 }*@invariant(l <= x & x <= r & v^2 = 1)
](l <= x & x <= r)
```

Informal correctness proof:

- Cases 1 and 2: If we hit the ball, then by $l + 2 \cdot T < r$ the ball must have been "close" to that player's wall, so it is now distance T or more from the opponent's wall, so at time T it is still in the playing area.
- Case 3: If we took branch 3, we were distance T or more from both walls, so we are still in the playing area.

3 Verifying Planes with Differential Invariants

The first week of class, Brandon had to go fly to Paris for a research conference. He made the mistake of taking Unification Airways Flight 424 for his flight from Pittsburgh to Newark. Pittsburgh is surrounded by the Allegheny Mountains and the pilots at Unification Airways are not very careful. On the way out of Pittsburgh, they almost flew into Mt. Washington and crashed. Luckily, every commercial flight in the US is equipped with a Ground Proximity Warning System (GPWS) that warns the pilot if they get too close to the ground. We're going to prove that if the pilot follows the GPWS warning, I don't crash into Mt. Washington and die. Nice! We're going to do the proof with differential invariants. Also nice![1]

The plane starts out at altitude 1 and GPWS tells the pilot to accelerate up with acceleration $a = 2$. We could model this as the following hybrid program:

$$\alpha' \equiv x := 0, y := 1; a := 2; \{x' = 1, y' = v_y, v_y' = a\}$$

In order to simplify today's proof, we're actually going to assume we've already solved for v_y, giving us an easier program:

$$\alpha \equiv x := 0, y := 1; \{x' = 1, y' = 2x\}$$

This will give us a trajectory of $y = x^2 + 1$, (because $v_y' = 2, y' = 2 \cdot x, y(0) = 1$) whereas the edge of Mt. Washington is $y = x$. QE can tell us $x^2 + 1 > x$, so all we have to do is prove that the plane follows the correct trajectory. Recall from Recitation 3 that paths in 2D space make nice invariants!

[1] You may have noticed this specific proof can be done with ODE solve as well. That is true, but differential invariants also work when solving an ODE is either totally impossible or much too difficult. However, it's nice to start out with a simple proof, and lots of the easiest proofs also happen to work with ODE.

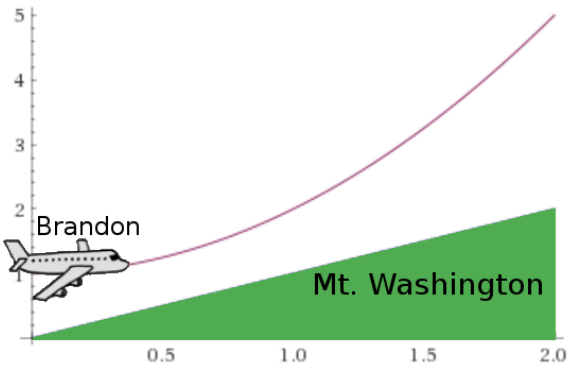

Figure 1: The Reason You Still Have a TA

3.1 Discrete invariants: Simulating ODEs with Loops

Recall from Lecture 1 that one way to get intuition about ODEs is to look at how we would simulate them. In this recitation we will reuse that intuition to take proofs about loop invariants and turn them into proofs about differential invariants. An intern at Unification Airways wrote a simulation for $y = x^2 + 1$ so luckily we can borrow her code. To get started, recall their's a cute recursive formula for the squares of natural numbers:

$$sq(0) = 0$$
$$sq(n+1) = sq(n) + 2 \cdot (n) + 1$$

If we express this property as a loop, we get a hybrid program for computing square numbers:

$$\alpha_1 \equiv (y := y + 2 \cdot x + 1; x := x + 1)^*$$

We can prove that this program maintains the property $y = x^2 + 1$ as an invariant.

$$\text{loop}\,\frac{\mathbb{R}\,\dfrac{*}{x = 0 \wedge y = 1 \vdash y = x^2 + 1} \quad [;]\,\dfrac{[:=]\,\dfrac{[:=]\,\dfrac{\mathbb{R}\,\dfrac{*}{y = x^2 + 1 \vdash y + 2 \cdot x + 1 = (x+1)^2 + 1}}{y = x^2 + 1 \vdash [y := y + 2 \cdot x + 1]y = (x+1)^2 + 1}}{y = x^2 + 1 \vdash [y := y + 2 \cdot x + 1][x := x + 1]y = x^2 + 1} \quad \text{id}\,\dfrac{*}{y = x^2 + 1 \vdash y = x^2 + 1}}{x = 0 \wedge y = 1 \vdash [(y := y + 2 \cdot x + 1; x := x + 1)^*]y = x^2 + 1}$$

Let's pay special attention to the second $\mathbb{R}$ goal:

$$y = x^2 + 1 \vdash y + 2 \cdot x + 1 = (x+1)^2 + 1$$

QE can tell us why this was true, but it doesn't give us intuition why α_1 is the right program to compute $y = x^2 + 1$. Because we're trying to simulate an ODE, we should think about how much the quantities x and y *change* at each loop iteration, so let's define $\Delta y = 2 \cdot x + 1, \Delta x = 1$ to denote how much each value changes at each step. Because $y = x^2 + 1$ holds at the beginning of each step, $y = x^2 + 1$ will be an invariant as long as the *change in each side* is the same, i.e. $\Delta y = \Delta\left[x^2 + 1\right]$ where $\Delta\left[x^2 + 1\right] = (x+1)^2 - x^2 = 2x + 1$.

Graphically, this means the simulation always plots points that are really on the trajectory of Brandon's plane. But this still isn't very satisfactory, because it only let's us see what happens every 1 time unit.

The intern updated her code to allow arbitrarily small timesteps ϵ. Notice this is a bit more complicated: some of the factors of ϵ were invisible before because they were all 1.

$$\alpha_2 \equiv (y := y + 2 \cdot x \cdot \epsilon + \epsilon^2; x := x + \epsilon)^*$$

She also updated her proof to show that the new simulation is also sound:

$$\text{loop}\,\frac{\mathbb{R}\,\frac{*}{x = 0 \wedge y = 1 \vdash y = x^2 + 1} \quad [;]\,\frac{[:=]\,\frac{[:=]\,\frac{\mathbb{R}\,\frac{*}{y = x^2 + 1 \vdash y + 2 \cdot x \cdot \epsilon + \epsilon^2 = (x + \epsilon)^2 + 1}}{y = x^2 + 1 \vdash [y := y + 2 \cdot x + 1]y = (x + 1)^2 + 1}}{y = x^2 + 1 \vdash [y := y + 2 \cdot x \cdot \epsilon + \epsilon^2][x := x + \epsilon]y = x^2 + 1}}{y = x^2 + 1 \vdash [(y := y + 2 \cdot x \cdot \epsilon + \epsilon^2; x := x + \epsilon)]y = x^2 + 1} \quad \text{id}\,\frac{*}{y = x^2 + 1 \vdash y = x^2 + 1}}{x = 0 \wedge y = 1 \wedge \epsilon > 0 \vdash [(y := y + 2 \cdot x \cdot \epsilon + \epsilon^2; x := x + \epsilon)^*]y = x^2 + 1}$$

3.2 From Discrete Dynamics to Continuous Dynamics

No matter how small ϵ gets, α_2 is still a discrete program so it still only tells us about what happens every ϵ seconds! Of course, to fix this, we need a differential equation:

$$\alpha_3 = \{x' = 1, y' = 2 \cdot x\}$$

How do we get this ODE? First, look at how much x and y change per interation:

$$\Delta x = \epsilon, \Delta y = 2 \cdot x \cdot \epsilon + \epsilon^2$$

Then by the definition of derivative: [2]

$$\begin{aligned} x' &= \lim_{\epsilon \to 0^+} \Delta x/\epsilon = 1 \\ y' &= \lim_{\epsilon \to 0^+} \Delta y/\epsilon \\ &= \lim_{\epsilon \to 0^+} 2 \cdot x + \epsilon \\ &= 2 \cdot x \end{aligned}$$

We won't bother proving it[3], but we could say the ODE is (semantically) the limit of the loop as $\epsilon \to 0$:

$$[[\alpha_3]] = \lim_{\epsilon \to 0^+} [[\alpha_2]]$$

3.3 Differential Invariants

What this means is that differential equations are, in a deep sense, like loops that happen continuously (i.e. infinitely often). Therefore the proof rules should be very similar. The difference will be that where `loop` had us prove what happens during one step of discrete change (i.e. compare Δx and Δy), we will now be looking at continuous change: x' vs. y'.

Without further ado, let's prove the safety theorem for α_3:

$$x = 0 \wedge y = 1 \vdash [\alpha_3]y = x^2 + 1$$

Naturally this will start off with the DI rule from yesterday's lecture: [4]

$$\text{DI}\,\frac{\Gamma \vdash e = 0 \qquad e = 0 \vdash P \qquad Q \vdash [x' = \theta \& Q]((e)' = 0)}{\Gamma \vdash [x' = \theta \& Q]P}$$

[2] We are never going to make you use limits, I just thought this helps explain the relation between the loop and ODE.
[3] This proof is probably really hard.
[4] I wrote the branches in a different order than Andre does because I wanted to talk about the easy branches first.

We already know what the precondition is, so we choose $P = (y = x^2 + 1)$. But $y = x^2 + 1$ doesn't look like $e = 0$ so we have to pay attention when choosing e. Luckily by subtracting from both sides, we get the equivalent formula $y - x^2 - 1 = 0$, so in this case we can choose $e = y - x^2 - 1$. [5]

Before we go any further, does anybody notice something we had in the loop rule that's missing for the DI rule? That's right! We can't assume the invariant $e = 0$ when proving $(e)' = 0$ — that would actually be unsound. The technical reasons are subtle, but the intuition is that now we have infinitely short induction steps, and so if we assumed the invariant for "the last step" and "the last step" was an infinitely short time ago, that's actually like assuming the conclusion. However, this won't be a limitation. By comparison with the loop invariant above, all `DI` asked us to prove is $\Delta y - x^2 - 1 = 0$, and that was true without assuming the invariant $y - x^2 + 1 = 0$!

$$\text{DI}\ \frac{x = 0, y = 1 \vdash y - x^2 - 1 = 0 \qquad y - x^2 - 1 = 0 \vdash y = x^2 + 1 \qquad \vdash [x' = 1, y' = 2x]((y - x^2 - 1)' = 0)}{x = 0, y = 1 \vdash [x' = 1, y' = 2x]y - x^2 - 1 = 0}$$

The first two branches close easily by QE, so let's focus on the last branch.

$$\vdash [x' = 1, y' = 2x]((y - x^2 - 1)' = 0)$$

What do we do next? Well, we can start by simplifying the derivative using the differentiation axioms from lecture:

$$\begin{aligned}(\theta_1 - \theta_2)' &= (\theta_1)' - (\theta_2)' \\ (\theta_1 \cdot \theta_2)' &= (\theta_1 \cdot (\theta_2)') + ((\theta_1)' + \theta_2) \\ (\theta^2)' &= 2 \cdot \theta \cdot (\theta)' \\ (x)' &= x' \\ (c())' &= 0\end{aligned}$$

$$\text{derive}\ \frac{\vdash [x' = 1, y' = 2x](y' - 2 \cdot x \cdot x' - 0 = 0)}{\vdash [x' = 1, y' = 2x]((y - x^2 - 1)' = 0)}$$

Notice that once we simplify the derivative, it still talks about derivatives of variables such as x' and y'. If we want to finish the proof we'll need to know what those are. Luckily, as the $=$ symbol suggests, $x' = \theta$ is a true equality everywhere throughout the ODE $\{x' = \theta\}$! This reasoning step is implemented with a rule called *differential effect,* which makes ODEs into assignments:

$$\begin{array}{c}[:=]\ \dfrac{\vdash [x' = 1, y' = 2x](2 \cdot x - 2 \cdot x \cdot 1 - 0 = 0)}{\vdash [x' = 1, y' = 2x][x' := 1](2 \cdot x - 2 \cdot x \cdot x' - 0 = 0)} \\ [:=]\ \dfrac{\vdash [x' = 1, y' = 2x][x' := 1][y' := 2x](y' - 2 \cdot x \cdot x' - 0 = 0)}{\vdash [x' = 1, y' = 2x][x' := 1][y' := 2x](y' - 2 \cdot x \cdot x' - 0 = 0)} \\ \text{DE}\ \dfrac{}{\vdash [x' = 1, y' = 2x](y' - 2 \cdot x \cdot x' - 0 = 0)}\end{array}$$

At this point we just have some arithmetic inside of the ODE, and that arithmetic looks really, really true. What do we do when we want to throw away a differential equation? Differential weakening, dW! [6]

$$\text{dW}\ \frac{\mathbb{R}\ \dfrac{*}{\vdash 2 \cdot x - 2 \cdot x \cdot 1 - 0 = 0}}{\vdash [x' = 1, y' = 2x](2 \cdot x - 2 \cdot x \cdot 1 - 0 = 0)}$$

Putting this all together, we can get the complete proof:

[5] This example was carefully chosen so that we could do this. In next week's lectures, Andre will talk in more detail about differential invariants for more complicated formulas.

[6] If you thought G or GV, you are also correct.

$$\text{DI}\ \frac{\mathbb{R}\ \dfrac{*}{x=0, y=1 \vdash y - x^2 - 1 = 0} \qquad \mathbb{R}\ \dfrac{*}{y - x^2 - 1 = 0 \vdash y = x^2 + 1} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbb{R}\ \dfrac{*}{\vdash 2 \cdot x - 2 \cdot x \cdot 1 - 0 = 0}}{\vdash [x' = 1, y' = 2x](2 \cdot x - 2 \cdot x \cdot 1 - 0 = 0)}\ \text{dW}}{\vdash [x' = 1, y' = 2x][x' := 1](2 \cdot x - 2 \cdot x \cdot x' - 0 = 0)}\ [:=]}{\vdash [x' = 1, y' = 2x][x' := 1][y' := 2x](y' - 2 \cdot x \cdot x' - 0 = 0)}\ [:=]}{\vdash [x' = 1, y' = 2x](y' - 2 \cdot x \cdot x' - 0 = 0)}\ \text{DE}}{\vdash [x' = 1, y' = 2x]((y - x^2 - 1)' = 0)}\ \text{derive}}{x=0, y=1 \vdash [x' = 1, y' = 2x]y - x^2 - 1 = 0}$$

We use the same structure for all (basic) DI proofs:

1. Pick an invariant
2. Prove that $Pre \to Inv$ and $Inv \to Post$. This will usually be by `master`.
3. Derive
4. DE
5. [:=]
6. dW
7. Now the ODE is gone, so finish the proof! (this is often, but not always `master`)

Does that sound like a lot of work? *It is*! But notice the only *creative* step was (1) (and maybe (6) sometimes). KeYmaera X likes to do the boring steps for you, so it has a `dI` tactic where you give it an invariant and it does the rest for you!

Even better: you can use the "details" feature from last week to look inside `dI` and see what the `dI` proof looks like! Similarly, if `dI` *fails*, you will now get a "Progress Until Error" button with the error message, which you can use to inspect what happened before the error, and what state the proof was in when it failed.