15-424/15-624 Recitation 5
Events, Delays and constraint discovery
15-424/15-624 Foundations of Cyber-Physical Systems

The latter part of these notes were taken from Stefan Mitsch's (`smitsch+fcps@cs`) notes from last year. Thanks Stefan!

1. **Using KeYmaera to discover constraints:** when developing a CPS model for use in KeYmaera, it's easy to forget some important initial conditions, or, if the problem is relatively complex, it's even possible that the designer doesn't have a complete understanding of the model.

   In these situations, KeYmaera can be used as an aid in order to *discover* constraints that would help make the model safe. Recall that quantifier elimination is a procedure that, given a formula, yields an equivalent quantifier-free formula. Sometimes, that formula is `true`, and the proof progresses, but sometimes this formula represents exactly what is needed for the system to be proven!

   To find out how to use KeYmaera in this way, please watch the wonderful tutorial video[1]. In more recent versions of KeYmaera, you will have to disable the "universal closure on QE" options. You can find it under the Hybrid Strategy tab, then Advanced. **Do not forget to reenable it afterwards!**

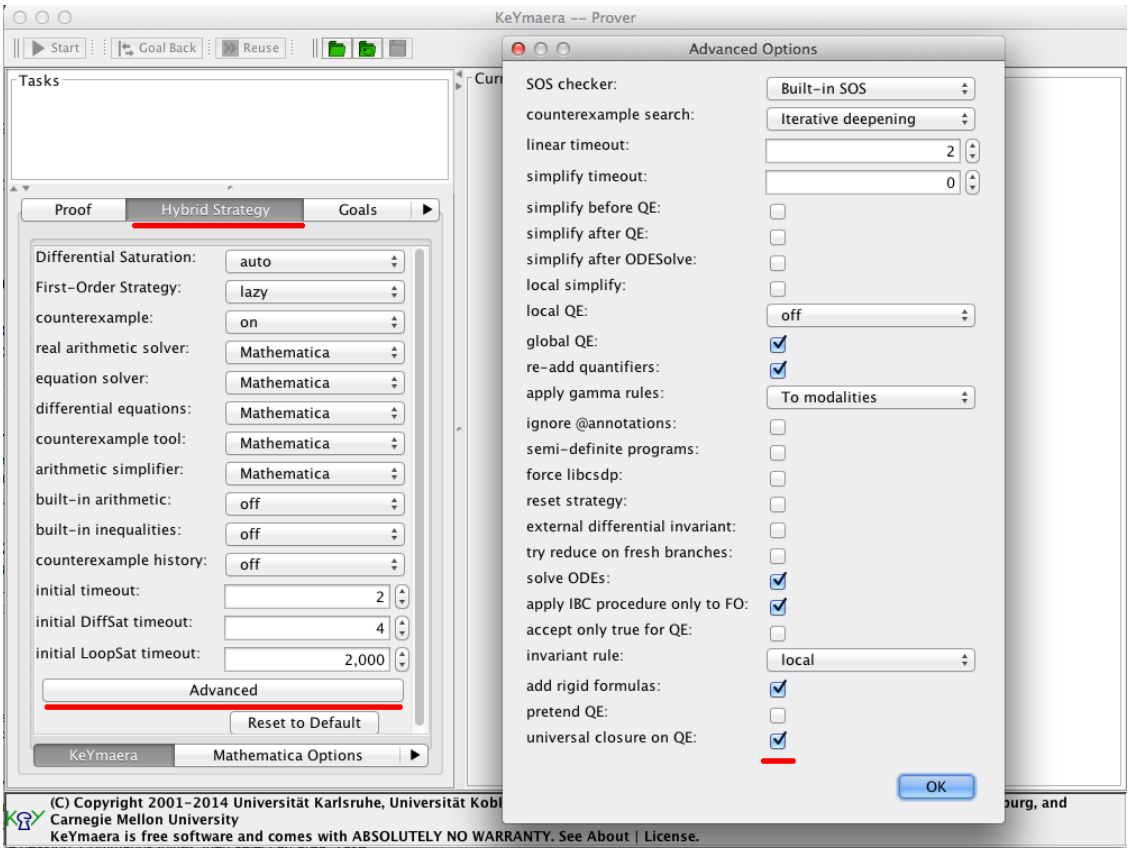

Figure 1: Find the weird option you need to disable.

[1] `https://www.youtube.com/watch?v=0fpRccI4Occ`

2. **A proof by hand:** let's prove $x = 5 \rightarrow [(x := 5; x := x + 1) \cup (x := x + 1; x := 5)]\,(x = 5 \vee x = 6)$, and let $\text{inv} \equiv x = 5 \vee x = 6$

$$\dfrac{\dfrac{\text{ax}\ \dfrac{*}{x = 5 \vdash x = 5, x = 6}}{\vee_r\ \dfrac{}{x = 5 \vdash x = 5 \vee x = 6}} \quad \text{ax}\ \dfrac{*}{x = 5 \vee x = 6 \vdash x = 5 \vee x = 6} \qquad \text{Other things}}{\dfrac{\text{ind'}\ \dfrac{x = 5 \vdash [((x := 5; x := x + 1) \cup (x := x + 1; x := 5))^*]\,(x = 5 \vee x = 6)}{x = 5 \vdash [((x := 5; x := x + 1) \cup (x := x + 1; x := 5))^*]\,(x = 5 \vee x = 6)}}{\rightarrow_r\ \dfrac{}{\vdash x = 5 \rightarrow [((x := 5; x := x + 1) \cup (x := x + 1; x := 5))^*]\,(x = 5 \vee x = 6)}}}$$

$$[\cup], \wedge_r\ \dfrac{\dfrac{\dfrac{\dfrac{*}{\text{inv} \vdash 5 + 1 = 5 \vee 5 + 1 = 6}}{\text{inv} \vdash [x := 5 + 1](x = 5 \vee x = 6)}}{\text{inv} \vdash [x := 5][x := x + 1]\text{inv}}}{\text{inv} \vdash [x := 5; x := x + 1]\text{inv}} \quad \dfrac{\dfrac{\dfrac{\dfrac{*}{\text{inv} \vdash 5 = 5, 5 = 6}}{\text{inv} \vdash 5 = 5 \vee 5 = 6}}{\text{inv} \vdash [x := 5](x = 5 \vee x = 6)}}{\text{inv} \vdash [x := x + 1][x := 5]\text{inv}}}{\text{inv} \vdash [x := x + 1; x := 5]\text{inv}}$$

$$\text{inv} \vdash [(x := 5; x := x + 1) \cup (x := x + 1; x := 5)]\,\text{inv}$$

3. **Quick notes on time-triggered proofs:** when you have a time-triggered model, your controller has to take into account the fact that once it makes a choice, it will have to commit to it for T time. For instance, if a car decides to accelerate, it needs to assume that in the worst case, it will accelerate for T time, and still needs to be safe after that. In other words, control decisions of time-triggered systems are critically dependent on this time T.

   When actually proving the system, you're faced with loop invariants. The question is whether these loop invariants should depend on T. At first sight, that seems wise. After the loop executes, the controller will still have to commit for T time units, so the loop invariant should acknowledge that fact.

   Interestingly, loop invariants don't really need to talk about T. Ultimately, it's the controller's job to ensure that, no matter how much time passes (up to T), the system is safe. Therefore, all the loop invariant needs to prove is that... the system is safe *right now*! There are situations where "looking into the future" by having T in the loop invariant can still work, but sometimes that might be too conservative, and thus impossible to prove! Furthermore, if we don't mention T in the loop invariant, we are simplifying arithmetic and reducing the number of variables! KeYmaera & Mathematica thank you kindly!

4. **Discussion: Why must we be careful with evolution domain constraints?**

   Let us suppose that we want to create a controller for a moving point that should stop before reaching a maximum position. The moving point can choose its velocity instantaneously.

   We begin with an event-driven architecture: our controller should be notified when the point is about to exceed the maximum point. Now consider the following controller, which is obviously unsafe.

$$x \leq max \rightarrow \Big[\ \big(\ \text{v} := 1000; \\ \quad \{x' = v,\ x \leq max\} \\ \quad \big)^*@\text{invariant}(x \leq max) \\ \Big](x \leq max)$$

The dℒ formula, however, is still valid, since the evolution domain constrant clips all undesired behavior just for the sake of not missing the event we are interested in. When we trust in this controller (we have proof, so it must be safe, right?) we would still end up in unsafe situations because the real world just does not respect our evolution domain constraint. This means that our evolution domain constraint is too restricted; it is not a good model of the real world.

KeYmaera can help us detect the bug in the controller, if we model dynamics outside the event as well, as in the following dℒ formula.

$$x \leq max \rightarrow \Big[\big(\text{v} := 1000; \\ (\{x' = v,\ x \leq max\} \cup \{x' = v,\ x \geq max\}) \\ \big)^* @\text{invariant}(x \leq max) \\ \Big](x \leq max)$$

Here it is crucial to model overlapping evolution domain constraints, since otherwise we could never switch between the two models of continuous dynamics.

We can fix the controller when we allow the point to move only when it did not yet exceed the maximum position.

$$x \leq max \rightarrow \Big[\big(\text{if } (x < max) \text{ then } v := 1000 \text{ else } v := 0 \text{ fi}; \\ (\{x' = v,\ x \leq max\} \cup \{x' = v,\ x \geq max\}) \\ \big)^* @\text{invariant}(l \leq x \leq r) \\ \Big](l \leq x \leq r)$$

5. **Example: How can we turn an event-driven architecture into a time-triggered system?**

   Event-driven models are often easier to verify, but they are hard (if not impossible) to implement, because such models require that someone in the environment detects the event at precisely the correct time. Time-triggered systems, which observe the environment for events in recurring intervals, are often more suitable for implementation purposes.

   We can turn any event-driven architecture in a time-triggered system:

   - Remove the evolution domain constraints for instantaneous event detection
   - Introduce a clock t and a sampling interval T
   - Adapt the controller: it may now take up to time T until the event is detected (solve the differential equations to find out how much safety margin you need)

$$x \leq max \wedge T > 0 \rightarrow \Big[\big(\text{if } (x + 1000T < max) \text{ then } v := 1000 \text{ else } v := 0 \text{ fi}; \\ \text{t} := 0; \\ \{x' = v,\ t' = 1,\ t \leq T\} \\ \big)^* @\text{invariant}(x \leq max) \\ \Big](x \leq max)$$

6. **Further Examples: Moving Point in an Interval**

   How can we create event-triggered control to keep the position of a moving point inside an interval $[l, r]$? The point is moving with constant velocity v to the right (i.e., closer to r) or constant velocity $-v$ to the left (i.e., closer to l). Our moving point is an unsteady one: if its position is closer to the right border, it moves to the left; if it's closer to the left border, the point moves to the right. How can we make the point stay within the interval?

$$v \geq 0 \land l \leq x \leq r \rightarrow \Big[\big(\text{if } \big(x > \tfrac{r+l}{2}\big) \text{ then } v := -4 \text{ else } v := 4 \text{ fi}; \{x' = v,\ l \leq x \leq r\} \big)^* @\text{invariant}(l \leq x \leq r) \Big](l \leq x \leq r)$$

Since the point can change its velocity instantaneously, we do not need extra space when turning around (such as for stopping with some braking force). Thus, the event we are interested in is when the moving point is about to cross either of the two interval bounds. We could simply include an evolution domain constraint that stops continuous dynamics when the point is about to leave the interval. But wait: unless the interval denotes two walls, the evolution domain constraint above cuts off perfectly realistic behavior. Remember that physics is rather non-negotiable. So, if we put in unrealistic physics models we may do just fine during the proof, but still build an unsafe system because the real world behaves differently.

What can we do to fix our model?

$$v \geq 0 \land l \leq x \leq r \rightarrow \Big[\big(\text{if } \big(x > \tfrac{r+l}{2}\big) \text{ then } v := -4 \text{ else } v := 4 \text{ fi}; \{x' = v,\ l \leq x \leq r\} \cup \{x' = v,\ x \leq l \lor x \geq r\} \big)^* @\text{invariant}(l \leq x \leq r) \Big](l \leq x \leq r)$$

The only reason why we added the evolution domain constraint in the first place was that we did not want to miss the event when the moving point crossed either of the interval's boundaries. So, we can simply add alternative continuous dynamics with an evolution domain constraint that watches for interval bound crossing from the outside. The evolution domain constraints of both alternatives ensure that we can follow either of the two evolutions and still won't miss the boundary crossing. Note: It is crucial to use overlapping evolution domain constraints. Otherwise, the program could never switch between the two alternatives.

Is this model valid? We use proof (this time with KeYmaera) to find out. We apply the proof rules of $\mathsf{d}\mathcal{L}$ to syntactically decompose our model and find a counter-example.

What does this counter-example tell us and how can we fix the model?

$$v \geq 0 \land l \leq x \leq r \land l < r \rightarrow \Big[\big(\text{if } \big(x > \tfrac{r+l}{2}\big) \text{ then } v := -4 \text{ else } v := 4 \text{ fi}; \{x' = v,\ l \leq x \leq r\} \cup \{x' = v,\ x \leq l \lor x \geq r\} \big)^* @\text{invariant}(l \leq x \leq r) \Big](l \leq x \leq r)$$

The counter-example showed us that the moving point cannot be strictly to the right of the interval center when the interval is empty. So, we introduce the additional precondition to fix our model.

Next, we want to see how we can make the moving point model easier to implement by switching to a time-triggered system. In a time-triggered system the environment does no longer tell us when a particular event happens. Instead, a time-triggered system checks for events at regular time intervals.

$$v \geq 0 \land l \leq x \leq r \land l < r \land T > 0 \rightarrow \Big[\big(\text{ if } \big(x > \tfrac{r+l}{2}\big) \text{ then } v := -4 \text{ else } v := 4 \text{ fi}; \text{ t } := 0; \{x' = v,\ t' = 1,\ t \leq T\} \big)^* @\text{invariant}(l \leq x \leq r) \Big](l \leq x \leq r)$$

However, simply introducing time is not sufficient, since the moving point is now no longer instantaneously informed when it is about to leave the interval. We have to consider the additional distance that the point may move in the worst case until it notices that it is near the boundary. This entails that we need a sufficiently large interval; otherwise, the moving point will not be able to move anywhere.

$$v^2 = 16 \land l \leq x \leq r \land l + 8T < r \land T > 0 \rightarrow \Big[\big(\text{ if } \big(x > \tfrac{r+l}{2} \land v \geq 0 \land x + vT > r\big) \text{ then } v := -4 \text{ fi}; \text{ if } \big(x \leq \tfrac{r+l}{2} \land v \leq 0 \land x + vT < l\big) \text{ then } v := 4 \text{ fi}; \text{ t } := 0; \{x' = v,\ t' = 1,\ t \leq T\} \big)^* @\text{invariant}(l \leq x \leq r \land v^2 = 16) \Big](l \leq x \leq r)$$

As an alternative solution, we could let the moving point choose a velocity that fits its current distance to the boundary. We also get rid of the requirement to cross the interval center to make our arithmetic a bit easier.

$$v \geq 0 \land l \leq x \leq r \land l < r \land T > 0 \rightarrow \Big[\big(\text{ if } (v \geq 0 \land x + vT > r) \text{ then } v := \tfrac{l-x}{T} \text{ fi}; \text{ if } (v \leq 0 \land x + vT < l) \text{ then } v := \tfrac{r-x}{T} \text{ fi}; \text{ t } := 0; \{x' = v,\ t' = 1,\ t \leq T\} \big)^* @\text{invariant}(l \leq x \leq r) \Big](l \leq x \leq r)$$

Up to now our models used only deterministic choices. For proving safety it is often beneficial to allow more flexibility with nondeterministic choice.

$$v \geq 0 \land l \leq x \leq r \land l < r \land T > 0 \rightarrow \Big[\big(\big(?v \geq 0 \land x + vT > r;\ v := \tfrac{l-x}{T}\big) \cup \big(?v \leq 0 \land x + vT < l;\ v := \tfrac{r-x}{T}\big); \text{ t } := 0; \{x' = v,\ t' = 1,\ t \leq T\} \big)^* @\text{invariant}(l \leq x \leq r) \Big](l \leq x \leq r)$$

7. **Quiz**

Suppose you have an event-triggered architecture as in the following $\mathsf{d}\mathcal{L}$ formula:

$$k \geq 0 \land x \leq max \rightarrow \Big[\big(\big((p := 0;\ (v := 0 \cup v := 1)) \cup (?x \leq max;\ p := 1;\ v := 0)\big); \{x' = (p - v)k,\ 0 \leq x \leq max\} \big)^* @\text{invariant}(x \leq max) \Big](x \leq max)$$

Fix the event-driven architecture, then turn the model into a time-triggered system.