**Recitation 3: Common Modeling Errors and Soundness**
**15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems**
**Notes by: Brandon Bohrer**
**Edits by: Yong Kiam Tan, (later) Katherine Cordwell (kcordwel@cs.cmu.edu), and Aditi Kabra**

# 1 Announcements

- Veribot is due Tuesday, 9/21.
- Recitation notes are posted at:
  `https://lfcps.org/course/lfcps.html`
- An archive of recitation code is also posted at the same page (if relevant). For example, for this recitation, all of the models and proofs used is in recitation03.kyx. You can load the entire archive into KeYmaera X by simply clicking `New Model` and selecting the relevant .kyx file. This should add all of the models in the archive to your database.

# 2 Motivation and Learning Objectives

In the lectures this week, we started looking at how to do safety proofs for hybrid programs in dL. Assuming that you have completed a proof, does that mean that the CPS (which you modeled with a hybrid program) is definitely safe? Unfortunately, that is not the case. Several possible issues remain, for example:

1. The reasoning principles you used in the proof could have been incorrect or unsound.
2. You might have made mistakes in the proof.
3. Your hybrid program might not accurately model the CPS.

Some of these issues can be resolved. To ensure that the reasoning principles behind all of the proof steps are correct, we can prove that the underlying axioms/proof rules are *sound*. To minimize the chances of making a mistake in your proof, KeYmaera X can be used to check your proofs. By design, KeYmaera X relies on a small trusted core of less than 2000 lines of code, making it a highly trustworthy tool.

The third issue, however, is just as important and not as easy to resolve. Simpler, more abstract models are easier to prove correct, but they might differ significantly from the real CPS that is in operation. Even if your model is sufficiently expressive, it may be the case that the CPS implementation itself is buggy.[1]

We will not be able to answer this difficult issue all at once. In this recitation, we will focus at the top and try to identify some common modeling issues with hybrid programs that makes them poor abstractions of your CPS.

[1] In a later lecture, you might hear about some of the work in our lab on correctly compiling a hybrid program model into an actual, verified implementation.

# 3 Empty Transition Relations

Let us start by recalling an exercise from the last recitation. The following formulas are:

1. $[\{x' = 1 \,\&\, x \leq 5\}]x \leq 5$ valid.
2. $\langle\{x' = 1 \,\&\, x \leq 5\}\rangle 0 = 0$ satisfiable.

From an initial state $x_0 = \omega(x)$ where $x_0 \leq 5$, both formulas are clearly true. This should be intuitive, but can be seen, e.g., by considering the general solution to the differential equation: $\varphi(t)(x) = x_0 + t$ which is defined on the maximum interval $t \in [0, 5 - x_0]$.

The issue is what happens when $x_0 > 5$? In this case, *no solution* exists that satisfies $\varphi \models \{x' = 1\} \wedge x \leq 5$. We say that the program has *no transitions* from such an initial state, or using the semantics, there exists no $\nu$ such that $(\omega, \nu) \in [\![\{x' = 1 \,\&\, x \leq 5\}]\!]$.

The box modality formula is true in such initial states $\omega$, because *all* of the zero transitions from $\omega$ lead to a safe state. Conversely, the diamond modality formula is false in initial state $\omega$ because it asks for *some* transition from $\omega$ but there are no transitions at all, not even one!

The following exercise tests your understanding of this subtlety.

**Exercise 1:**
Let $\alpha$ be an arbitrary hybrid program, $P$ be an arbitrary formula. Write a program $\beta$ that makes the following formula valid: $[\alpha;\beta]P$
Answer: $?\mathit{false}$

Notice that we now have a way of proving *any* hybrid program safe for *any* postcondition. We just need to add $;?\mathit{false}$ behind the program! This brings us to the modelling issue: your new and improved hybrid program almost certainly does not accurately model a CPS because it does not even have any transitions.

# 4 Common Modeling Bugs

Having an empty transition relation is but one way that hybrid program models can go wrong. In this section, we shall go through a few common issues.

## 4.1 Tests, Domain Constraints, and Hybrid Programs

Besides testing for *false* in hybrid programs, other kinds of tests could also lead to bad models. For example, let us consider the following four hybrid programs:

$$\alpha \equiv \{x' = v, v' = a \,\&\, v \geq 0\}$$
$$\beta \equiv \{x' = v, v' = a \,\&\, v \geq 0\}; ?v = 0$$
$$\gamma \equiv t := 0; \{x' = v, v' = a, t' = 1 \,\&\, t \leq T\}$$
$$\delta \equiv t := 0; \{x' = v, v' = a, t' = 1 \,\&\, t \leq T\}; ?(t = T)$$

**Exercise 2:**
Give an intuitive reading of what each of the programs mean. Think about when you might use these programs to model, e.g., a car braking and some of the issues that could arise.
Answer:

$\alpha$ runs the motion equations, as long as $v \geq 0$. Typically used to model braking to a stop when $a < 0$.

$\beta$ runs the motion equations, as long as $v \geq 0$ and additionally ensures that we have really come to a stop at the end.

$\gamma$ runs the motion equations, with a clock $t$ subject to the ODE evolving for at most $T$ time.

$\delta$ runs the motion equations, with a clock $t$. Ensures that the ODEs have run for exactly duration $T$.

The programs can be grouped into two: $\alpha, \gamma$ retain the potential nondeterministic behavior of running the ODEs, while $\beta, \delta$ remove this nondeterminism by discarding all runs except the ones that satisfy the test at the end.

Let us start with the pair $\alpha, \beta$. What does it mean to prove a safety property $[\alpha]P$ versus $[\beta]P$?

In particular, if $P \overset{\text{def}}{\equiv} v = 0$, then the latter formula is valid.

The model and the relevant proof are in rec3beta1.

This is usually a **major red flag** that you did not need part of your hybrid program at all to prove a property!

The issue, as mentioned earlier, is that instead of proving that the safety postcondition $P$ at all times along the runs of the ODE, we have now only proved it when $v = 0$.

That is not the only problem though. Suppose we "forgot" that we were supposed to be braking, then let us consider the formula:

$$a = 1 \land v > 0 \rightarrow [\beta]x \neq 5$$

**Exercise 3:**
Is this formula valid/satisfiable/unsatisfiable, and why?
Answer: The answer is again, yes, this formula is valid! In fact, we could have changed the postcondition to any arbitrary $P$ and it would still be valid because there are no runs of $\beta$ under the given assumptions (why?).
**Note: The model we studied in class and the relevant proof are in rec3beta2. Here, we simply noted that KeYmaera X completed the proof after solving the differential equation, but that the GV(1) trick would not work.**

Contrast this with:

$$a = 1 \land v > 0 \rightarrow [\alpha]x \neq 5$$

This time, the formula is not valid because $x \neq 5$ is required to hold at all times along the solution.

A very similar situation occurs with the time-triggered programs $\gamma, \delta$. These programs explicitly include a differential equation for the time variable. In $\gamma$, we are still allowing nondeterminism in the choice of duration for running the differential equation.

Why might this be reasonable? Think about $\gamma$ as part of a model for a periodic controller that wakes up, makes a control decision, then goes to sleep. We might know, a priori, that it must sleep for *at most* $T$ time, but only very rarely in the real world do you know that it will sleep for *exactly* $T$ time, which is what $\delta$ requires.

Our notion of "all runs" of a program, as expressed by the box modality, quantifies over all of the final states that can be reached by running a program. This is subtly different from the notion of "along all paths" of a program, which would not only quantify over final states, but also all of the intermediate states reached by the program (although for ODEs these notions align).

For this course, we work with the box modality, since it can already encode a surprising amount of useful path-like properties.

There are more advanced logics[2] that do allow quantifications over "all paths" of a program, which you are encouraged to explore if you are interested.

## 4.2 Operator Precedence

Although operator precedence allow us to save on parentheses and braces when writing down formulas/programs of dL, they can sometimes cause you to accidentally prove a formula that you did not mean to prove!

An example from Lab 0:

$$\forall x \forall y \forall z \; x < y \land y < z \rightarrow x < z$$

This formula proves fine with QE in KeYmaera X, but it does not express transitivity over the reals. That is because the operator precedence rules of dL dictate the following reading of the formula:

$$\Big( \big( \forall x \forall y \forall z \; x < y \big) \land y < z \Big) \rightarrow x < z$$

This formula is also valid, but only because the left of the implication is vacuously false.

The model and the relevant proof are in rec3trans.

If you ever run into issues with operator precedence, remember the cheatsheet from Assignment 1 or look it up in the textbook.

## 4.3 Overly Conservative Controllers

Another way we can fail to accurately model a CPS is to design a controller that is far too *conservative*. For example, if we have a car that never accelerates from rest will certainly never crash, but it would also not be very useful.

[2]For example, look up Differential Temporal Dynamic Logic.

For Lab 1 (and most of the early labs), this will indeed be the case with your controllers because the only way you can choose acceleration is to set it to some fixed initial value with a discrete assignment $a := e$. As we have argued earlier for time-triggered programs, however, it is unlikely that the acceleration can be set to *exactly* the value of $e$. We will see how to work around this later in the course.

On the other hand, conservative controllers could be easier to prove correct. Indeed, it is often easier to start with a simpler, more conservative controller before making it progressively more complicated.

### 4.4 Division by Zero

Another tricky issue is the use of division in models. Recall that the term syntax of dL does not actually allow you to write down divisions. In practice, divisions turn out to be useful for writing down models, and so KeYmaera X does allow you to write them down.

The key rule here is to **ensure that you never divide by zero**. This includes symbolic terms, so for example if you write $\frac{x}{y}$, you must ensure that $y$ can never take value 0 in the context where the fraction appears.

## 5 Modeling Problem

We modeled the floodgates of a Dam.

1. The reservoir has a cross section area of $A$ (so when the water level falls by height $\Delta H$ the volume of water in the reservoir has decreased by $A.H$.

2. The dam gates have a cross section area $a$. When you open them the water flows out at a rate of $a.v$ where $v$ is the velocity at which the water is flowing out.

3. As per Bernoulli's principle, the water flows at a velocity of $\sqrt{2gh}$ where $h$ is the current height of the water in the reservoir.

4. Water may flow in to the dam from the source river at a rate of at most `maxFlow`.

5. You are only allowed to take action to open or close the gates every $T$ seconds.

6. Suppose that we have a global counter for time, $t$, and that the flow of the river at any point of time is given by $r(t)$. `maxFlow` $\geq r(t) \geq 0$.

7. At all times, you must keep the height of the water above `hmin` and below `hmax`.

We want to model the flow in the dam and also write a controller that decides whether or not to shut the gates. We want to prove the system safe by deciding on a reasonable safety condition. In the process we will need to identify reasonable assumptions that we can make about the variables of the system so that we can prove safety.

### 5.1 Rate of change of height when gates are open

Find an expression for $\frac{dh}{dt}$.
Answer: When the gates are open, $\frac{r(t)-a\sqrt{2gh}}{A}$. When closed, $\frac{r(t)}{A}$.

### 5.2 What is the smallest height such that you can keep the gates open for time $T$ and still be certain the the water will remain above `hMin`?

You can assume $h' = \frac{r(t)-a\sqrt{2gh}}{A}$.
Answer: $(\sqrt{hMin} + aT\sqrt{2gh}2A)^2$

### 5.3 What is the greatest height the you can allow the water to be at and still leave the gates closed for time $T$, knowing the water level won't exceed `hMax`?

Answer: $hMax - (T.maxFlow)/A$

We used the solutions to these problems to write a model (Floodgates in the uploaded KeYmaera X file).

We used the first part – the expression for $h'$ – to describe the physical evolution of the water level in the reservoir. The result from the second part informed our controller when it was okay to close open the floodgates. Then we used the condition from the third part to determine when the controller could close the floodgates.

We carefully inspected our domain constraints and test clauses to understand whether we were making any errors by getting stuck or discarding states that we did not intend to. Finally, we went over the assumptions on the structure of the dam and on the value of constants, that we needed to make. We reasoned about the physical implications of the assumptions, making sure that they were consistent with what we were trying to model.

We started a proof, playing around with some of the rules and tools in KeYmaera X.

## 6 Soundness

In previous recitations, we have defined the syntax and semantics of dL. The *axiomatics* of dL are what we use to reason about hybrid programs.

The axioms we encounter will look just like formulas of dL, except they could contain schematic variables, for example in the choice axiom, $\alpha, \beta, P$ are schematic:

$$([\cup]) \quad [\alpha \cup \beta]P \leftrightarrow [\alpha]P \wedge [\beta]P$$

We say that an axiom is *sound* iff all of its instances (i.e., with the schematic variables replaced with concrete terms/formulas/programs) are valid.

We use equivalence axioms $P \leftrightarrow Q$ heavily in proofs.

Informally, one way we prove a formula $P$ is to use a chain of equivalence axioms:

$$P \leftrightarrow P_1 \leftrightarrow P_2 \leftrightarrow \cdots \leftrightarrow P_n \leftrightarrow Q$$

The $P_i$'s are (usually) successfully simpler formulas, until $Q$ is a sufficiently simple formula that we can trivially tell whether it is true or false. In this course, and in KeYmaera X, such a "sufficiently simple" formula will typically be a formula of $\text{FOL}_{\mathbb{R}}$, whose validity is decidable. If $Q$ is a valid formula, then the chain of implications and equivalences above (along with the soundness of axioms) yields the validity of $P$.

**Exercise 4:**
Why was that last sentence "If $Q$ is ... validity of $P$" okay?

## 6.1 Proving Axioms Sound Semantically

Let us step through an example of how an axiom can be proved sound using the semantics.

Consider the [∪] axiom. The axiom is of the form $\phi_1 \leftrightarrow \phi_2$, which says that $\phi_1$ is true in a given state $\omega$ if and only if $\phi_2$ is also true in that state. Formally, we need to show: $\omega \in [\![\phi_1]\!]$ iff $\omega \in [\![\phi_2]\!]$ for an arbitrary state $\omega$.

At first glance, this is slightly onerous since we have to show both directions of the implication:

$$\text{if } \omega \in [\![\phi_1]\!] \text{ then } \omega \in [\![\phi_2]\!]$$
$$\text{if } \omega \in [\![\phi_2]\!] \text{ then } \omega \in [\![\phi_1]\!]$$

Fortunately, for most of the equivalence axioms that you will encounter in this course, the proofs are relatively straightforward and can be done with a sequence of iff ( $\lbrace\rbrace$ ) steps.

You will get lots of practice with proving axioms in the assignment, but let us walk through a proof of the [∪] axiom.

Let $\omega$ be an arbitrary state, then:

$$\begin{aligned}
\omega \in [\![[\alpha \cup \beta]P]\!] &\lbrace\rbrace \nu \in [\![P]\!] \text{ for all } (\omega,\nu) \in [\![\alpha \cup \beta]\!] \\
&\lbrace\rbrace \nu \in [\![P]\!] \text{ for all } (\omega,\nu) \in [\![\alpha]\!] \cup [\![\beta]\!] \\
&\lbrace\rbrace \nu \in [\![P]\!] \text{ for all } \big((\omega,\nu) \in [\![\alpha]\!] \text{ or } (\omega,\nu) \in [\![\beta]\!]\big) \\
&\lbrace\rbrace \nu \in [\![P]\!] \text{ for all } (\omega,\nu) \in [\![\alpha]\!] \\
&\qquad \text{and } \nu \in [\![P]\!] \text{ for all } (\omega,\nu) \in [\![\beta]\!] \\
&\lbrace\rbrace \omega \in [\![[\alpha]P]\!] \text{ and } \omega \in [\![[\beta]P]\!] \\
&\lbrace\rbrace \omega \in [\![[\alpha]P \wedge [\beta]P]\!]
\end{aligned}$$

Notice that the proof using the semantics was fairly long and tedious. The nice thing about axioms is that they only need to be proved once and for all for any applicable instances. From now on, we are justified in using the [∪] axiom in syntactic proofs freely without drilling all the way into the semantics.

We can even use this to justify further *derived axioms*. For example, here is a triple choice axiom:

$$([\cup^3]) \quad [(\alpha \cup \beta) \cup (\gamma \cup \delta)]P \leftrightarrow [\alpha]P \wedge [\beta]P \wedge [\gamma]P \wedge [\delta]P$$

Instead of expanding the semantics (which would give us the correct answer), we can justify this by applying the axiom $[\cup]$ 3 times:

$$[(\alpha \cup \beta) \cup (\gamma \cup \delta)]P \leftrightarrow [\alpha \cup \beta]P \wedge [\gamma \cup \delta]P$$

$$[\alpha \cup \beta]P \wedge [\gamma \cup \delta]P \leftrightarrow ([\alpha]P \wedge [\beta]P) \wedge [\gamma \cup \delta]P$$

$$([\alpha]P \wedge [\beta]P) \wedge [\gamma \cup \delta]P \leftrightarrow ([\alpha]P \wedge [\beta]P) \wedge ([\gamma]P \wedge [\delta]P)$$

**Exercise 5:**
What is wrong with the above proof?
Answer: Technically, we also need to make use of the fact that $\wedge$ is associative! Notice also, that such a proof from the axioms does not really have a proper structure, and can get much messier once we work with multiple axioms. Next, we will see a way of structuring dL proofs using the sequent calculus.

## 6.2 Sequent Calculus

The sequent calculus for dL gives a structured way of writing down proofs. Indeed, this is the way proofs are managed and displayed in KeYmaera X. It should also be the way that you write down fully formal proofs on paper.

A sequent has the form (where $\Gamma, \Delta$ are lists of formulas):

$$\Gamma \vdash \Delta$$

The sequent above should be understood as the formula: $\bigwedge_{P \in \Gamma} P \rightarrow \bigvee_{Q \in \Delta} Q$, i.e., the sequent is valid iff the corresponding formula is valid.

In class, we developed a basic sequent calculus that we will use for dL. Most of the propositional proof rules are fairly straightforward, for example:

$$(\wedge\mathrm{R}) \quad \frac{\Gamma \vdash P, \Delta \qquad \Gamma \vdash Q, \Delta}{\Gamma \vdash P \wedge Q, \Delta}$$

Let us explore the more subtle rules for quantification:

$$(\forall\mathrm{R}) \quad \frac{\Gamma \vdash P(y), \Delta}{\Gamma \vdash \forall x\, P(x), \Delta} \quad (y \notin \Gamma, \Delta, \forall x\, P(x))$$

To prove that $P(x)$ is true for all $x$, we introduce a new, fresh variable $y$, and prove $P(y)$. The side condition is important here: let us see what can go wrong if we left out the freshness condition $\Gamma$:

$$\dfrac{\dfrac{*}{x = 0 \vdash x = 0}\,\mathbb{R}}{x = 0 \vdash \forall x\,(x = 0)}\,\forall\mathrm{R}$$

The conclusion of this derivation is not valid, but the broken $\forall$R allowed us to prove it!

**Exercise 6:**
Come up with a way to derive an conclusion that is not valid if we forgot to include $\forall x\, P(x)$ in the freshness condition.
Answer:

$$\dfrac{\dfrac{*}{\vdash y * y \geq 0}\,\mathbb{R}}{\vdash \forall x\,(x * y \geq 0)}\,\forall\mathrm{R}$$

In contrast, the left rule for universal quantification does not have any restriction

$$(\forall\mathrm{L}) \quad \frac{\Gamma, P(e) \vdash \Delta}{\Gamma, \forall x\, P(x) \vdash \Delta} \quad (\text{arbitrary term } e)$$

The $\forall$L rule does not need a side condition. In fact, the whole point of being to instantiate the for all quantifier with an arbitrary term $e$ is so that you can gain some assumptions about the variables that are already in the context $\Gamma, \Delta$.

Now that we have seen the rules for universal quantification, here is a quick trick (at least, classically) for understanding the corresponding rules for existential quantifiers.

Suppose we have the equivalence axiom:

$$(\exists_{\leftrightarrow}) \quad \exists x\, P \leftrightarrow \neg\forall x\, \neg P$$

Then, recalling the contextual equivalence and negation rules from lecture, it is easy to *derive* the rules for existential quantifiers:

$$(\exists\mathrm{L}) \quad \frac{\Gamma, P(y) \vdash \Delta}{\Gamma, \exists x\, P(x) \vdash \Delta} \quad (y \notin \Gamma, \Delta, \exists x\, P(x)) \qquad (\exists\mathrm{R}) \quad \frac{\Gamma \vdash P(e), \Delta}{\Gamma \vdash \exists x\, P(x), \Delta} \quad (\text{arbitrary term } e)$$

Derivation for $\exists$L:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, P(y) \vdash \Delta}{\Gamma \vdash \neg P(y), \Delta}\,\neg\mathrm{R}}{\Gamma \vdash \forall x\, \neg P(x), \Delta}\,\forall\mathrm{R}}{\Gamma, \neg\forall x\, \neg P(x) \vdash \Delta}\,\neg\mathrm{L}}{\Gamma, \exists x\, P(x) \vdash \Delta}\,\exists_{\leftrightarrow}$$

Notice that the $\forall$R step required a freshness assumption on $y$. Fortunately, the side condition for $\exists$L precisely says that $y$ satisfies these freshness assumptions.

**Exercise 7:**
Derive the $\exists$R rule.

Answer: Very similar to the derivation of $\exists$L.

As mentioned in lecture, the contextual equivalence rule is used often enough that we will simply omit it and write the axiom name (e.g. $\exists_{\leftrightarrow}$) directly. We can also use it to give a more structured derivation of $[\cup^3]$, although the proof is not super interesting:

For example, we technically applied $[\cup]$ three times in the first step below.

$$\dfrac{\dfrac{\textcircled{1} \qquad \textcircled{2}}{\vdash ([\alpha]P \wedge [\beta]P) \wedge ([\gamma]P \wedge [\delta]P) \leftrightarrow [\alpha]P \wedge [\beta]P \wedge [\gamma]P \wedge [\delta]P}\leftrightarrow\text{R}}{\vdash [(\alpha \cup \beta) \cup (\gamma \cup \delta)]P \leftrightarrow [\alpha]P \wedge [\beta]P \wedge [\gamma]P \wedge [\delta]P}[\cup]$$

After using $\leftrightarrow$R we get two premises, one for each direction of the biimplication. We shall only do $\textcircled{1}$ in detail since $\textcircled{2}$ is similar.

$$\dfrac{\dfrac{*}{[\alpha]P, [\beta]P, [\gamma]P, [\delta]P \vdash [\alpha]P \wedge [\beta]P \wedge [\gamma]P \wedge [\delta]P}\wedge\text{R,id}}{([\alpha]P \wedge [\beta]P) \wedge ([\gamma]P \wedge [\delta]P) \vdash [\alpha]P \wedge [\beta]P \wedge [\gamma]P \wedge [\delta]P}\wedge\text{L}$$

Besides the usual sequent calculus rules for first-order logic connectives, we will also need one additional rule for real arithmetic:

$$(\mathbb{R}) \quad \dfrac{*}{\Gamma \vdash \Delta} \quad (\text{ if } \bigwedge_{P \in \Gamma} P \rightarrow \bigvee_{Q \in \Delta} Q \text{ is valid in FOL}_{\mathbb{R}} \text{ })$$

Fortunately, the theory of $\text{FOL}_{\mathbb{R}}$is decidable, i.e., there is an algorithm that, when given a formula of $\text{FOL}_{\mathbb{R}}$, returns whether it is true or false. In KeYmaera X, this is implemented by asking validity in an external tool like Mathematica (or Z3).