15-424/15-624 Recitation 8
Proof intuitions and KeYmaera specifics

1. **A diversion: what if there's no sequent?**

   Take the proof rule for $[\cup]$.

   $$[\cup]\ \frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi}$$

   Where did that darned sequent ($\vdash$) go? Do we need it? What does it mean if it's not where?! The answer to these questions *and more* in the next paragraph, so keep reading!

   If there's no sequent, then the understanding is that the rule can be applied to either the antecedent or the succedent of a sequent! Notice how the proof will end up looking different depending on which side it is.

   In the antecedent, we'll get to use $\wedge_L$ and simply add both sides of the conjunction to the assumptions. The proof doesn't branch. In the succedent, we now have to prove *two* things, meaning the proof will branch.

2. **A simpler unwind rule**

   How can we deal with $\langle \alpha^* \rangle \phi$? Of course, with a proof rule, but which one! There's a very general one in the lecture notes, but your TAs are enormous procrastinators, and prefer simpler rules.

   The meaning of the formula is clear enough: is there an execution of α^* such that ϕ holds at the end. In other words, can we find a number of iterations n of the program α which leads us to stop at a state where ϕ holds?

   $$\langle {}^{*^n} \rangle\ \frac{\langle \alpha^n \rangle \phi}{\langle \alpha^* \rangle \phi}$$

   This is actually a family of proof rules! There's one for each $n \in \mathbb{N}$. If we can get away with simpler rules by using these "families" instead of a single rule, why shouldn't we?

   Well, we dare you to try to prove a non-trivial property using this rule! Can you figure out the exact n for which this will work? When dealing with complex systems, the clear answer to that is going to be a flat "no". In other words, we're simplifying the rule itself, but unfortunately we are giving the prover (whether that's you or KeYmaera) the burden of figuring out the hard choice. And that's not very helpful, which is why the other rule is used.

   Now, because we procrastinated and came up with a new proof rule, that comes back and bites us in the ***. We have to prove its soundness!

Let ν be any state. By assumption $\nu \models \langle\alpha^n\rangle\phi$, for some $n \in \mathbb{N}_0$ and thus there is μ s.t. $(\nu, \mu) \in \rho(\alpha^n)$ and $\mu \models \phi$.

We must show that $\nu \models \langle\alpha\rangle\phi$, which holds iff there is $m \in \mathbb{N}_0$ s.t. $(\nu, \omega) \in \rho(\alpha^m)$ and $\omega \models \phi$. We just take $m = n$ and $\mu = \omega$.

3. **Rule application order**

Very often, proofs will have very similar branches. To reduce our workload, it's useful to avoid branching if we can. Take the following proof.

$$\dfrac{\dfrac{\dfrac{x > y \vdash \langle\alpha\rangle\phi, \langle\beta\rangle\phi}{x > y \vdash \langle\alpha\rangle \vee \langle\beta\rangle\phi}\vee_R}{x > y \vdash \langle\alpha \cup \beta\rangle\phi}\langle\cup\rangle \qquad \dfrac{\dfrac{\dfrac{y < 0 \vdash \langle\alpha\rangle\phi, \langle\beta\rangle\phi}{y < 0 \vdash \langle\alpha\rangle \vee \langle\beta\rangle\phi}\vee_R}{y < 0 \vdash \langle\alpha \cup \beta\rangle\phi}\langle\cup\rangle}{x > y \vee y < 0 \vdash \langle\alpha \cup \beta\rangle\phi}\vee_L$$

Well, that's no good! Two of those branches are essentially the same! It's also very likely that the proof would continue to be replicated on both branches, sometimes to the point where only the QE arithmetic at the end is different. The extra work happened because we decided to branch earlier rather than later. As long as you have non-branching rules that don't require any user interaction, you should do those!

$$\dfrac{\dfrac{\dfrac{x > y \vdash \langle\alpha\rangle\phi, \langle\beta\rangle\phi \qquad y < 0 \vdash \langle\alpha\rangle\phi, \langle\beta\rangle\phi}{x > y \vee y < 0 \vdash \langle\alpha\rangle\phi, \langle\beta\rangle\phi}\vee_r}{x > y \vee y < 0 \vdash \langle\alpha\rangle \vee \langle\beta\rangle\phi}\langle\cup\rangle}{x > y \vee y < 0 \vdash \langle\alpha \cup \beta\rangle\phi}\vee_l$$

Ah, much better. So remember, procrastinate your branching!

4. **Delayed assignment**

The substitution rule for assignment is really nice and simple.

$$[:=]\ \dfrac{\phi^{\theta}_{x}}{[x := \theta]\phi}$$

But in reality, the substitution rule is a *hipster*. It only applies to programs before they are cool, such as when you add loops and ODEs. Take the following very simple example bouncing ball example, where the ball has just hit the ground and is going to bounce.

$$[:=]\ \dfrac{[x' = -cv, v' = A]\phi}{[v := -cv][x' = v, v' = A]\phi}$$

In this instance, the property is mainstream and cool because it contains an ODE. So the substitution rule can't be applied, and the above proof is *unsound*. The problem that the assignment is only meaningful at time $t = 0$ of the ODE, but the substitution is making changes *throughout* the execution of the ODE.

A similar phenomenon can be seen in the following:

$$[:=]\ \frac{[(x := 4)^*]\phi}{[x := 3][(x := x + 1)^*]\phi}$$

This clearly changes the behaviour of the program in a fundamental way. The cool insight is that instead of applying the substitution immediately, we can *delay it*, much like KeYmaera does. After we get rid of the programs, it's then possible to perform the substitution!

$$\text{update simpl.}\ \frac{[x := x - cvt + \tfrac{1}{2}At^2, v := -cv + at]\phi}{\text{ODE solve}\ \dfrac{\{v := -cv\}[x := x + vt + \tfrac{1}{2}At^2, v := v + at]\phi}{[:=]\ \dfrac{\{v := -cv\}[x' = v, v' = A]\phi}{[v := -cv][x' = v, v' = A]\phi}}}$$

Notice that after applying ODE solve, all the occurrences of v and x refer to their *original* values, which is why it's become sound to perform the substitution. In loops and ODEs, an occurrence of a variable refers to their value at whatever iteration or time of the loop or ODE it refers to.

5. **ind loop vs ind loop (global)**

   There are two variants of loop rules that can be used in KeYmaera. One of them allows you to keep your original assumptions, the other one does not. They do so by using the $\forall^\alpha$ construct, which universally quantifies all of the *bound* variables of program α.

   What this essentially does is it "hides" the original value of the variable. Thus, it allows you to keep all the information that you had from before, since the relevant parts have been "hidden" through quantification.

   You are encouraged to look at the attached inv.key program, and prove it with the invariant $x > 0$, using both ind and ind (global) rules. You'll notice that, to keep things sound, one of them hides information about the non-relevant y, whereas as the other does not.

   When previous information about non-bound variables is hidden, you must include it in the invariant (since it doesn't change anyway). When it isn't, you have perhaps too much arithmetic lying around, and that might put a strain on QE. But you can always hide that information later.