Improving KeYmaera

Less clicking, more proving

Abstract

Our goal with this project was to implement some new convenience features for KeYmaera in order to streamline the process of proving cyber-physical systems. We found that KeYmaera 3 often required more work than necessary. While we had some ideas as to how to improve KeYmaera initially, these changed when we switched to working with the in-development KeYmaera 4. All of our improvements focus around a text-based interface we added to KeYmaera 4 which can be used as an alternative to the existing mouse-based interface. This interface can be much faster for experienced users.

We implemented a text-based interface which allows applying tactics to single formulas. It features autocompletion suggestions, the ability to repeatedly apply the same proof rule, and the ability to apply several tactics all together. These features allow for much more efficient interaction between the user and the KeYmaera 4 theorem prover.

Introduction

Using KeYmaera can be painful.

Having interacted with KeYmaera quite a bit, we came to the conclusion that it is often the case that proving the safety of a hybrid program can be a relatively straightforward endeavor, after one has discovered the correct invariants. Telling KeYmaera about said invariants can be the real challenge in the proof process. The interactive steps of a proof can be tedious and mechanical. Hiding formulas unrelated to the goal in order to use quantifier elimination, applying rules such as and-left repeatedly to large formulas, and repeatedly cutting in the same differential invariants for each branch of a control decision are particularly time consuming and repetitive steps one must take when interacting with KeYmaera. The process can be streamlined by simple methods such having text files listing all of the relevant invariants and formulas so that one must simply copy and paste them into the input dialog for logical cuts in KeYmaera 3. These kinds of repetitive steps motivated us to want to improve the user interaction with KeYmaera. We wanted to be able to give something as simple as the cut and paste algorithm to KeYmaera: cut in each differential invariant in the correct order and then quantifier elimination would verify the invariant. Users could be saved from countless hours of clicking and possible hand and wrist injuries, if only it were possible to give KeYmaera a list of invariants and the order in which to use them. Then more properties could be proved automatically. Thus by decreasing the amount of clicking we improve the amount of proving.

KeYmaera has undergone much revision during its life, and consequently its codebase is quite large. After meeting with Dr. Stefan Mitsch, we learned it would be much more worthwhile to implement our improvements in the upcoming version of KeYmaera which had a much smaller and updated codebase. KeYmaera 4 is presently under heavy development and as such not all of our suggested improvements can be implemented yet. We are however, able to improve the KeYmaera-user experience by adding a text based interface. Instead of right-clicking a formula, selecting a rule to apply and hoping one didn't accidentally select the incorrect formula or rule, one can now type into KeYmaera the desired sub formula and rule to use with our clear and concise KeYmaera 4-text syntax. Thus by reducing the amount of clicking and increasing the amount of typing, we accomplish our namesake of increasing the amount of proving.

Related Work

The interface for KeYmaera 3 allows the user to apply rules to a formula by first selecting the formula with the mouse. Certain rules however, can only be used once a formula is free of conjuncts and disjuncts. This is due to how formulas and rules are stored internally. This forces repeated use of the formula simplification rules such and-left and and-right .As mentioned earlier, it was our goal to ease the physical burden on the user caused by repeated clicking. We believe a user can more easily choose which rules to apply by typing them in rather than selecting them with a mouse. This type of user interface is similar to that of other theorem provers such as Coq and Isabelle. Neither of these theorem provers however is capable of dealing with continuous dynamics like KeYmaera is. Coq and Isabelle have been used mostly for verifying mathematical theorems and proofs, whereas KeYmaera has been used more in proving the safety of controllers for hybrid systems. KeYmaera is also capable of verifying mathematical proofs since it also uses a classical sequent calculus. All three systems have automated theorem proving tactics. Isabelle uses natural deduction rules along with classical logic rules such as proof by contradiction. It's proof syntax was designed to be similar to human proof syntax and so proofs can appear verbose. Coq syntax is similar Isabelle's but less similar to human proof syntax. Both systems display the proof status next to the user input.

KeYmaera 4 uses a sequent calculus type of display with goals on the right of the sequent and assumptions on the left of the sequent. Whenever the proof tree must split into two branches, subgoal branches are created and only the current branch is displayed. The user can then continue working on the current branch or switch to one of the branches created. More details on how the proof tree changes when rules are applied can be found in the implementation section.

Implementation

Since our proposed contributions to KeYmaera 4 could be implemented entirely in the frontend our first step towards adding these features was to learn the codebase. Since KeYmaera 4 has a web-based interface the code we were concerned with is a combination of HTML and Javascript. Like most major Javascript projects today KeYmaera makes use of a variety of Javascript frameworks to simplify its codebase. Most relevant to us KeYmaera uses AngularJS, a framework which helps structure code for modern web applications, and jQuery, the most popular Javascript framework which makes writing Javascript much more convenient. We first had to become familiar with these frameworks in order to know how to begin on our improvements.

Before we could write any code though we had to come up with a notation we could use to apply proof rules textually. Since at the time we started the only rules which could be applied referred to just one formula at a time, our focus was on being able to apply just these proof rules through the textual interface. Thus, we started out with this notation:

<formula identifier>/<tactic identifier>

We decided on this because it contained exactly as much information was necessary. Initially we were using the internal formula and tactic names. This meant that to apply the *and left* rule from differential logic to the first part of the antecedent you would type:

ante:0/dl.and-left

Parsing this text into it's the formula identifier ("ante:0") and the tactic identifier ("dl.and-left") was simple, however it took some work to actually use this information to apply the right rule in the right place. First, whenever the current proof changed we would examine every formula in the updated proof and load up all of the tactics that could be applied to it. When the user hit enter in the text field we added we would then simply check if the formula identifier corresponds to a formula in the current proof and then if the tactic identifier refers to a tactic we could apply to this formula. If both of these were the case we then sent a request to apply the tactic the same way the existing click-based interface did.

With the existing formula and tactic identifiers this interface is awfully verbose and probably not much faster than just using the mouse. To simplify the formula identifiers we implemented a suggestion from André Platzer to refer to formulas in the antecedent with negative numbers and formulas in the succedent with positive numbers. For the tactic identifier we just created shorter aliases for several common tactics. We also changed the separator from a forward slash to a colon for cosmetic reasons. Applying the same rule as before now looks like:

-1:and

Note that we're being completely unambiguous with this tactic name as it can only refer to *and left* in this context. Implementing these changes was reasonably simple. When we read in the new tactic identifier it is easily converted into the internal tactic name. Implementing the short tactic names required slightly more work as these aliases were not attached to the tactics when we initially load them. Instead, as we're loading the tactics initially we check if there is an associated short name and add it to the tactic we just loaded. Then, when we're checking if a tactic identifier matches a known tactic, in addition to comparing it to all of the full tactic names (which are still valid choices), we compare it to all of the known short names.

Our next step in making this user interface even more intuitive was to add autocompletion suggestions to the text field. This drastically improves the usability of our text interface because it allows our users to see the names of all applicable tactics and thus using this interface no longer requires memorization of the various tactic names. Since jQuery was already in-use in KeYmaera we decided the simplest way to add autocompletion would be through the jQuery autocompletion widget. However, adding this required installing an extension of jQuery called jQuery-UI. This was done using Bower, a Javascript package manager already being used on KeYmaera.

Once we were able to use the jQuery autocomplete widget we had to actually figure out which suggestions to present. We choose to provide a function which looks at the currently entered text and makes suggestions based directly on that. This allowed us to implement two levels of suggestions. First, if the user hasn't fully specified a tactic id we restrict our suggestions to formula ids which could match what the user has currently entered. Only after a full formula id and colon are entered do we suggest possible tactics that could be applied. We decided to suggest all possible entries so users get suggested both the shorter aliases for tactics as well as the full names for them. By breaking it up into two different levels of suggestions we aim to keep the autocomplete suggestions more focused and helpful.

By the time we'd completed this an interesting new feature had been added to KeYmaera 4: saturated tactics. This is another name for the iterated tactics we originally suggested in our proposal. This is a way to repeatedly apply the same tactic to a formula such as applying *and left* to break A ^ B ^ C ^ D into A, B, C, and D. Once we saw these saturated tactics were available in the latest version of KeYmaera 4 we investigated how they were applied. In order to add these tactics into our notation we simply added an asterisk after any tactic id to make it refer to the iterated version. Thus the example used above would become:

-1:and*

To avoid polluting the autocomplete suggestions we added the suggestion to saturate a tactic as a third layer which only appears after a full tactic id has been entered. This way we don't have essentially two copies of every tactic id in the suggestion box.

After implementing this Stefan Mitsch, one of the primarily developers of KeYmaera 4, suggested we add support for entering multiple textual commands into the interface. The first step to implementing this was simply allowing our text box to have multiple lines. Next, we had to expand our syntax to support applying multiple tactics. We decided that similar to Javascript we would allow commands to be separated by either new lines or semicolons. However, because of how KeYmaera 4 is setup it isn't

feasible to have autocomplete suggestions for anything after the first statement since applying a rule can change what's in both the antecedent and the succedent in addition to which rules can be applied.

To implement this we first replaced all new line characters with semicolons. Then, we used the semicolons to split the tactics into individuals pieces. Next, we tried to apply the first piece and we removed it from the text entry. The hard part came in applying the remaining tactics. Since applying tactics relies on us knowing which tactics can and cannot be applied we have to wait for our tactics object to update. This is somewhat difficult because of the asynchronous nature with which we request which rules can be applied to which formula. In order to call our new code once that tactic finished updating we used a modern design pattern found in AngularJS called promises. This allows us to cleanly wait for several different events to finish before running our code. Once the tactics object finished updating we simply attempted to apply the next queued tactic until they are all gone. Initially this solution seemed to have some problems, however after updating our codebase to the latest state of KeYmaera these problems disappeared.

What We Learned

While working on KeYmaera 4 we learned quite a bit about the design of modern web applications. Working with AngularJS was especially insightful as it exposed us to a variety of real-world design patterns we had not encountered previously. We were able to use dependency injection, promises, and the model-view-controller design to implement our improvements to the KeYmaera interface. This experience will be enormously useful should we go on to work with web applications in the future.

In addition to what we learned about writing web code, we also learned about the design process while deciding how our interface should work. The design is just as important as the actual implementation because even if a poorly designed interface is implemented perfectly it will do little to help users. We found that the design of our notation was an iterative process. Piece by piece we improved what we had until we reached our final result. However, we know there is room to make it even better if we had more time.

Next Steps

While our changes have improved the usability of KeYmaera we have several ideas we haven't had time to implement. The most obvious omission from our textual interface is the ability to apply tactics which operate on multiple formulas at once. One example of such a tactic is *axiom close* which takes a formula from the

antecedent and a formula from the succedent. Our proposed notation for applying tactic such as this is:

<formula1 id>,<formula2 id>,...:<tactic id>

For rules such as *axiom close* where there is exactly one formula in the antecedent and one formula in the succedent the ordering of these formulas will not matter since there is no ambiguity. However, in some cases like *apply formula* the order has to matter. In this case the first formula will be applied to the second formula similar to dragging the first formula onto the second formula in KeYmaera 3. Autocomplete could continue to work for each of the individual formulas as it before. That is, it would only suggest new formulas until the colon was entered.

The next feature we would implement would be support for the interactive tactics. These are tactics like *loop induction* and *cut* which require more than just formulas as input. Our proposed notation for these tactics is as follows:

<formula1 id>,<formula2 id>,...:<tactic id>:<user input1>,<user input2>,...

Thus, attempting to apply *loop induction* to the first formula in the succedent with the invariant x>y would be:

1:loop-ind:x>y

For these tactics we would not be able to provide autocompletion suggestions for the user inputs since these items require intuition which KeYmaera lacks.

Finally, one minor improvement which could be added to our textual interface would be allowing the step and auto tactics to be used. The step tactic works either applied to an individual formula or applied to the whole sequent while the auto tactic only works while applied to the sequent.

With these additions our textual interface would have feature parity with the click-based interface which currently exists for KeYmaera 4.

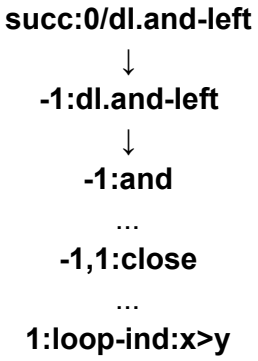
Summary

Increasing the speed and ease of use for KeYmaera users was the goal of this project. To this end, we implemented a text-based interface for KeYmaera 4 which can be much faster to use than the mouse-based interface inherited from previous KeYmaera versions. The interface allows for the selection of formulas based on an indexing scheme for the different parts of the sequent at hand. Rule application is then done by typing the name of the desired rule. The text interface provides the user with autocompletion suggestions of applicable rules and uses our syntax of shortened rule names. We implemented what we believe is an intuitive and concise syntax for indicating formulas and rules. The syntax we use is short and unambiguous thanks to the formula naming scheme which indexes sub formulas in the antecedent with

negative integers and sub formulas in the succedent with positive integers. The text interface also allows the user to indicate that a rule should be used multiple times and can be given multiple rules to apply sequentially. If the re-indexing of all sub formulas after a rule application is known, a proof that previously required user interaction at each step, could require only an initial input of the sequence of rules to apply. This can undoubtedly allow experienced KeYmaera users to complete proofs more quickly. We believe this addition to KeYmaera 4 will help remove much of the mechanical user effort that the previous mouse interface could require.

Appendix

Evolution of Syntax:



Example usage:

Step by step:

$\vdash (A \wedge B \wedge B \rightarrow C) \rightarrow C$
input text: 1:imply
$(A \wedge B \wedge B \rightarrow C) \vdash C$
input text: -1:and*
$A, B, B \rightarrow C \vdash C$
input text: -1:imply
$A,B \vdash C,B$ $A,B,C \vdash C$

Multi-step:

$\vdash (A \wedge B \wedge B \rightarrow C) \rightarrow C$
input text: 1:imply;-1:and*;-1:imply
$A,B \vdash C,B$ $A,B,C \vdash C$