Tutorials for RODIN

October 26, 2007

Introduction

This tutorial should provide the user with a tour through the most important functionalities of RODIN, so that he gets a understanding of how the program works. The tutorial is divided into 5 sections:

In the first section, a very simple project is created from scratch. The essential steps in working with components are illustrated here.

The second section provides an example that shows how events of different machines can be connected together. It also gives an introduction on working with the prover

After section 1 and 2, the user should have developed a feel of the basic windows of RODIN, and when they are needed. He might want to combine the two default perspectives into one. Section 3 explains how this can be done. This section may be omitted by users who feel comfortable switching between two perspectives.

Section 4 shows how to use apply reasoning on models in the prover.

Section 5 shows a proof in a mathematical setting, and then provides a few examples, on which the user can work on by himself. In these proofs, everything except the basic rewrite rules has to be done by the user.

Files

For this tutorial, you will be needing 4 example files. They are:

- `Celebrity.zip`, which will be needed in section 2.
- `Doors.zip`, which contains the model that is used in section 4.
- `Closure.zip`. This is the model on which you perform proofs in section 5.

- `Galois.zip`. This model is not really needed, but serves as an explanation to why the proof in section 5 works.

1 A First Example: The Birthday Book

To begin with, let us start with a simple model of a "birthday book", similar to the one in the Z Notation Reference Manual[1]. This little book is a simple tool that can be used to keep birthdays of different people. All we can do with it is writing people's birthdays into it. So, the initial model only has one event. We create the model as follows:

1.1 The Event-B Perspective

1. Start RODIN. You should be directed automatically to the Event-B perspective. If this does not happen, you will have to change to this perspective manually (Eclipse help describes how this is done).

2. Create a new Project as described in section 1.3 of the Manual. Name the project `BirthdayBook`.

3. Create a new Context Component named `BirthdayBook_C0` in the Project you created. This can be done similarly to creating the new project. (See also Section 1.8 of the Manual)

4. In this model, we are not interested in the specific structure of the date. All we want this context to contain are two carrier sets, one for dates and one for people. We will call them `DATE` and `PERSON`. You can add them to the context as described in section 2.1 of the manual. Now either press Ctrl+S or click on the save Icon to save the context.

5. Now, we need to create a Machine Component. Proceed as you did to create the Context Component, just choose "Machine" this time. Give the Machine Component the name `BirthdayBook_0`. Once the component is created, a window with the machine's dependencies appears. Add `BirthdayBook_C0` to the seen contexts so that you can access the carrier sets.

6. We need a variable that reflects the contents of the book. We call it "birthday'. The following information on the variable can be entered into the New Variable Wizard (Manual Section 3.2.1), which can be accessed by an icon on the tool bar. Since every person has at most one entry in the birthday book, but not every person has an entry in it, "birthday" should be a partial function from `PERSON` to `DATE`. At initialization, we want the book to be empty. (The symbol for partial functions can be written by typing in `+->` . Section 8 of the Manual contains all on how to write Event-B symbols in ASCII.)

[1]The Z Notation Reference Manual can be found at http://spivey.oriel.ox.ac.uk/mike/zrm/index.html

7. Last, we create the event. Open the New Event Wizard (as seen in Section 3.5.1 of the Manual) on the top tool bar. Name the new Event `AddBirthday`. It has two parameters, `p` and `d`, where `p` is a `PERSON` and `d` is a `DATE` (enter these as guards). As action, write $birthday := birthday \cup \{p \mapsto d\}$ (Remember, Section 8 of the manual explains how to write Event-B symbols). Now, save the machine. If you have done everything correctly, the type checker should not return anything to the problems window at the bottom of the screen. It appears that we have successfully created a model of the birthday book. But have we?

1.2 The Proving Perspective

Now, we switch to the proving perspective. You see several windows on the screen:

- The Proof Obligation Explorer (Section 5 of the Manual): Here you can browse through proof obligations. As you can see, `BirthdayBook_C0` has no proof obligation and `BirthdayBook_0` has two. The A in the icons of the obligations means that the automatic prover attempted both obligations. The green icon next to the first proof obligation indicates that it already has been proved, and the red icon next to the second proof obligation tells that the proof is not yet completed. Proof obligations can have three colors, red, green and blue. The blue color means that the proof has been reviewed, meaning that it has been discharged without proof by the user. Click on the second prove obligation to display the proof. This fills a couple of other windows.

- The Proof Tree (Section 6.2 of the Manual): Here you see a tree of the proof that you have done so far and your current position in it. By clicking in the tree, you can navigate inside the proof. Currently, you have not started with the proof yet, so there is no new place to move to.

- The Proof Control (Manual, section 6.4): This is where you perform interactive proofs.

- The Selected Hypothesis window (Manual, section 6.7): The hypothesis that are currently being used for the proof are displayed here. You can add hypothesis into it from the Search Hypothesis window and from the Cached Hypothesis window(See section 6.4 in the Manual on how to open these windows).

- The goal window (Manual, section 6.3): This window shows what needs to be proved at the current position inside the proof tree. Currently we need to show that birthday is still a partial function from `PERSON` to `DATE` if it is extended by an entry.

There is no way to prove the goal if a birthday is already entered into the book for a certain person. So, our event needs an additional guard that restricts `p` to people for whom there is no entry yet in the book. This can be done in the proving perspective. Just switch to the machine and add the additional guard (`p` $\notin$ `dom(birthday)`) using

the editor (On adding guards: Manual, section 3.5.2). If you save the document now, you will see that the auto-prover can conclude.

Congratulations! You have built your first model with RODIN.

2 The Celebrity Problem

The next model that you will work with is the so-called celebrity problem. In the setting for this problem, we have a "knows" relation. This relation is defined so that

- no one knows himself,
- the celebrity knows nobody, but
- everybody knows the celebrity.

The goal is to find the celebrity. The provided development, once completed, yields a linear-time algorithm for the problem.

2.1 Modeling

1. Make sure that you have no existing Project named `Celebrity`. If you do, then rename it (Section 1.7 in the Manual). Then import the archive file `Celebrity.zip`. For this, choose "Import..." in the File menu, and then select "Existing Projects into Workspace". Then, select the according archive file. (Read more on importing projects in section 1.6 of the Manual)

2. The tool takes a few seconds to extract and load all the files. Once it is done, it shows that there are a few problems with this project. In the first part of this section, our goal is to fix these problems and conclude the proofs.

3. First of all, we take a look at the error. It states that an event called "celebrity" is not refined. Double-click on the error in the Problem Window to open the `Celebrity_1` machine. If you look at its events (by pressing the "Events" tab), you can see that it actually does have a "celebrity" event. The problem is that it is not declared as a refinement. In order to do so, right-click on the event and select "New Refine Event". This declares that the event is a refinement of an event with the same name in the abstract machine. As this is the case here, we can now save the project and the error disappears. (If the event in the abstract machine had a different name, you would have to edit it. For more on this issue, see Section 3.9.1 of the Manual).

4. Next, we deal with the warnings. The three remaining warnings state that witnesses are missing. In any abstract event that uses parameters, if the concrete event has no parameter with the same name, the tool needs a witness so that it notices what value the parameter should take. Witnesses are also needed for variables

that have a nondeterministic assignment in an abstract event and do not appear in the concrete model. (See also Section 3.9.4 of the Manual) To create the witness, double-click on the warning to open the concrete model (here `Celebrity_2`). Then, right-click on the "celebrity" event and select "New Witness".

5. An empty witness has been created, which we need to fill. Its name will have to be x if we want it to be a witness for the parameter x. Next, for the content. If we switch between the two machines (either by pressing Ctrl+F6 or by clicking on their tabs), we see that the abstract event has the assignment $r := x$, while the concrete one has the assignment $r := b$. So, $x = b$ is the witness. Edit the Details and save the file. One warning will disappear, two to go.

6. Try completing the other two witnesses on your own. A hint: Both witnesses are simple equalities, and both can be found by comparing the third guard of the abstract event with the second guard of the concrete one. Remember to give the witness the name of the variable it stands for. If you completed this step correctly, there should be no warning, info or error left on the Problems window.

2.2 Proving

1. All we have to do now is prove. Switch to Proving Perspective. Browsing around in the Obligation Explorer (Section 5 of the Manual shows you how), you can see that the auto-prover did quite a good job. If you have chosen the witnesses correctly, all except for five proofs already should be completed. Except for the last one of them, all of them could be proved with a different external prover, but in order to learn a few new techniques, we will prove them with the p0 prover.

2. Let's start with the proof in `Celebrity_0`. Select the proof by clicking on it. What you need to prove is that P is not empty. Enter P in the proof control and open the Search Hypothesis Window. Like this, you get all hypothesis that have P in them. We need to find one that works toward the goal. $c \in P$ does that, so add it to the selected hypothesis (Section 6.7 of the manual explains how) and click on p0 in the Proof Control. The proof succeeds.

3. Next, we look at `celebrity/act1/SIM` of `Celebrity_1`. Here we need to prove $x = c$. You have no hypothesis about c selected, so look for them, like you did for hypothesis with P in the last proof. This time, you can see that $c \in Q$ is all you need to conclude the proof. So add it to the selected hypothesis and the proof will succeed using p0.

4. `remove_1/inv2/INV`, is a little more complex. In order to prove the statement, it suffices to prove $x \neq c$. so type this into the proof control and press the Add Hypothesis button (ah). Now, press p0 until it does not get you any further. Now, try selecting the right hypothesis by yourself in order to complete the proof. If you cannot find it, you may also just select all hypotheses.

5. In order to move to the next proof obligation, you may also use the Next Undischarged PO button of the Proof Control. The next proof can be solved the same way as the last one. You just need to add at least two hypotheses this time.

6. In the proof in `Celebrity_2`, you have to fill in an existential quantifier. First, look in the list of hypothesis if you find any variable that is in P, and select that hypothesis. Then, instantiate b' and R' correctly (For instance, if you want to instantiate b' with c, then $P\backslash\{c\}$ is a good choice for R') by typing the instantiations in the Goal Window and then clicking on the red existential quantifier. Now, all open branches of the proof tree can be proved with p0. After this, we have completed all the proofs, and the model is ready for use.

3 Customizing a perspective suitable for RODIN

So far, you needed two different perspectives to work with RODIN. But really, it is possible to work with only one perspective. In this section, we try to customize a perspective so that we do not need any other. If you have experience with customizing Eclipse perspectives, you may only want to read the next paragraph which contains a few thoughts about a good perspective for RODIN.

As a start, we should think about what we want the perspective to look like. The proving perspective already is pretty nice. We just could use little bit more editing space and the windows of the Event-B perspective. To create more space, we could move all windows that currently are on both sides of the editing area onto one side, as they never really need to be used simultaneously. For even a bit more space, we could dock all of these windows onto the so-called Fast View Bar, so that they disappear when they are not needed. Like that, there should be enough space to even work split-screen with different components, for example, we could have an abstract machine on one side of the editing surface, and the concrete machine on the other.

Most of the perspective editing is simply drag and drop. First of all, you need to find the Fast View Bar. Usually, it is at the bottom end of the Eclipse window. But it also can be on the side or hidden inside the Shortcut Bar. For our purposes, it probably is best to have it on the right side of the screen. Place it there by dragging it with the mouse. Now, add some items to it. To do that, press the New Fast View button on the bar. It might be useful to leave the Goal, the Problems and the Proof Control window at the bottom of the screen, as you may want them to stay open while editing. A good choice for the Fast view may be:

- Project Explorer,
- Obligation Explorer,
- Search Hypothesis,

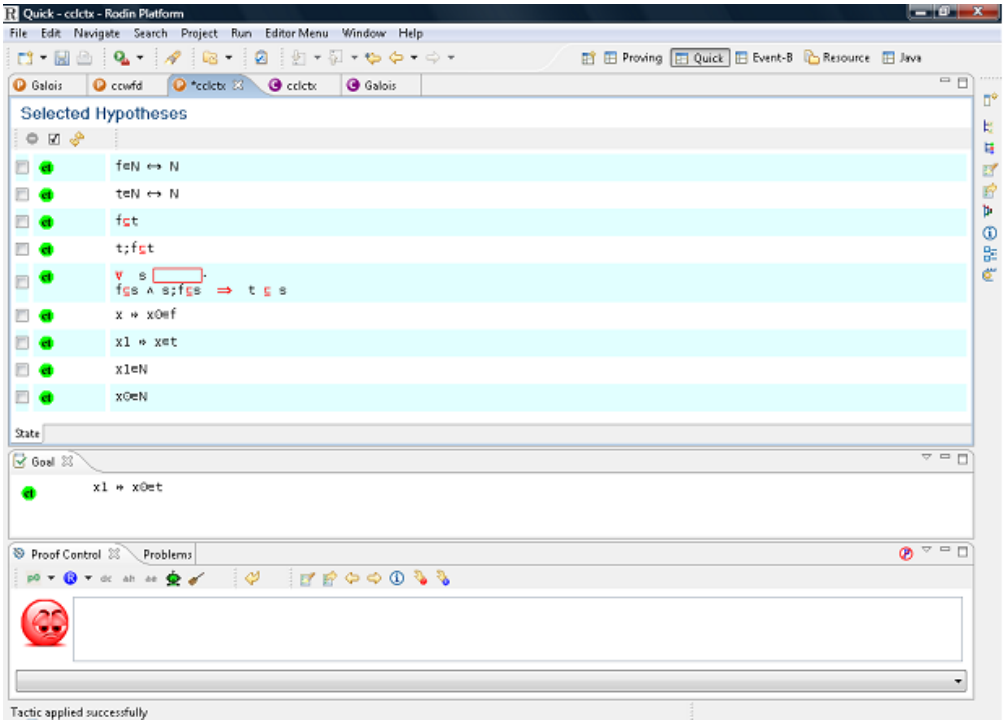

Figure 1: Our self-made Quick perspective

- Cache Hypothesis,
- Proof Tree,
- Proof Information
- Progress Window.

All of the windows that you cannot create directly when clicking on the New Fast View Bar can be found in Others/General. Once you are done, the window should look like in Figure 1. Click on "Save Perspective As..." in the Window menu to save the perspective.

4 Location Access Controller

In this section, we will take a closer look at a few more complex proofs. For this, we will use the model of the Location Access Controller, which is saved in the `doors.zip` project archive. Explanations on the model can be found in the chapter entitled " Location Access Controller" of the Event-B book. Our task is to develop the proofs for deadlock freeness of the initial model and of the first refinement.

Import `doors.zip`. Looking through it, you will see that everything already has been proved. This is true, however, RODIN does not do any deadlock freeness proof yet, so you will have to add them yourself.

4.1 Initial Model

First, let us look at the initial model, constisting of `doors_ctx1` and `doors_0`. There are two carrier sets in the model, one for people and one for locations. Then, there is a location called *outside* and a relation *aut* which reflects where people are allowed to go. Everyone is allowed to *outside*. The model only has one event, which changes the location of a person. So, proof of deadlock freeness means proving, that someone can always change room.

1. Add a new theorem to `doors_0` called "DLF" and change the predicate so that it is the disjunction of all guards (since there is only one guard here, it would be $\exists p, l \cdot (p \mapsto l \in aut \land sit(p) \neq l)$.

2. Save the machine. You will see that the autoprover fails to prove the theorem (`DLF/THM`). Let us analyze whether this is an inconsistency in the model. In order to succeed with the proof, we need a tuple $p \mapsto l$ that is in aut, but not in sit. Searching the hypotheses, we find AXM4 of `doors_ctx1`, which states that there is a room l, where everyone is permitted to. So, for every person p in P, $p \mapsto l$ and $p \mapsto outside$ is in aut. Since these are different, at least one of them cannot be in the function sit. Now, all we would need to prove is that P is nonempty. This holds, as carrier sets always are nonempty, but is a bit hard to derive. Add the hypothesis $P \neq \varnothing$ using the ah button. Then, click on the Auto Prover button (The button with a robot on it). Other provers do not work here. After successfully adding the hypothesis, we can conclude the proof as follows:

3. Add AXM4 (the one with the existential quantifier) to the selected hypothesis. You see that it automatically is instantiated as l. Next, click on the "¬" of $\neg\ P = \varnothing$ in the search hypothesis window. This creates a selected hypothesis $x \in P$. We can now instantiate p in the goal with x.

4. In order to instantiate l, we need a case distinction. Type $sit(x) = l$ this into the proof control and click on Case Distinction (dc) to look at the two cases $sit(x) = l$ and $sit(x) \neq l$. Before starting with the cases, the prover now wants you to prove that $x \in dom(sit)$. This can be done with p0, as sit is a total function. In the first case x is situated in l, so it cannot be in *outside*. So, you can instantiate l with *outside*. In order to prove that x is allowed to *outside*, you will need to select the hypothesis $P \times outside \subseteq aut$. Then you can finish this case with p0. If you look at the proof tree, you see that you now have reached the other branch of the case distinction. In this case, you can simply instantiate l with l, as x is not situated there. Finally, click on p0 to finish the proof.

4.2 First Refinement

Now we get to the a bit harder proof: The deadlock freeness proof for the first refinement. There is not much that has changed. The constant *com* has been added in order to describe which rooms are connected. Additionally, we have a constant *exit*, which will be explained later.

1. Open `door_1` and add a theorem called DLF stating $\exists q, m \cdot (q \mapsto m \in aut \wedge sit(q) \mapsto m \in com)$ to it, then save the file. Once again, the prover fails to prove deadlock freeness automatically.

2. At the beginning of the proof, there are no selected hypothesis at all. So we need to select a few. The old deadlock freeness theorem will be useful, AXM7 of `doors_ctx2` too. To begin with, we try to avoid using *exit*, as we want to keep things as simple as possible. But since *sit* and *aut* are inside the theorem, we also are likely to need $sit \subseteq aut$, $sit \in P \rightarrow L$ and $aut \in P \leftrightarrow L$.

3. Once again, the prover automatically rewrites the existential quantifiers in the hypotheses. We now look at the proof. There is an easy case if $sit(p) = outside$. Solve it.

4. For the other case, we will need the notion of *exit*. The axioms about *exit* state that

   - (AXM 3) Every room except the outside has exactly one exit.
   - (AXM 4) An exit must be a room that communicates with the current one.
   - (AXM 5) A chain of exits leads to the outside without any cycles or infinite paths.
   - (AXM 6) Everyone allowed in a room is allowed to go through its exit.

   In our proof, we still need to show that anyone who is not outside can walk through a door. For this, AXM 5 is useless, so we add all hypothesis containing *exit* except for AXM 5. Now we only need to instantiate q and m correctly and concluding the proof should not be too hard. For q, the choice p is obvious. But it is not quite as easy for m. Expressed in language, m must be the room behind the exit door of the room that p is currently in. Try translating this into set theory. But do not worry if you get it wrong. You can still go back in the proof by righ-clicking at the desired point in the proof tree and choosing prune.

5 Mathematical proofs

By now, you should know how to create and edit models, and how to do simple proofs with help of the predicate prover. In this section, we will look at more complex proofs in mathematical settings. For this, it is advisable for you to have some knowledge of the theory, but you can also see this as a pure proving exercise. First, we try to perform a

proof without the predicate prover in order to get familiar with all the tools. Then, you can try proving three further theorems by yourself.

5.1 First proof on Transitive Closure

1. Open the "Closure" project. it is a simple mathematical model, in which f is a binary relation. The axiom defines t so that it is the transitive closure of f. You will have to prove that $t; t \subseteq t$.

2. For that, we have to instantiate the s of $\forall s \cdot (f \subseteq s \wedge s; f \subseteq s \Rightarrow t \subseteq s)$ with $(N \times N) \setminus (t \sim; ((N \times N) \setminus t))$. This instantiation is a Galois transform. The project `Galois` shows the properties of galois transforms. Simply said, these transforms are a sort of inverse for the subset relation between relations. In the proof, this is useful, as we can easily derive the goal from the implication (Because of the main property of the Galois transform, $t \subseteq s$ is equivalent to $t; t \subseteq t$). The first part of the condition of the axiom also becomes easy to prove, as $f \subseteq s$ is equivalent to $t; f \subseteq t$. However, the second part of the condition, $s; f \subseteq s$ does not have such an easily provable equivalence. We will just have to believe that the instantiation also works here.

3. If you want to check that the instantiation really does work, you can use the predicate prover now and it will succeed. Afterwards, prune the predicate prover step so that you can do the proof yourself.

4. If the instantiation succeeded, you now should have a new, quite lengthy new hypothesis which has the form $p1 \wedge p2 \Rightarrow p3$, where $p1$, $p2$ and $p3$ are predicates. This hypothesis will help us to split the proof into three parts. First, we will prove $p1$, then $p2$, and last, we will derive our goal out of $p3$. To split the proof, click on the red arrow of the implication and choose "Apply Modus Ponens using this hypothesis". If you now look at the pending subgoals by using the "Next Pending Subgoal" button in the Proof Control, you will see that the proof has been split into the three described parts above.

5. We will start off proving the first subgoal $f \subseteq (N \times N) \setminus (t \sim; ((N \times N) \setminus t))$. First of all, we need to translate the statement a bit more towards predicate logic. Click on the subset symbol in the goal and choose the first option (Remove inclusion in goal). As described, this translates the inclusion into predicate logic. If you look at the proof tree, you will see that the prover automatically has done another two steps, removing the quantifier and then the implication in the goal.

6. Now, you will need to completely translate the goal into predicate logic. For this, click on any red symbol that appears in the proof goal until there is none left. Should you have any tilde operators or cross products in the selected hypothesis, then it will not hurt to translate them, too. To do that, click on the red "$\in$" next to then. In the end, your proof should look like in Figure 2.

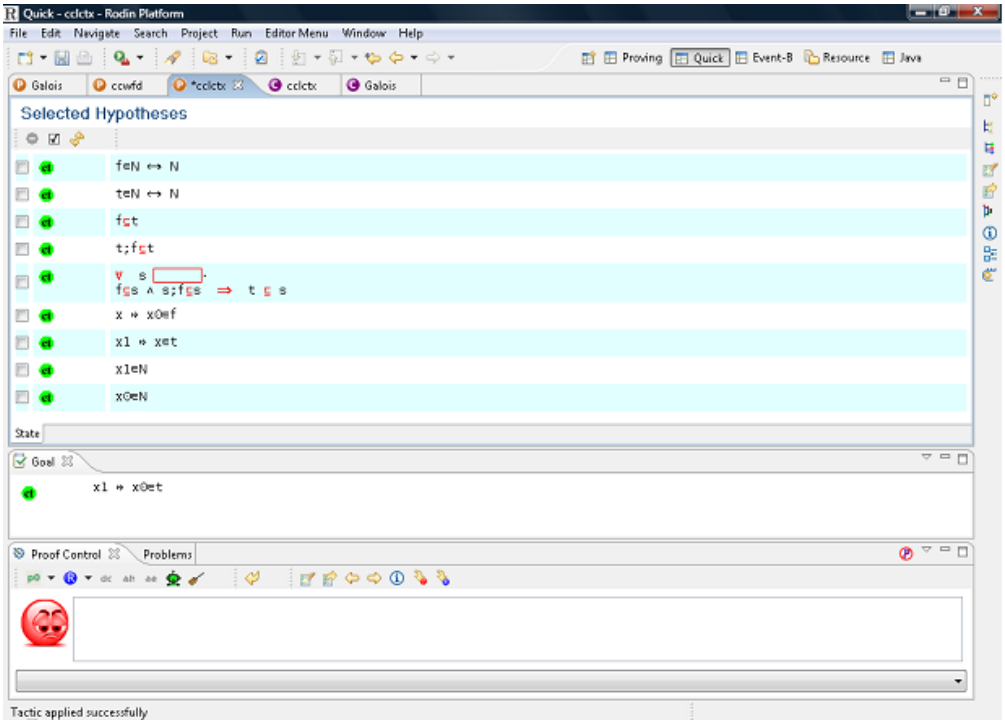

Figure 2: The proof after step 6

7. As the goal appears to be provable from $t; f \subseteq t$, $x \mapsto x0 \in f$ and $x1 \mapsto x \in t$, remove all the other hypotheses, if you like to keep your workbench tidy. This can be done by choosing the three hypotheses and then clicking the "Inverse selection" button first and then the "Remove hypotheses" button.

8. To prove the current goal, we first prove that $x1 \mapsto x0 \in (t; f)$. From that, $x1 \mapsto x0 \in t$ follows, because of $t; f \subseteq t$. Add the hypothesis $x1 \mapsto x0 \in (t; f)$ by entering it in the proof control and pressing the (ah) button. Once more, the proof will branch. For the first branch, click on the red "$\in$" symbol in the Proof Goal window to translate the goal into predicate calculus. You will then need to instantiate a variable, but the choice should be obvious if you look at the selected hypotheses. The second part of the proof can be solved by removing the "$\subseteq$" in $t; f \subseteq t$ and then instantiating correctly.

9. For the time being, we do not yet want to prove the second subgoal, but proceed with the last one. So review this subgoal by pressing the blue button in the proof control. The prover automatically skips to the next subgoal.

10. The proof of the third subgoal will only require the hypothesis $t \subseteq (N \times N) \setminus (t \sim ; ((N \times N) \setminus t))$. So you can remove all the others. Remove the inclusion in the hypothesis and in the goal in order to transform them towards predicate calculus. Then, click on the "$\in$" of the hypothesis $x \mapsto x0 \in t; t$ in order to introduce $x1$.

You can now instantiate x with $x1$ and $x0$ with $x0$ in the hypothesis with the quantifiers, as is done in the proof of the theorem in the `Galois` project.

11. After translating the hypothesis into predicate logic by pressing all "$\in$", you get a rule that states $\neg(\exists x \cdot x1 \mapsto x \in t \sim \wedge\ x \mapsto x0 \in (N \times N) \setminus t)$. Click on the "$\neg$" to get a hypothesis with a universal quantifier.

12. Now, you will want to get this hypothesis into a better appliable form. By using rewrites (clicking on the red symbols), you should be able to transform it into $\forall x \cdot x \mapsto x1 \in t \wedge x \in N \wedge x1 \in N \Rightarrow x \mapsto x0 \in t$. Try to get the hypothesis into the right form yourself. Should you completely mess something up, you can still go a few steps back in the proof tree by clicking on the "Backtrack from the current node" button in the proof control.

13. The instantiation for x should now be obvious. After the instantiation, you get a new hypothesis. If you Apply the Modus Ponens on this, the smiley in the Proof Control will turn blue. This means that all the non-reviewed goals have been proven. So, you will now have to prove the second subgoal. Click on "Select the next review subgoal" to get there.

14. The proof of the second subgoal begins similar to the others. Remove the inclusion in the goal and simplify the goal as far as possible. Next, translate the new selected hypotheses into predicate logic by clicking on the "$\in$" symbols in the selected hypothesis until there is none left. There should be one selected hypothesis that starts with a negation. Remove the negation in it. Now we get a hypothesis beginning with a universal quantifier. It should look like this, just the variable names might differ: $\forall x0 \cdot\ \neg\ (x \mapsto x0 \in t \sim \wedge\ x0 \mapsto x2 \in (N \times N) \setminus t)$.

15. The hypothesis stated above can also be brought into a more easily comprehendible and directly appliable form. In the end, it should state something like $\forall x0 \cdot (x0 \mapsto x \in t \wedge x0 \in N \wedge x2 \in N \Rightarrow x0 \mapsto x2 \in t$, where names may vary. Get the hypothesis into the right form yourself using rewrites (similar to step 12). Now, the instantiation for $x0$ should be obvious, as there is only one variable that maps to x (In Figure 3, $x1$).

16. The rest of the subproof is the same to steps 7 and 8, except for the variable names. So you can either repeat these steps or try to copy the proof. To copy the proof, right-click on the part that you need on the proof tree where you added the hypothesis and choose copy. Then right-click on your current location and select paste. By all chance, the proof did not work due to the different names. Look in the proof tree for where the wrong name has been used and prune that part. You will have to redo that bit of the proof again.

5.2 Exercise: Theorem 2

In Theorem 2, you have to prove that $t = f\,\mathrm{union}(t; f)$. For this, you have to prove both directions ($t \subseteq f \cup (t; f)$ and $f \cup (t; f) \subseteq f$). Start with the second part, as it is much

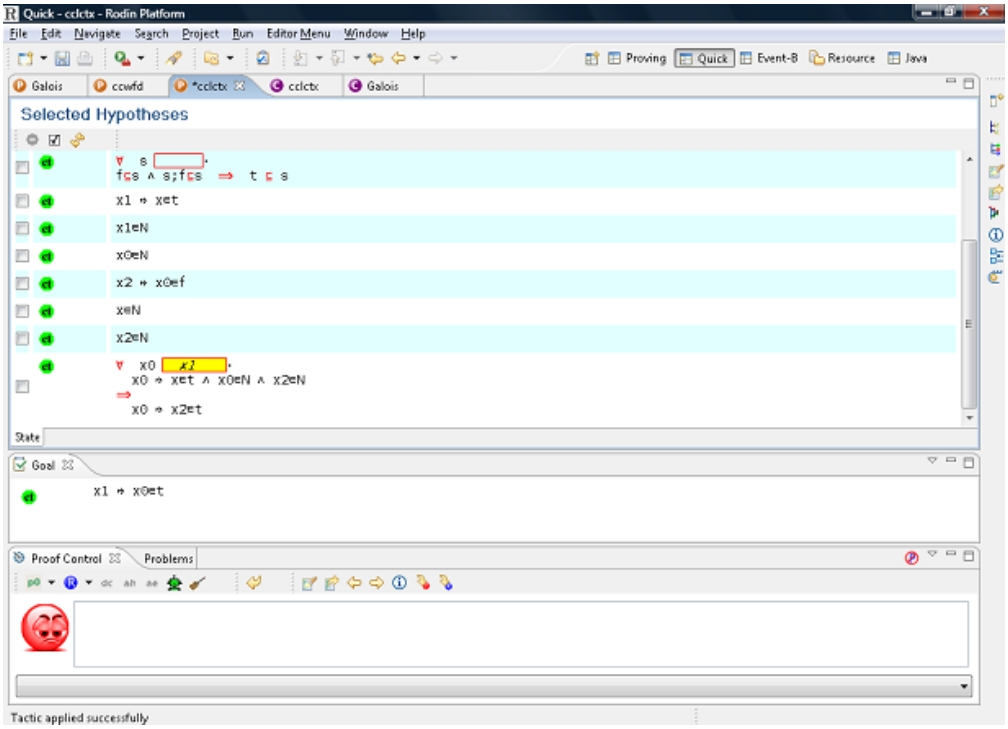

Figure 3: The proof at step 15

easier. The first part also should not be too hard, if you instantiate the hypothesis with the quantifier as early as possible.

5.3 Exercise: Theorem 3

Theorem 3 is very similar to Theorem 2. Proving also works in a very similar fashion, except that it takes a few steps more, as we do not have any axioms on $(f;t)$. Then, Theorem 1 usually is useful.

5.4 Exercise: Theorem 4

Theorem 4 states that $\forall b \cdot f[b] \subseteq b \Rightarrow t[b] \subseteq b$. Once again, you will need the right instantiation of a hypothesis to succeed with the proof at some stage. Here, $((N \setminus b) \times N) \cup (b \times b)$ will be a good instantiation. This proof is quite lengthy, as there are many proofs by cases that you will need to perform. A lot of the cases can be solved by successfully adding the negation of a selected hypothesis and thus creating a contradiction.