MARK SEIFTER: Hi, everyone. A few of you may know me from my tutorial, but if you don't I'm Mark Seifter I'm the senior TA here at 6034 and these Friday omega recitations are going to be a little bit different than any of the other things in the class.

All right, so, what is omega recitation. Omega recitation is in this lecture style room, but I hope to make it more like a recitation or a tutorial by, unfortunately for some of you, perhaps, calling on you guys to participate during the omega recitation, as we attempt to work with some of the material that Patrick has given us the big picture for in the lectures. And if you've gone to recitation so far you may received a little bit of enrichment into how the algorithms work.

Well, here in omega recitation we're going to figure out how you can work the algorithms. And that's very important, because in this class being able to work the algorithms by hand in some of these examples I put up here, or like Patrick sometimes does in lectures to teach you the algorithms, is a crucial way of demonstrating that you understand how the algorithms work, and we use it on a quiz.

The way we use it on the quiz is we-- well this is an old quiz problem. It's from last year. It tripped up a lot of people. So we're going to go over it today. I'm going to tell you all the tricks that tripped up a lot of people, in addition to emphasizing tricks that I've picked up over of the three years of TA-ing this course before, so that you will not fall prey to any of these tricks. Hopefully. I say hopefully, because some of these tricks I talked about omega recitation last year and people still got tricked by them. I guess maybe some people didn't come to omega recitation or they were just very tricky tricks.

So be sure to come to omega recitation. Pay attention to the tricks. Pay attention to how these things are solved. Ask questions if you're not sure. And when we get done with the hour together I'm hoping that everyone's going to do really well on the rules part of the first quiz. So without further ado, if there are no questions on that, which I don't think there probably will be, let's move on to the problem at hand.

As some of you may have noticed, it is a Harry Potter based problem. One of our TAs last year was a big Harry Potter fan.

So what we have here a series of rules and a series of assertions. Now if you happen to be real go getter and have already looked at all the past quizzes online, you may have already seen this problem. And if you really, really have attention to detail you may notice something a little bit different in the way I wrote it as compared to the way it was written online. In fact, you may, if you have a lot of times a detail, but you're not really in on Scheme or LISP, you may think the way I wrote it is much easier to understand. And in case you can't understand the other way, and in case one of the TAs decides to write it in a LISP-like way again, let me explain.

If you look at one of my rules, say if x is ambitious, and x is a squib, then x has a bad term. That's rule 0 up at the top. That's written in sort of what we call in fixed notation. It's easy, it's simple. All operators are in the order you would expect. 2 plus 3 equals 5.

There's another kind of notation that is sometimes used on previous quizzes. And by sometimes, good number of times, actually. I'm not sure why we still use it that way, because of the fact that the class has move on to Python, but we do, sometimes. And that's called prefix notation.

In prefix notation, the operator comes first. So for instance, plus 2 3 equals 5. The plus comes first. The function, then, comes first. In this case, the function is plus. So if we were writing this in prefix notation, in fact, the way I received it, when it said, "if and x is ambitious, x is a squib, then x has a bad term. The and went first and there's a parentheses which included everything that was scoped under the and. Does anyone have any questions about prefix notation in case it comes up?

Just remember, if things start being written in a really weird way with lots and lots and lots of parentheses, whatever's outside the parentheses, whether it be "plus" or "and" or "or" is an operator that acts on the things that are inside. However, we're not going to deal with that now, because I think the most important thing is to understand is the rules, and the less important thing is to understand prefix notation. And you can do a problem at home that's written in prefix notation to test yourself on that.

So what rules do we have here. We've got equals 0 through 5. For some reason they are labeled with P's. They are often labeled with either P's or R's. The secret on why they are

labelled with P's is that one year somebody labeled them with P's for like "proposition" or something like that. And then other TAs looked at that test and sometimes also labeled it with P's and it sort of down the line continued to be labeled with P's, and sometimes with R's.

So these P's are these six rules. The first rule is if x is ambitious, and x is a squib, then x has a bad term. So what's the deal with these question mark x's? The question mark before an x, or perhaps a y down here, indicate that there's a variable waiting to be bound. We're not assuming there's a Harry Potter character named x, and only if that question mark x, the mystery character, only if mystery x character is ambitious can that person possibly have a bad term.

What we're seeing is any character in the Harry Potter universe, or not the Harry Potter universe, maybe a rhinoceros, can fit into that x. So for instance, if a rhinoceros is ambitious, and a rhinoceros is a squib, then a rhinoceros has bad term. That rule saying, for any x this is true. And it's very important how we treat the question mark x, and how we bind question mark x, when we do both back chaining and forward chaining. I'll get back to that, because some people made some very, very small mistakes that really messed up a lot of their forward and backward chaining last year. It was actually very tricky. Jeremy, the TA who wrote this was very clever, and it makes it really great case study for you guys.

All right, so Rule 1, if x lives in Gryffindor tower then x is a protagonist. By the way, for conciseness I'm going to be using GT for Gryffindor tower when I write these in later on, and I'll use SD for Slytherin dungeon.

Speaking of which, Rule 2, if x lives in Slytherin dungeon, then x is a villain, x is ambitious. Why there are two things here? Well, after the "if" we have what we call antecedent, that's something that needs to be true in order for this rule to match. After the "then" we have what is called the consequent, and in this case there are two consequents. Anyone who lives in Slytherin dungeon is automatically a villain and also ambitious. So you can think of there being sort of an "and therefore" for the purposes of both of those assertions would be added to the knowledge base.

Rule 3, if x is the protagonist or x is a villain, and x is ambitious, then x studies a lot. By the way, the scope for this, just to be sure that we're clear, is this. So we need them to be a protagonist or villain and no matter what they have to be ambitious.

Rule 4, if x studies a lot, and x is a protagonist, x becomes Hermione's friend.

And Rule 5, if x snogs y, and x lives in Gryffindor tower and y lives in Slytherin dungeon, then x has a bad term.

So those are our six rules that we can use to understand Jeremy's world of the Harry Potter universe.

And we also start off with four assertions. Let me not underestimate the value of always looking to the assertions. It's one of my white star ideas that are up here on the board. Let me see. This. Perfect. Always check the assertions before using a rule. This really tripped people up last year, and you'll see why, because we're doing last year's problem.

Our four assertions that we start. With assertion 0, Millicent lives in Slytherin dungeon. Assertion 1, Millicent is ambitious. Assertion 1 is what tripped people up, so remember that Millicent is ambitious. Assertion 2 Seamus lives in Gryffindor tower. And assertion 3, Seamus snogs Millicent. So those those are our four assertions that we've already started with.

Now the two things we're going to have to do are backward chaining and forward chaining. Now when you guys learned these two backward chaining and forward chaining, raise your hand if thought forward chaining was harder than backward chaining. I knew it! I can prove whatever point I want because noone wants the raise their hand. I also think backward chaining is higher than forward chaining.

Raise your hand if you think backward chaining is higher than forward chaining? First of all, we have a good number of people. Second of all, since no one wants to raise their hand, I could just ask it the other way. That's a pro tip if you're ever with a large group of people. You can prove any point you want by asking the other direction. No one will raise their hand.

So I agree that backward chaining is higher than forward chaining. And I disagree with Patrick that we're going to get out early, so let's start with backward chaining first. So that we make sure that we spend the bulk of our time with it. The forward chaining, well, you just go through pretty methodically and add new rules. Backward chaining, you have to draw this crazy tree. There's a lot of places to get lost in the middle of the road.

So let's do some backward chaining. And to do that, I'll do it over here on the left side. So when we're doing backward chaining, we have to remember a few things that are written directly on the quiz. So you're not going to have to worry about that. But they're still pretty

important. So-- actually, I'll write it on here first. I'm going to read them off. So when working on a hypothesis, the backward chainer tries to find a matching assertion in the list of assertions first.

If no matching assertion is found, the backward chainer will try to find a rule with a matching consequence. A rule that has something in the then that can prove the assertion is trying to figure out. So for instance, if I was doing backward chaining on Seamus [? Snaus ?] Millicent, what happens? I'm immediately done because there's an assertion that's assertion three that says Seamus [? Snaus ?] Millicent. I proved it. I'm done. I'm happy. We can leave, we can go home.

Unfortunately, that's not what the quiz asks us to do. Now, let's say that instead, we were supposed to say, I don't know, Seamus is a protagonist. Well then, we'd wind up looking through here and we would look at the fact that rule one can prove that someone is a protagonist. And we'd curse and try to prove whatever is in the antecedent of rule one, which is, OK, well, does he live in Gryffinder Tower? Assertion two, he does. So that's some really quick.

And if that was too, fast don't worry. We're going to go step by step with the actual problem. But I just wanted to give two really easy problems, really quickly, how it's going to do it. Let's go step by step with the real problem. Let's keep in mind, backward chainer never adds new assertions to the list of assertions. And if you have a tiebreak, you always order based on rule order first, P0 through P5. And if the same rule matches with more than one thing in your list assertions, then you tiebreak based on the order of the assertions. Very important.

Disambiguation and tie-breaking are a big place to get messed up. So we're going to try to prove that Millicent becomes Hermione's friend. And so I'm going to abbreviate some of the things, but not for the very first line. So Millicent becomes Hermione's friend. We start drawing a goal tree.

Now you guys are going to learn exactly what these mean very soon, in fact, next lecture. But for now, trust me when I say these goal trees are depth first. And I'll explain what that means because some people have messed themselves up and spent more time than they needed to by treating the goal tree in a different way.

Now, Millicent becomes Hermione's friend. Let's pretend that we are the backward chainer.

We're trying to prove that this is true or disprove and say, well, it's definitely not true with what we have. So let's see. I'm now going to make you guys help. And people in the back think they won't be called on. So I often like to call on people in the back. What do you think is the first thing we should do? Very first thing?

Yes, look for a matching assertion. Excellent. Do we have a matching assertion, everyone? No, we don't. That would be the world's easiest quiz problem. We do not have a matching assertion. Great. So, since we don't have a matching assertion, what now?

AUDIENCE: We start to look at the rules.

MARK SEIFTER: That's right. And do you see any rule that could prove that Millicent becomes Hermione's friend?

AUDIENCE: P4.

MARK SEIFTER: That's right. You can see in P4, x, which can be anybody. Anyone is capable of being Hermione's friend. Great. So P4 is our rule of the hour. So we're going to use P4. And when we use P4 to prove that Millicent becomes Hermione's friend, we're going to have to add something or another to the goal tree. So let's see. What do we have to add to the goal tree?

AUDIENCE: We have to see if Millicent studies a lot [INAUDIBLE].

MARK SEIFTER: That's right. Does everyone see that? In order for her to become Hermione's friend, by rule four, we have to see if she studies a lot and is a protagonist. Question?

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: OK, that's a very good question. You're going to get screwed up if you look at the antecedent in backward chaining first. It's backwards partially for a reason. You need to look at the consequent. Now why is that?

Well, let's say there is a rule six that said if x becomes Hermione's friend, then Hermione feeds x Polyjuice Potion, or something like that. Will that rule help us in back chaining to figure out if Millicent becomes Hermione's friend?

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: Some people are shaking their head. But think about it, will that rule be able to prove it? No.

Now, if they do become Hermione's friend and we want to do some forward chaining, we'll figure out that they're going to transmute into some kind of other thing because of the Polyjuice Potion. But it's not going to help us do the one thing we want to back chain, which is to prove that thing on the top. So we actually need to look for something that has our current goal in its consequent. Then we add on the antecedents.

As people I've asked so far-- sorry, I don't know your names-- like Patrick, have correctly given me every time, which is excellent. So notice also that she didn't say x studies a lot needs to be added to the goal tree. She said Millicent studies a lot. Oh, another question. Great.

AUDIENCE: So obviously after putting rule 4 there, shouldn't we first check the assertions-- check on the assertions instead of our conditions [INAUDIBLE]

MARK SEIFTER: So the question is, once we put rule four in there, should we check to see if those antecedents of rule four are already in the assertions before checking other rules? The answer? You're not only correct, sir, you're exactly one step ahead. So I'm going to assume I called on you for the very next thing. That's exactly what we do once we draw it onto the tree. You're exactly on the way. Further question? Perfect.

All right, great. So again, we're putting up Millicent studies a lot. Millicent is a protagonist. We're not putting up x. Some people did that. And of course it's an n node as we heard. We need them both to be true or we're not going to be able to continue onward. So we have-- I'm going to use m for Millicent. Millicent studies a lot. Also over here, Millicent is a protagonist. And we've already heard that we need to search in the assertions before we go into any rules for what we're looking for next.

The question is, what are we looking for next and in what order? This is where that thing I told you about depth search is crucial. We are going to look here on the left node. And if it has any children, if we have to keep going down, we are not going to look here yet. There's a few reasons for that. One of which is that we're lazy. The evaluation, if you learned about it, is also-- if you learn about lazy evaluation, is also lazy. And if we can disprove this study's a lot branch, we don't have to do any work with the protagonist branch. We're done. We're out of here. And it's a loss.

So we're going to move down the study's a lot branch. And we already heard from the audience, we need to look to see if it's in the assertions. Everyone, is Millicent studies a lot in the assertions? No, it's not. Well then, now we're going to get to that question that I had before

about the antecedent or the consequent. Because study lies here in the antecedent of rule four, but we're not going to be able to use rule four here. In fact, that's how we got here in the first place.

So the question is-- let's see, the question is, do we have a rule in the consequent that matches Millicent studies a lot. Yeah, that's right, P3. That's right. So P3 would give us Millicent studies a lot. And so therefore what will we add to the goal tree?

AUDIENCE: We add both that is Millicent either a protagonist or a villain and [INAUDIBLE].

MARK SEIFTER: Yeah, she. Turns out to be a girl. So that's exactly right. We heard that we need to add-- we need to add an AND node, which also has an OR node at the bottom of it, because this is a little bit of a complicated rule. So we have an AND node. So we have an AND node. And the first thing on the AND node is Millicent is a protagonist or Millicent is a villain.

And the second thing on the AND node is Millicent is ambitious. Actually, that is the second thing on the AND node. I hope that's what I said. All right, the second thing on the AND node is Millicent is ambitious. So here's our handy little tree. Now this is where it's important, as I said, we're doing a depth first search.

We're going down along the left branch. So where do you think we're going to go next on this tree?

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: Not quite. That's a good guess. You didn't say we're trying to prove m is a protagonist on the right branch, which is a very common mistake. What do you do?

AUDIENCE: You should go down to see if there's a consequence for m is-- Millicent is a protagonist.

MARK SEIFTER: Which one of the two Millicent is a protagonist do we want to look at? Yeah, the farthest down left. That's right. So you guys will learn depth for search on Monday, at which point it will become much clearer what I'm saying. But yeah, you always follow the left branch as far down as you can. Then you take the right branch of that same branch that you followed.

Great, so we're going to try to find that Millicent is a protagonist. What do we do first? Everyone? Check assertion, yes. Is it in there? No, it's not. OK. So, therefore we're going to try to find a rule as we heard about whether Millicent is a protagonist. So, all the way in the back?

All the way in the back, is there a rule that matches?

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: That matches in its consequent that Millicent is a protagonist.

AUDIENCE: Um, there is a rule.

MARK SEIFTER: OK. Well, let's give it a try. Which one?

AUDIENCE: One.

MARK SEIFTER: Rule one, that's right. So therefore, we would have to add what to our goal tree?

AUDIENCE: Another [INAUDIBLE].

MARK SEIFTER: Right. And what's on that please?

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: Yeah, Millicent lives in Gryffindor Tower, perfect. So, our new thing that we're searching for is-- pretend this connects. Actually, I guess it does connect. We're looking for m lives in Gryffindor Tower. Great. And since it's depth first, that's where we go next. Great, we're on a roll.

So what do we do first? Do we have that in assertions? Some people say yes, but the answer is no. We have, in fact, Seamus living in Gryffindor Tower. Most of you said "no." You're right. The majority rules. We don't have any assertions.

However, do we have a rule with that in the consequent? No. So what do we do? People are saying different things that are all correct. Backtrack, you say. Go to the next. We can't prove it. All correct. Put a big x. This isn't true.

Now we'd look up. We're on an OR node. So we're not done yet because either of those can be true. So now we go back up, backtrack, Millicent is a villain. What do we do first? Check assertions. Is it in there? No, it's not. Don't worry, we're getting to one where it will be, where about 40%-- or 30% to 40% of the class lost points, very, very soon.

So, we see that Millicent is a villain. And it's not in the assertions. So therefore, is there any rule that has that in its consequent?

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: We're looking for Millicent is a villain.

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: Oh, you can't see it.

All right, I will move it down slightly. It got a little bit off kilter when I put in the parentheses around the "and." Hold on. There you go.

AUDIENCE: If [INAUDIBLE] is a protagonist or is a villain. [INAUDIBLE]

MARK SEIFTER: We're trying to prove that she's a villain, though. So do we have any with that in the consequent? [INAUDIBLE] anything that will prove that she's a villain if we fire off that rule. Want to give here a little bit of help?

AUDIENCE: P2. If she lives in Slytherin, then she's a villain [INAUDIBLE].

MARK SEIFTER: Ah, see that now? It had two, so it was a little bit harder.

Now some people ask, what about that ambitious thing? Can we like add that to the tree or something? No. The backward chainer is single-minded, focused, and stupid. Even though it's later going to need to prove that she's ambitious, it doesn't care. It just sees the villain there. Villain, oh, that's great. As long as we've got the antecedent we are going to be happy.

This is actually an important point, not in this particular problem, but in some other problems. So by using P2, we're obviously going to have the antecedent x lives in Slytherin dungeon. And x here is Millicent. So Millicent lives in Slytherin dungeon. All right, so what's the first thing we do when we see this new assertion? We check the assertions. Is it in there?

Yes. That's not where everyone lost points, but yes, it is in there. Check. This OR node is happy. Now we move up to the AND node with Millicent is ambitious. So Millicent is ambitious. What do we do now?

AUDIENCE: So isn't the consequent of P2 that x is a villain and x is ambitious. I don't know if it's [INAUDIBLE] or whatever, but in our assertions we have that Millicent is ambitious. Is that necessary to let us--

MARK SEIFTER: To let us use rule two. Actually, no. The question is, do we need the x is ambitious to be in our

list of assertions in order to use P2. For instance, if we didn't have the assertion that Millicent is ambitious, would P2 have a problem firing? So, it's a very interesting question. It's sort of what actually I was trying to mention earlier when I said it's single-minded, it's focused, it doesn't care about the other one.

Because you might say, well, wait a minute, P2 could never have actually fired if it didn't have both of its consequents. But in fact, the backward chainer doesn't care. All that the backward chainer is looking to do is to prove whether or not there's a possibility that Millicent might be a villain.

It's not trying to say, oh, all the results of some certain rules are making Millicent a villain and all the other things that are there, like ambitious, are in the database. It's not the forward chainer. It actually doesn't care. This sometimes leads to unnecessary computation. But it's very simple, and it's very simple to code.

So it often will probably speed you up in the long run. Like if there's, for instance, 100 thing in the then and you only care about one of them, it would be a waste, almost, to add those 99. There's a follow up over here somewhere? [INAUDIBLE]

AUDIENCE: Order m of 99. And so [INAUDIBLE]

MARK SEIFTER: You can try. However, then at that point, every time you check the assertions, you'd be attempting to check those 99. Also, you might not have used those rules. The question is, could you make a hash table with all of the 100 things that were in the consequent? And then, after having made that hash table since you were already walking through those anyway when you were checking it, you could add those to the assertions. The problem is that you might not necessarily want-- with the backward chainer-- you might not necessarily want to think that all those things in the consequent are necessarily true.

However, you might not care about them. And then you have to make an order and computation every time you check that list of assertions in the upper left.

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: You don't know what assertions there are. You have to check the entire list of assertions in the upper left every time you get to a new node in the goal tree. So each one is order 1. But I'm not saying to find a specific assertion is order n. I'm saying every time you get a new node in the goal tree, you have to check all the assertions you've added.

AUDIENCE: You have to do that anyway. [INAUDIBLE] assertions.

MARK SEIFTER: Yeah, you'd have to check for the four. But let's say that there were 10,000 consequents of P2. You would have to check 10,000 instead of four if you added them all after proving P2 was true on the goal tree. Also, another important note is that you might not use every branch of the goal tree, if there's like an OR that's up higher and you wind up using a different branch, in which case you'd probably have to make a separate hash table for every sub-branch that you have and then remember which hash tables were updated based on which sub-branch. And then sub-branches die.

It's probably more effort than it's worth. It's certainly an interesting question. It's a great question for a debate in recitation. Also, I'm happy to talk about it later. I think that someone who very intelligently made hash tables and thought the problem through, as Patrick said, more knowledge can mean a better-- well, actually maybe he hasn't said more knowledge can mean a better search because we haven't done that lecture yet, but--

AUDIENCE: You're talking about implementation decision and using hash table might well be a good way to do this. The problem is that we're presuming some things about how the rules are firing and what [INAUDIBLE] they're firing that depend on the order of assertions in the table. So if we used the hash table, we'd lose the order. [INAUDIBLE]

MARK SEIFTER: That's true. That's true. But I'm thinking like if someone wanted to make that implementation as like do some research in rules based systems, it's possible you can increase the running speed. In this class, we are not completely as focused on the fastest algorithms. But I still think that a cool thing to try. Questions from people in the crowd? We'll start with you.

AUDIENCE: So ignoring the hash table?

MARK SEIFTER: Ignore the hash table. That's a good idea because it's a little extra enrichment thing. It's not anything you would possibly need to know for the quiz.

AUDIENCE: [INAUDIBLE] So if it became an [INAUDIBLE] are you using this "then" statement because you want the villain. So along with using this you get x is ambitious. [INAUDIBLE] used after that, another "then" statement that we also use that says x is not ambitious. So how would you resolve that?

MARK SEIFTER: So the question is, let's say you used rule two somewhere in the tree to get that x is a villain,

which then says that x is ambitious. Then later, you have x is not ambitious somewhere in one of the other rules that you then later need. The question is, how do you resolve that? What does it do? Well, first of all--

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: First of all, interestingly, if it says x is not ambitious-- literally like that. And I'm not being pedantic here. This is one of the tricks someone played in a previous quiz. You can have that and x is ambitious both in the list because it's a dumb rules-based system it doesn't know that you can't have both of those on the list.

So if you have a positive assertion in a consequent, x is not ambitious in a consequent, x is not ambitious, it'll be happy to add Millicent is not ambitious while Millicent ambitious this is already there, because it's stupid. Now if it had delete Millicent is ambitious in the consequent and deleted something that was previously there, that would be an interesting problem. It could cause mistakes.

Your back chaining would probably make the mistake of allowing this. However, this is exactly what we do not, I believe as a policy, do not have DELETE statements on the quizzes. At least previous quizzes did not have them, except for in a hypothetical of what happens if we add a DELETE statement here. I don't think we've ever made people work things through with the delete.

Delete would possibly cause some issues. Question in the back?

AUDIENCE: So, just to check this, we're doing backwards chaining, we don't actually add assertions to your [INAUDIBLE].

MARK SEIFTER: Never add assertions in backward chain to the assertion table. It's dumb. The system is dumb. I'm not saying it's a dumb idea to do this. It's actually pretty fast to do it this way. But the system is dumb. The question is, do you add assertions or you don't add assertions? You do not.

You simply go through, checking all the things on your goal tree until the goal tree is either proven or disproven, then you're out of here. So, cool so far, everyone? Good questions from everyone. These are all questions that are things that often trip people up.

So, our next thing, Millicent is ambitious. We sort of-- the cat is out of the bag because there is

a question mentioning that there is an assertion that says that. But I direct you guys to our favorite rule, rule two. Shouldn't we use rule two to prove that Millicent is ambitious? The answer is if you want to lose four points or however many points, then yes.

But if you don't want to lose points, then this is already correct by virtue of assertion one. So I guess I should write, also, that this over here, Millicent studies a lot rule, this rule is rule four. Yeah, that's rule four [INAUDIBLE] and protagonist. Down here to prove that she studies a lot is rule three.

And then at the bottom here we have what rules here? We have the protagonist rule, which is one, and the villain rule which is two. This is correct virtue of A0. OK, great. So we've proved this whole AND node. We know that Millicent studies a lot. But now we're going to have to go a little bit more quickly based on time, and so I will do the remaining things. Anyone who has not been called on is off the hook. But that doesn't mean not to pay attention because there's some interesting stuff still to come.

So, Millicent is a protagonist. Well, good news, everyone. There's one other reason that I can do this and that's that you've already told me what I need to do. Most is a protagonist, you use rule one, rule one. And check if she lives in Gryffindor tower. Now, will it do it again? Yes it will. It hasn't cached that it's already tried that.

Now, hash table not withstanding, caching something you've already tried may be a good idea with backward chaining, but we do not do that in this class. That's an implementation detail. It's another idea of something you can do. We don't catch things that we've already tried. We try them again, dammit. And when we try it again, maybe it'll work this time. No, it never works. It never works this time.

This fails. This AND node fails. The whole thing fails. They're not friends at all. They're bitter enemies. Millicent does not become Hermione's friend. So now they ask us a few questions which are pretty easy to answer based on the ordeal we've just been through. So, determine the minimum number of additional assertions that we would need to add for Millicent to become Hermione's friend.

But you're not allowed to add an assertion that matches a consequent of a rule. You can only add an assertion-- in other words, you can't add an assertion that will prove some other rules. You have to add an assertion that directly-- yeah, that's basically the rule here. Because there's a lot of choices. But he wanted one particular answer.

And so can anyone think of an assertion that doesn't match any consequent of any rule that will make her be Hermione's friend. A lot of people say Millicent lives in Gryffindor. That's correct. So, part three, your solution to part two causes an uncommon situation. What is the uncommon situation? And what should you do to the list of assertions to solve this problem?

Does anyone know what the uncommon situation would be if we added that she lives in Gryffindor? Hand is raised up here. What is your answer? [INAUDIBLE] That's true. She lives in Gryffindor and Slytherin. So how would you fix that? [INAUDIBLE] Yes, you could take out Slytherin from the assertions and say that she only lived in Gryffindor.

A lot of people gave the answer to this question, which is a perfectly reasonable thing to do in a rules-basis system. They said, well, we can add a rule that says that if you live in x and y is different than x, then you can't live in y. The problem with that, among other things, is that we asked for a way to change the assertions and not the rules. And you gave us a way to change the assertions. That's the right answer.

There was another backward chaining problem, this is 2009 quiz one, and I will leave it off for the moment. You guys should take a look at it, in particular with variable binding. Because remember, you always have to bind the variable that's relevant to you. But that doesn't mean that you always have to bind all of the variables.

For instance, let's say that we wanted to prove that Millicent has a bad term, which as it turns out, is what we wanted to prove in the other backward chaining. Very quickly and without doing the problem, can anyone who thinks they're very clever at backward chaining tell me exactly what things would be added to the goal tree to prove that Millicent has a bad term?

Raise your hand. I won't pick out a victim because I'm asking this to happen pretty quickly. No one? All right.

AUDIENCE: You would add the three antecedents to rule--

MARK SEIFTER: To rule five. So what would you add specifically?

AUDIENCE: For anyone, variable-wise, you would just have to show that there is at least one person who matches.

MARK SEIFTER: So can you just sort of read out, what would the first thing say with the snogs?

AUDIENCE: X snogs-- or Millicent snogs anyone.

MARK SEIFTER: Yeah, Millicent snogs y. That's crucial. Some people did a lot of different things. Some people put Seamus in already. You can't do that. The system is stupid, it doesn't know that the person is Seamus. Also, some people said it worked, because look, they're snogging each other, but it's the wrong snoggle direction up there. Because we have Seamus snogging Millicent, not Millicent snogging Seamus.

But yes, exactly. It would be Millicent snogs y. Millicent lives in Gryffindor tower. y lives in Slytherin dungeon. So, let's give forward chaining all the attention it deserves, about eight minutes, which will be more than enough.

OK, we still got all these rules. We've still got all these assertions. Instead of doing that horrible backward chaining-- well, it's not that horrible-- but we'll do forward chaining. It'll be really easy. We'll just add new assertions as they go. Remember the tiebreak order. Rules in order from 0 to 5. And if the same rule could fire off with multiple different assertions, we'll use the assertions in order from 0 to 3.

So let's see. We don't have much time, but with our four assertions, let's see, what rules possibly could match with our four assertions? Well, I'll figure out for you. Do we have an assertion about an ambitious person? Yes. But they're not a squib. So not rule zero. Do we have someone that lives in Gryffindor? We absolutely do. Rule one is matches.

Do we have someone that lives in Slytherin? We do. Rule two matches. Do we have a protagonist or a villain? We don't have any protagonist or villain. Rule three is not going to match because it's in an "and." do we have someone who studies a lot? We do not have someone who studies a lot. Rule four is not going to match.

Do we have some snogging? We do have some snogging. So rule one, two, and five match. So m for matching, we have one, two and five. Everyone, which one wins the tiebreak between one, two, and five? One, because it comes first numerically. Which one is rule one if x lives in Gryffindor tower?

The only binding for x is Seamus. Seamus lives in Gryffindor tower. We're firing off rule one. So new assertions, the first assertion that we'll add is Seamus is it a protagonist. Great. Now, there's one other thing, and it's the main thing I wanted to tell you about in forward chaining. About our old friend, rule one, now that we've done this, does rule one still match an assertion

in the database? Yes. It does.

But what's going to stop us from constantly doing rule one every time because it comes numerically before the other rules? What stops us there is a part of our implementation. And it's a very important part that people sometimes forget. It's what I like to call the no-impotent rules implementation detail. That is, if a rule is completely, 100% impotent, it would do absolutely nothing. Then you do not fire it. You go to the next one. That's pretty important.

Let's say that you possibly had a delete, which you won't have on the quiz. That means that if the thing to delete is missing, it's impotent. If the thing to delete is there, it's not impotent because you can delete. Let's say you have 500,000 things that you're going to add to your assertions. 499,999 are already in your database. One of them is missing. Is it an impotent rule? No, it's not. You have to fire it anyway.

You can't be like, oh, it's mostly impotent rule. No. You have fire off the rule if it will do anything. But right now rule one won't do anything because we've already added Seamus is a protagonist. So what rules match down? Well, Seamus becoming a protagonist is nice and all, but he's not ambitious, so we still don't match rule three. We still match rules one, two, and five.

As I basically just got finished telling you, the one that we're going to fire off now is two. That's right. So, rule two is if x lives in Slytherin dungeon our only [? binding ?] for that is Millicent. And that will show that she's a villain and she's ambitious. So what assertions will we add? Millicent is a villain. And people stopped. That's right. Good job, everyone. You remembered that Millicent is already ambitious from assertion one, so we don't have to add them both. A lot of people did. They were wrong.

So Millicent is a villain. The other part of that rule doesn't do anything. But it's not an impotent rule because one of the things is not on there. Great. So, what rules match now? Well, as it turns out, now that Millicent is a villain and ambitious, we match rule three as well. So we match one, two, three, and five. What rule are we going to fire, everyone? Three. Because one and two are impotent rules.

We'll fire rule three. Well, Millicent is a villain. She's also ambitious. She's our only match for three. Therefore, what assertion do we add? Millicent studies a lot. That's right. So, Millicent studies a lot. However, she's not a protagonist. We still don't match rule four. Oh well, we

figured that out already ourselves from backward chaining that we were never going to match that.

So what matches? It's still one, two, three, and five. Seeing as one, two and three are impotent, the only match is five. So the only match we care about, the only one we're going to fire is five. And the result? Let's see. x snogs y. That's Seamus snogs Millicent. x lives in Gryffindor, that's Seamus. y lives in Slytherin, that's Millicent. So what do we add? Seamus has a bad term.

Simple, painless, it works out. We're done. And we got 100 on this quiz. Hopefully you guys will, too. Question? The question is, if you have a new assertion which matches a rule but it's really high number, it's all the way down at the bottom of the rules, but it's new and it has new stuff, should we do that first, maybe, because it's new?

Oh, you mean that it has a lower number, a lower number?

AUDIENCE: [INAUDIBLE]

MARK SEIFTER: OK, so the question is, if a rule-- let's say that rule three made Millicent be a squib. Then yes, you'd immediately not only match zero-- zero would immediately fire because numerically it comes first. You'll see that on some of the old quizzes. And you may see that in tutorial. If you guys want to do some more practice problems in tutorials, your TAs will have that available as one opportunity in tutorial. But yeah, that's exactly right.

It happened that it went in order this time, but it will definitely hop anywhere it can in order to get to the lowest numbered assertion.