ANON - 14:31
Q: is there only 1 part for lecture 6?
Priority: N/A
Konstantin Kuzmin - 14:31
A: No, three parts.

ANON - 14:32
Q: ah okay. I only see 1 part on the mediasite
Priority: N/A
Konstantin Kuzmin - 14:33
A: Check all pages, please.

ANON - 14:32
Q: What is the timeline for homework 1 grading?
Priority: N/A
Konstantin Kuzmin - 14:34
A: We are doing our best to get HW1 grades out by early next week.

ANON - 14:32
Q: nevermind it was just acting strangely i see all parts now
Priority: N/A
Konstantin Kuzmin - 14:33
A: Great!

ANON - 14:37
Q: When will study materials be posted for Exam 1?
Priority: N/A

ANON - 14:38
Q: A bit off topic: Is there a way for us to see what questions we got wrong on the quizzes?
Priority: N/A
Konstantin Kuzmin - 14:43
A: You can see your answers if you just go back to the gradeable on Submitty. Quiz solutions will be discussed as part of preparation for the test, so you would be able to compare correct
A: Unfortuantely, Submitty only shows rubric to the students for manually graded questions, not the autograded ones.

ANON - 14:40
Q: Can you explain the no "common prefixes" rule for the grammar not being LL(1)?
Priority: N/A
Steven Haussmann - 14:42
A: To be LL(1), a language must be parsable without reading more than one symbol ahead. If any of your productions have at least one symbol of common prefix, then you need to read at least two symbols ahead to distinguish between them.

Steven Haussmann - 14:43
A: So, for example, A -> aa | ab would not be LL(1), because reading ahead one symbol wouldn't distinguish between aa and ab

Ana L. Milanova - 16:26
A: Yes, what Steven says. If we have a nonterminal, e.g., A, and A has 2 or more productions that have a common prefix of symbols, then we say that the "common prefixes" rule applies. E.g, S -> if b then S E | if b then S, here the 2 productions for S have the common prefix "if b then S".

ANON - 14:43
Q: How do we denote an OR in prolog? (If possible)
Priority: N/A
Steven Haussmann - 14:48
A: The ; operator lets you do this. It lets you provide several options, like in happy(X) :- rich(X) ; famous(X).

Steven Haussmann - 14:49
A: It's more-or-less equivalent to just writing two separate rules, but it can be more concise.

ANON - 14:50
Q: Backtracking to another possible binding occurs at the point when a unification fails, right?
Priority: N/A
Ana L. Milanova - 14:57
A: Yes. And also, if Prolog has exhausted all the possibilities at a branch.

ANON - 14:53
Q: why does Prolog, in some instances, print false after all the options have been exhausted?
Steven Haussmann - 14:55
A: I believe this occurs when it can't determine that there are no further answers before you ask for another one -- consider how it will sometimes be able to stop automatically

Steven Haussmann - 14:56
A: When you ask for another answer, it backtracks and tries to find more bindings. If that fails, it prints false.

ANON - 14:53
Q: Why would Prolog try matching takes(jane, cs) with takes(jane, his) when it already tried takes(jane, his) with takes(jane, cs)? In other words, why would it not resume searching for matches where the current entry is?

Ana L. Milanova - 14:56
A: We have to use negation. takes(X,Z), takes(Y,X), not(X == Y) will do it in this case. More on negation next time.

Ana L. Milanova - 14:59
A: Sorry, above answer is the answer of a different question.

Ana L. Milanova - 15:01
A: Prolog searches from the beginning for takes(X,Z), and separately, from the beginning for takes(Y,Z). (I might be misinterpreting the question.)

ANON - 14:55
Q: How do we avoid jane being in class with jane? would we have to add a production like notSamePerson(X,Y)?

Ana L. Milanova - 15:01
A: We can use negation. takes(X,Z), takes(Y,X), not(X == Y) will do it in this case. More on negation next time.

Ana L. Milanova - 15:07
A: I have a typo in the answer above: classmates(X,Y) :- takes(X,Z), takes(Y,Z), not(X == Y).

ANON - 15:00
Q: Why is [x ] y] a two leaf tree instead of a 3 leaf one (to include a null list)?
Steven Haussmann - 15:02
A: Both X and Y are unbound variables. It's true that Y will need to be a list of some kind for this to be a valid statement, though.

ANON - 15:01
Q: Thank you!
Priority: N/A

ANON - 15:01
Q: I don't know if this is just my computer, but the audio seems to be very glitchy, cutting in and out
Priority: N/A
Ana L. Milanova - 15:06
A: The audio turned better than usual on this lecture, it sounds quite clear in Mediasite... It sounds fine on my side when Prof. Kuzmin streams, but it is probably different on different machines/connectivities.

ANON - 15:02
Q: What " | " means in the list?
Priority: N/A
Ana L. Milanova - 15:03
A: "|" is used to write list in head-and-tail notation. [H | T] denotes a list with head H and tail T.

ANON - 15:02
Q: Is unifying saying like they are the same?
Priority: N/A
Steven Haussmann - 15:03
A: When two variables are unified, we are asserting that they mean the same thing (by representing the same structure). So, yes, that's a reasonable way to describe it.

ANON - 15:07
Q: So if I had [X | Y] and was asked to draw the representation, it's okay to draw a tree with 2 leaf (w/t the null leaf) or a tree with 3 leaves (w/ the null leaf)? I'll get full points?
Priority: N/A
Ana L. Milanova - 15:09
A: No, it is strictly a list of 2 leaves, X and Y. You will have a different tree, the three with 3 leaves, if the list was [X,Y].
ANON - 15:07
Q: Sorry, I missed what =? means
Priority: N/A

Ana L. Milanova - 15:09
A: L =? R is just a notation I used to ask the question "What is the result of the unification of L and R?"

ANON - 15:09
Q: So if you have a Predicate that is R, would it be incorrect to do "[H|T] = R" to extract Head + Tail? As in, we can never convert between Improper + Proper Lists?
Priority: N/A

Ana L. Milanova - 15:11
A: We cannot "convert" from improper to proper list. If you try to unify [1|2] with [1,2], then unification fails.

ANON - 15:11
Q: But if R is a List, can we do [H|T] = R to extract head and tail data?
Priority: N/A

Ana L. Milanova - 15:11
A: Correct.

ANON - 15:11
Q: What's the difference between proper and improper lists?
Priority: N/A

Ana L. Milanova - 15:12
A: A proper list has the "terminal leaf" being the null list []. An improper list may have a constant or a structure as "terminal leaf". So these are different structures.

ANON - 15:13
Q: So extracting the Head, Tail via [H|T] = R does not change the List to an Improper List?
Priority: N/A

Ana L. Milanova - 15:15
A: No. Unification does not change the structure of the list. If R is the list [1, 2, 3] then we'll extract 1 into H and [2,3] into T.

ANON - 15:13
Q: Just checking my understanding with [X | Y], this will form only improper lists if Y is an element, and will only form proper lists if Y is a list?
Priority: N/A
A: Yes. A proper list always has a tail that is either another proper list or an empty list.

A: Yes, p(... X ...) represents a structure that has X nested somewhere. The structures can be arbitrarily complex, you can have p(...,[[[X]],...]) or p(q(X),...), etc.

A: [H|T] is a way to write a list with a head H and a tail T. If R is a list, then the unification will make [H|T] equivalent to that list -- thus extracting the head and tail

A: It is significant that the structures are different. The list on the left has 2 leaves and the list on the right has 3 leaves.

A: I can't think of a good reason to use an improper vs. a proper list. But it is easy to create an improper list instead of a proper list during computation, then have unification fail when we don't expect.

A: That is acceptable. H will be 5, and T will be the empty list.
ANON - 15:19
Q: so unification isn't assignment, but does basically the same thing as assignment?
Priority: N/A
Steven Haussmann - 15:20
A: It does not perform an assignment; it does not copy a value from the right hand side to the left hand side. It is true that both assignment and unification assure that both sides are equal after they execute, I suppose.
Ana L. Milanova - 16:38
A: Yes, essentially they both make value on the left and the one on the right "equal". But assignment does not "fail", e.g., X can have a value of 1, and we can reassign a value to X, the assignment X = 2 works. But unification may fail, e.g., if X is 1 then
Ana L. Milanova - 16:38
A: we try X = 2, X = 2 fails.

ANON - 15:20
Q: Would [[] | 5] or [{null list} | 5] be an improper or a proper list if {null list} is the head?
Priority: N/A
Steven Haussmann - 15:22
A: The only thing that decides if a list is proper is whether its tail is either a proper list or an empty list. Those are both improper.

ANON - 15:21
Q: How do we access the Nth element of a proper list? do we recursively extract the head and tail?
Priority: N/A
Steven Haussmann - 15:22
A: You can write it out as [A,B,C,D,E|Rest] = SomeList.
Ana L. Milanova - 16:38
A: Yes. No direct access in lists.

ANON - 15:21
Q: Can you clarify binding vs unification?
Priority: N/A
Ana L. Milanova - 15:25
A: "Binding" refers to the variable being bound to a value. E.g., X is bound to "jane" means that X has the value of "jane". "Unification" refers to the process of making terms equal, and the process of unification may result in bindings of variables to values.
Q: Can we only unify a free variable with some other variable (maybe free, maybe not)?
Priority: N/A
Ana L. Milanova - 15:27
A: You can unify a variable with a structure or with a constant as well. We may unify a variable with another variable as well; if second variable has value, then first variable gets that value as well.
Ana L. Milanova - 15:28
A: If neither variable is bound to a value, then if either one gets bound to a value, the other one gets that value as well.

ANON - 15:23
Q: what is actually the difference between a proper list and an improper one. like besides what the leaf node is. Is one better than the other. Why would one want to use one rather than the other?
Priority: N/A
Steven Haussmann - 15:25
A: A proper list is consistently shaped: its tail is always either a proper list or an empty list. An improper list will have a discontinuity at the end: the tail could just be an atom, for example.

ANON - 15:24
Q: why does head tail syntax make improper lists if that is not ideal? if it is not ideal, why ever use head tail syntax?
Priority: N/A
Steven Haussmann - 15:26
A: It makes an improper list if the elements you provide result in one, like [2 | 3]. There's nothing stopping you from making a proper list: [2 | [3 | []]]

ANON - 15:25
Q: so bindings here are just possibilities?
Priority: N/A
Ana L. Milanova - 15:29
A: Yes, you can think of bindings this way. Prolog searches for answers, and it may try different possibilities/bindings.

ANON - 15:27
Q: What happens if you `member` with unbound variables in both the left and right hand sides?
Priority: N/A
Steven Haussmann - 15:30
A: Prolog will generate an infinite number of possible bindings for X and Y if you ask for member(X,Y). Each one is a list of distinct unbound variables (with X somewhere in it), since the only
criteria is that the X shows up somewhere in Y
   Ana L. Milanova - 15:30
   A: member(X,Y).
   Y = [X|_14624] ;
   Y = [ |14622, X|_15354] ;
   Y = [ |14622, _15352, X|_16084] ;

ANON - 15:33
Q: Is it possible to make this a one line implementation?
Priority: N/A
   Ana L. Milanova - 16:45
   A: Not that I can think of.

ANON - 15:34
Q: On slide 9, improper and proper lists, why can't abc unify with
   abc, and Y unify with Y?
Priority: N/A
   Steven Haussmann - 15:36
   A: [abc, Y] corresponds to [abc | [Y | []]]. No matter what
   value you choose for Y, [abc | [Y | []]] will never be equivalent to
   [abc | Y]
   Ana L. Milanova - 15:36
   A: They cannot unify in the sense that there does not exist a
   binding for Y that will make the two structures isomorphic (i.e.,
   "equal").

ANON - 15:36
Q: I'm still slight confused why member(a, b | a) fails. Why does the
   Unification fail?
Priority: N/A
   Ana L. Milanova - 15:41
   A: member(a, [b|a]) fails essentially because [b|a] is an
   improper list. We'll end up rumming member(a,a) and the member clauses
   expect lists as the second argument.
   Steven Haussmann - 15:42
   A: This is why the distinction between proper and improper
   lists is so important -- the tail of a proper list will always be a
   list.

ANON - 15:36
Q: To prevent failure with member(a,a), could we add a fact of
   member(A, A)?
Priority: N/A
   Ana L. Milanova - 15:38
   A: If we add this fact, then member(a,a) will succeed, but I
   am not sure off the top of my head how this change would affect the
expected behavior of member.

Steven Haussmann - 15:39
A: member(a,a) *should* fail; an atom is not a member of itself, just as [1,2,3] is found nowhere in the list of [1,2,3]!

ANON - 15:37
Q: are we allowed to use these library predicates in our homework?
Priority: N/A
Ana L. Milanova - 15:39
A: Yes, these are all standard built-in predicates in Prolog that you’ll have to use in most meaningful Prolog programs.

ANON - 15:39
Q: I see how it says that like you can ask for more answers? does this mean we call it and we can get all of the answers and iterate through somehow or does it mean we just have to keep calling the operation?
Priority: N/A
Steven Haussmann - 15:41
A: When working interactively, the program will pause and wait for input. ; will cause it to backtrack and give a new answer, whilst . will terminate. For non-interactive work, look at predicates like forall/2

ANON - 15:42
Q: But should it not be that member(a, [b|a]) goes to member(a, [a]) which leads to true?
Priority: N/A
Steven Haussmann - 15:43
A: No, because the tail of [b|a] is not [a]. The tail is a.
Steven Haussmann - 15:44
A: If you write [H|T], H will be the head and T will be the tail. So [3 | 2] is an improper list with a head of 3 and a tail of 2, for example.
Steven Haussmann - 15:44
A: [3, 2] is a proper list, with a head of 3 and a tail of [2 | []]. You could also write this list as [3 | [2 | []]].

ANON - 15:47
Q: It seems like most of these procedures can perform an operation and the inverse of that operation if the arguments have variables in the 'reverse' order. Is this true for all procedures defined this way (this 'recursive' method)?
Priority: N/A
Ana L. Milanova - 15:54
A: No, in general this is not true. Many predicates are not invertible, particularly ones that use arithmetic. This comes in last
part of lecture.

ANON - 15:56
Q: I got a question wrong on the second quiz but can't find where I can see what question I got wrong
Priority: N/A
Konstantin Kuzmin - 15:57
A: You can see your answers if you just go back to the gradeable on Submitty. Quiz solutions will be discussed as part of preparation for the test, so you would be able to compare correct solutions with your answers.
Konstantin Kuzmin - 15:58
A: Unfortunately, Submitty only shows rubric to the students for manually graded questions, not the autograded ones.

ANON - 15:57
Q: is there any difference between X is 4 and X = 4?
Priority: N/A
Ana L. Milanova - 15:59
A: In this case, there isn't. The right-hand-side is a numerical constant, so "is" proceeds immediately with unification. But there is a difference in X = 4+1 and X is 4+1.

ANON - 15:58
Q: so will quiz solutions not be posted until the test?
Priority: N/A
Ana L. Milanova - 16:00
A: We will go over (and show in lecture slides) all quizzes with answers, as a review for the test. We will do that during the lecture that immediately precedes the test.

ANON - 16:00
Q: Would X = 4 + 1 not work?
Priority: N/A
Steven Haussmann - 16:01
A: X would unify with 4 + 1, binding 4 + 1 to it. If you want the expression to be evaluated, you want to use "is"; X is 4 + 1 will result in a binding of 5 for X

ANON - 16:00
Q: Wait so in the case of this Sum Function/Predicate, is it dealing with an Improper List since we are using [H|T] or could either type accepted? Or is it based on the Base Case that it should be a Proper List?
Priority: N/A
A: It works only with proper lists, where all elements are integers.

A: Yes, due to the base case, if given an improper list, it will return false.

Steven Haussmann - 16:04
A: Your intuition about the base case is correct. [H|T] will happily unify with an improper list, but if T is not a proper list (say, it's a number), then no rule will match when we recurse on it.

ANON - 16:04
Q: why does ?- member(a, [b, c, X]). results in X=a?
Priority: N/A
Steven Haussmann - 16:05
A: Prolog finds a binding for X such that the predicate is satisfied.

ANON - 16:06
Q: Would the len of [A|B] be 2? Or does it not function on "improper" lists?
Priority: N/A
Steven Haussmann - 16:06
A: length([A|B], 2) will cause a type error.
Steven Haussmann - 16:07
A: length([A|B], X), I mean. My bad!
Steven Haussmann - 16:07
A: actually, scratch both of those...that's the case for if you have atoms or numbers or something similar in there. With unbound variables, it will find an infinite number of lists by binding different things to those variables.

ANON - 16:09
Q: What's the length of this list len( [1, [2, [3,4 ]], 5, ]R), if there exist nested list
Priority: N/A
Steven Haussmann - 16:10
A: The length of the list is the number of elements in the list. The length/2 predicate doesn't care what those elements happen to be.
Steven Haussmann - 16:12
A: I'm not sure what the R is here -- did you mean to write [1, [2, [3, 4 ]], 5 | R]? If so, the list could have a length of 3, 4, 5, or any greater number, depending on what R is.
Ana L. Milanova - 16:50
A: Assuming you mean len([1, [2,[3,4]], 5], R), i.e. R is the result, then R is 3. As Steven says, len does not care what the
elements are.

ANON - 16:10
Q: So "X is 4, X = X + 1" fails because X is a unified to a constant so it cannot be unified with a variable?
Priority: N/A
Steven Haussmann - 16:14
A: It fails because 4 can never unify with 4 + 1 -- they're different structures
Steven Haussmann - 16:16
A: This applies even when the two sides would evaluate to the same number. 2 + 2 = 3 + 1 fails.

ANON - 16:11
Q: When you do something like append(X,Y, [a,b]) and get X = [] Y = [a,b] is it unified so any calls of Y in the future will have the unified [a,b]. Also will this change when the ';' is called and get reunified X = [a] Y = [b]?
Priority: N/A
Steven Haussmann - 16:13
A: It's stateless, so the unification isn't "remembered" between invocations.

ANON - 16:13
Q: When making a predicate that takes in a list, can we always assume the input is a proper list or do we have to do any additional 'error checking' and are there times we should change code to accept improper lists?
Priority: N/A
Steven Haussmann - 16:16
A: I would assume that you'll get correct input, unless specifically asked otherwise.

ANON - 16:14
Q: Like len ([1, [2],3],R) we can get the length of this list is 3, so R = 3, so what if we have a nested list like [1, [2, [3,4] ] ,5 ], what's the length of this list?
Priority: N/A
Steven Haussmann - 16:15
A: In this case, the length will still be 3. There are three elements in the list: 1, [2, [3, 4]], and 5

ANON - 16:15
Q: When will study materials be posted for Exam 1?
Priority: N/A
Ana L. Milanova - 16:52
A: We are looking at about a week before the test.

ANON - 16:16
Q: could we solve sum([], 0). sum([A|B], R) :- .... not solving for improper lists and constants with an additional assert after our first base case, sum(R,R)? I expect this would return the improper tail or const as the result
Priority: N/A
Ana L. Milanova - 16:18
A: Yes, that should work.

ANON - 16:17
Q: Still confused with the meaning of f(H, H1) in Processing a list
Priority: N/A
Steven Haussmann - 16:18
A: f is a predicate that takes two arguments; you provide it a value in its first argument and an unbound variable in its second argument. The predicate will unify that variable with whatever its result is.
Steven Haussmann - 16:20
A: So, for example, your predicate might just be f(A, B) :- A = B. In this case, invoking f(3, H1) results in H1 being unified with 3.

ANON - 16:18
Q: Related to an earlier question: the only thing that would unify with 3+1 would be 3+1 right?
Priority: N/A
Steven Haussmann - 16:19
A: Yes, I believe that's the only arithmetic expression that will unify.
Ana L. Milanova - 16:19
A: Yes, correct. 3+1 won't unify with 4 or 1+3.

ANON - 16:21
Q: i missed the first video shown this lecture - what is just the last part of lecture 5?
Priority: N/A
Ana L. Milanova - 16:23
A: In the beginning of lecture we did some Q&A that from Lecture 5.