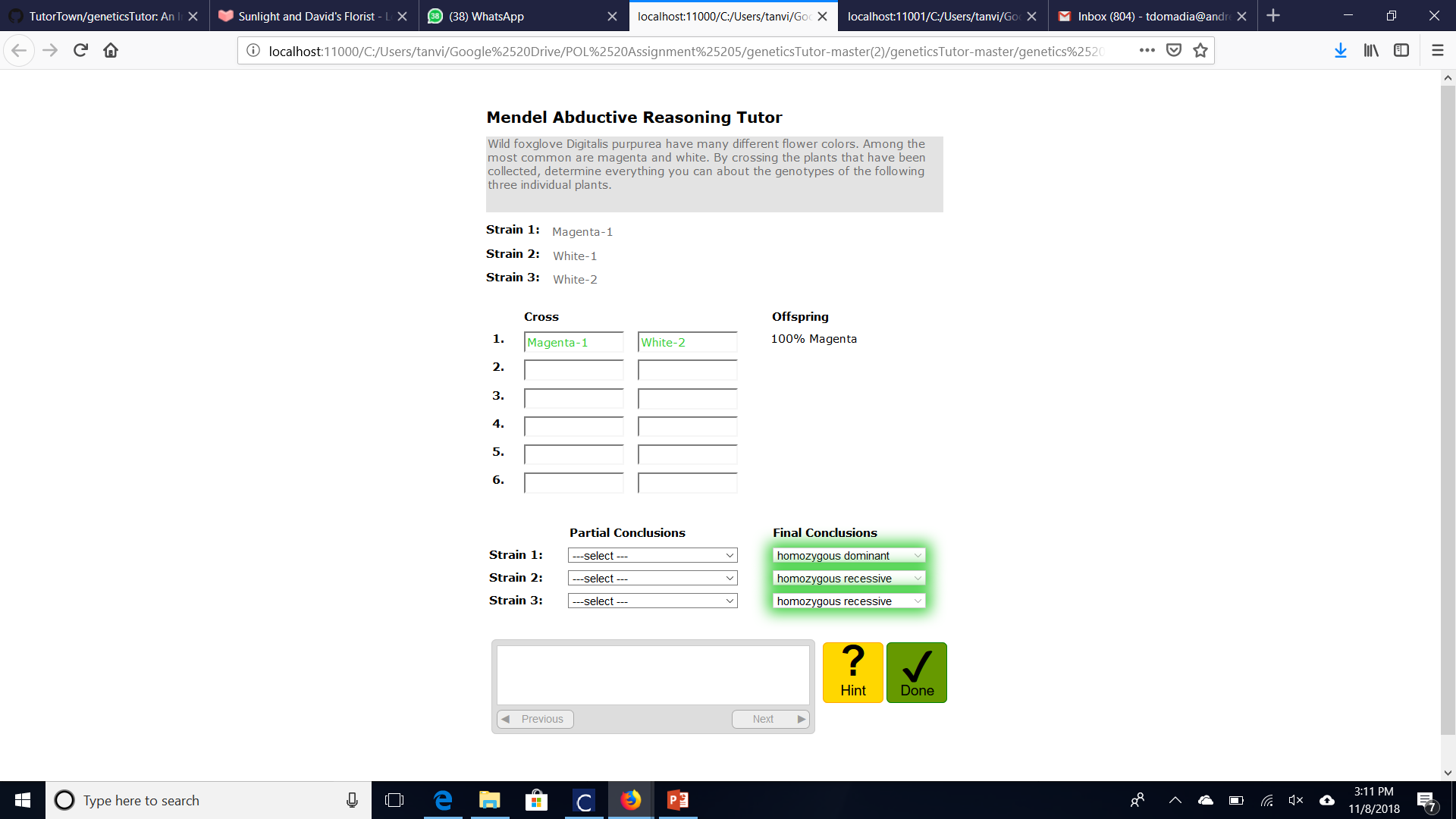
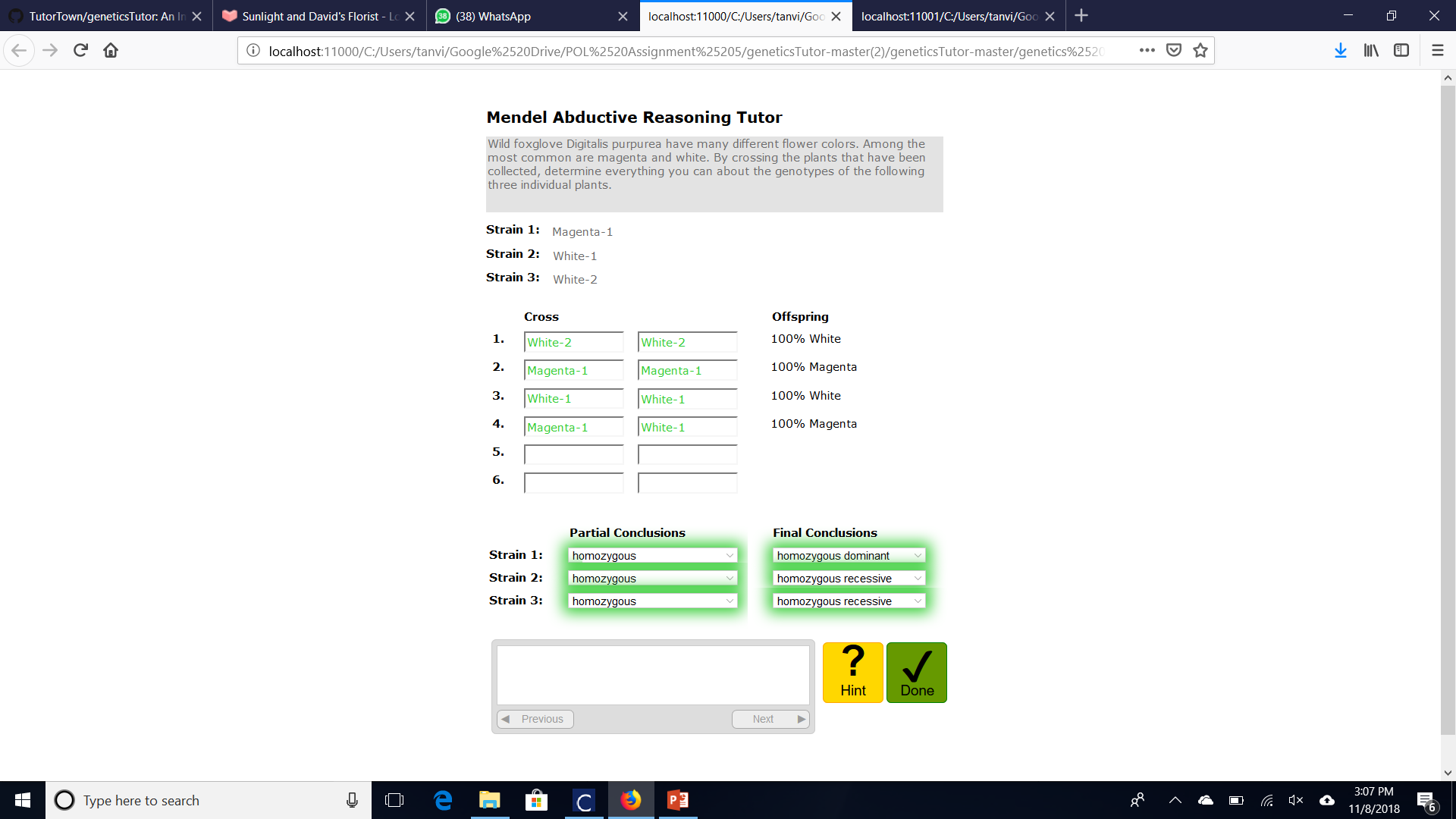
# Screenshots

## Foxglove

Screenshot 1



Screenshot 2

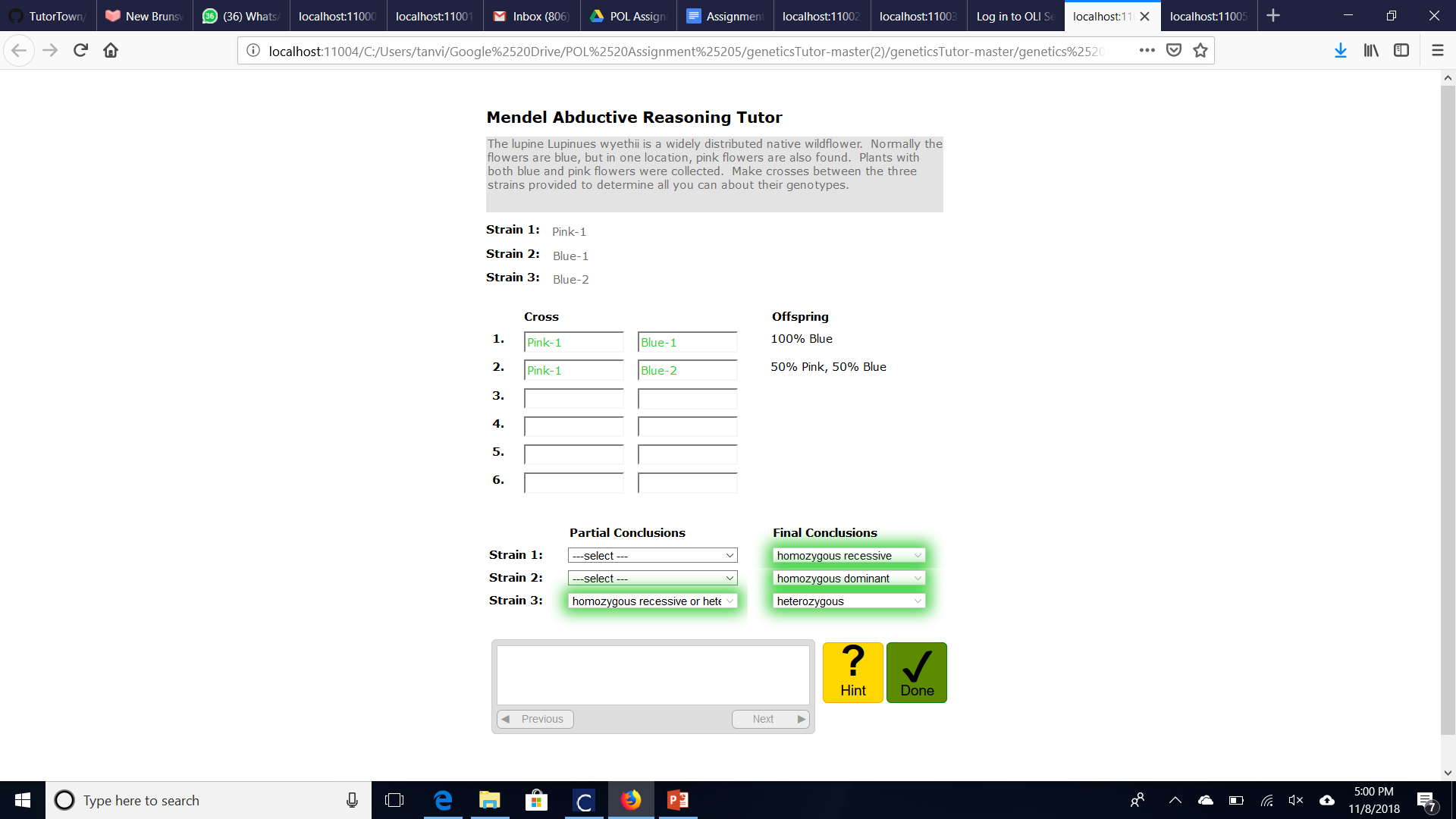


## 

## 

## Lupine

Screenshot 1



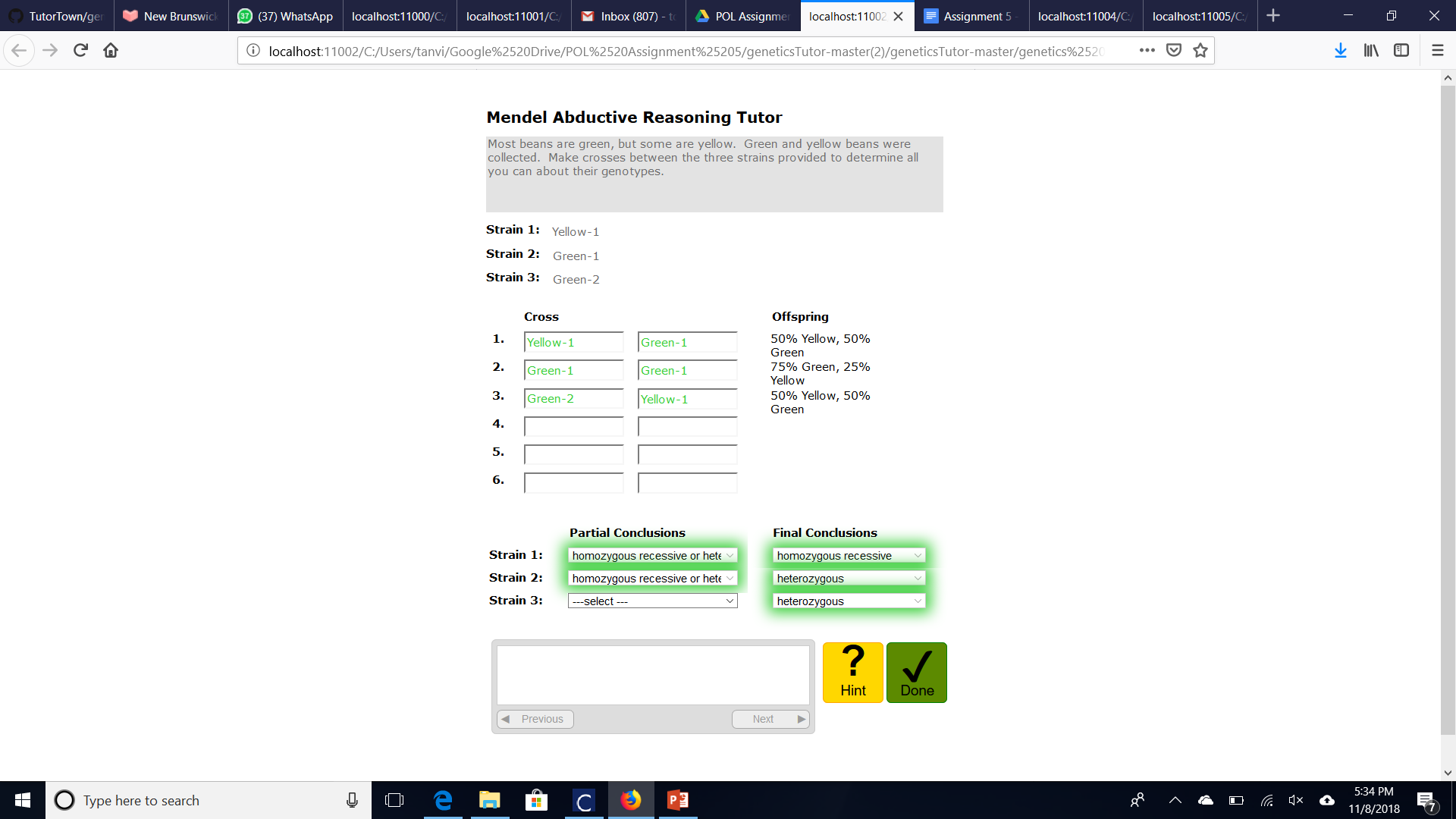
Screenshot 2

## 

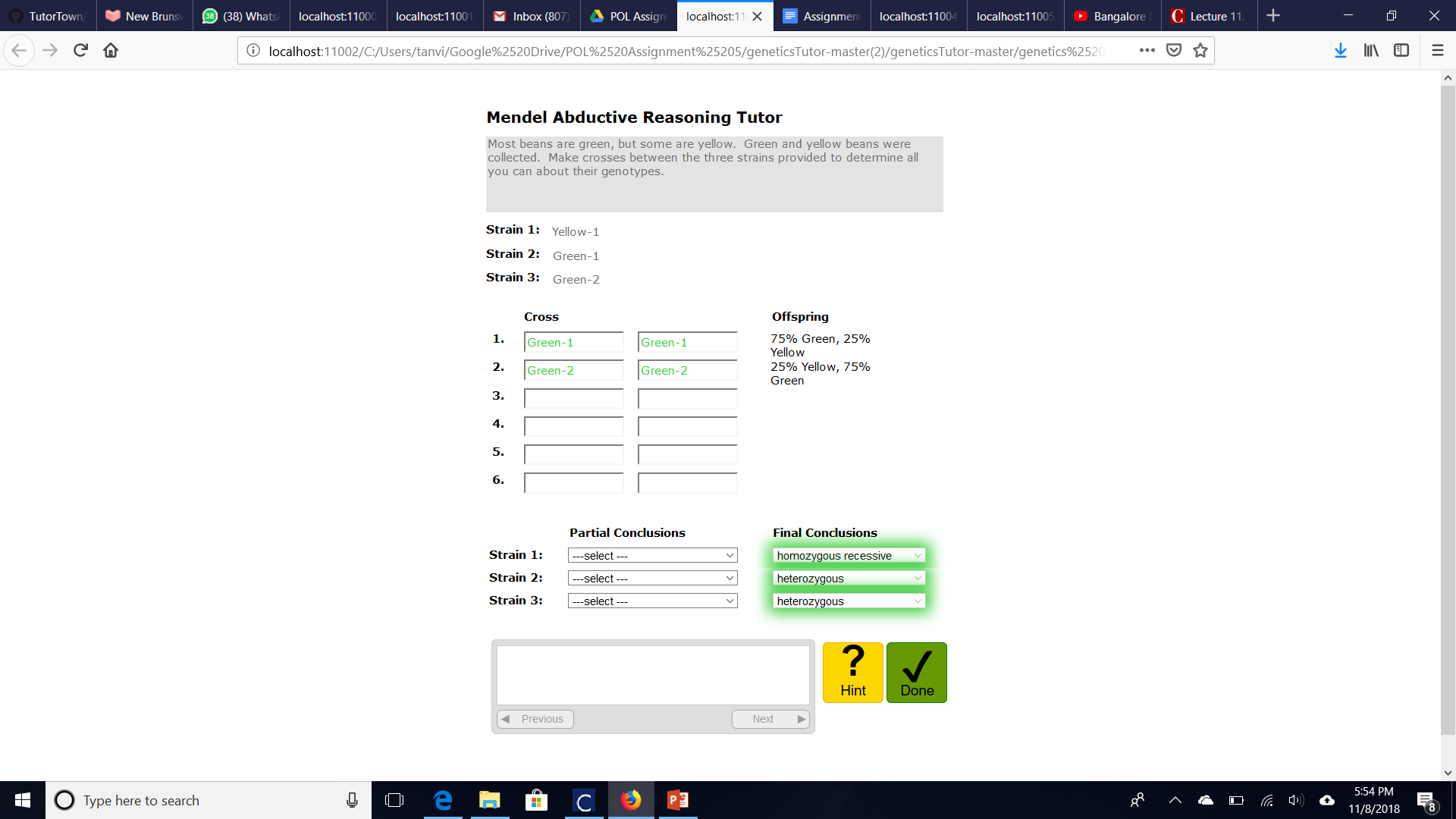
## 

## GreenBeans

Screenshot 1

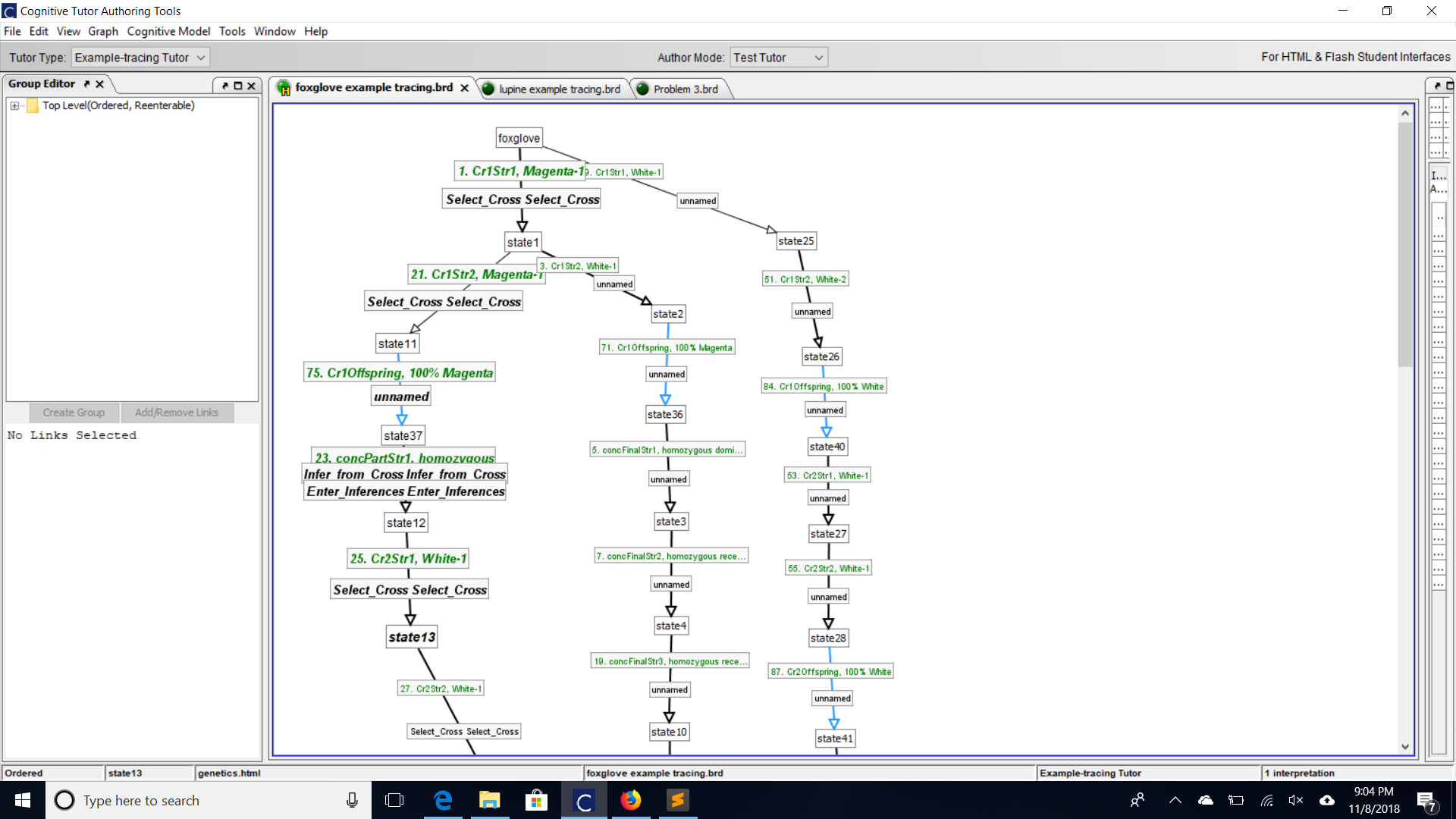


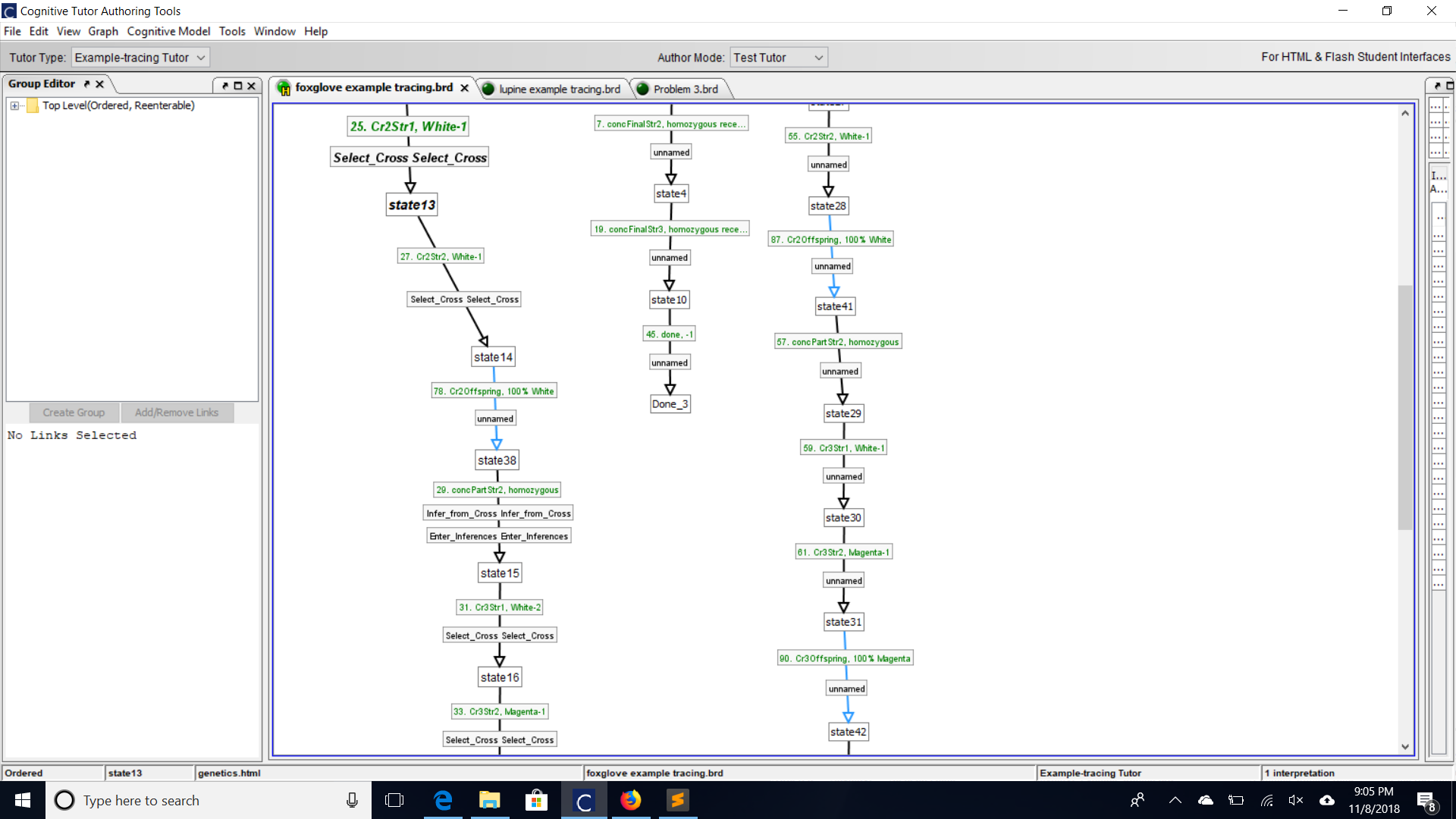
Screenshot 2

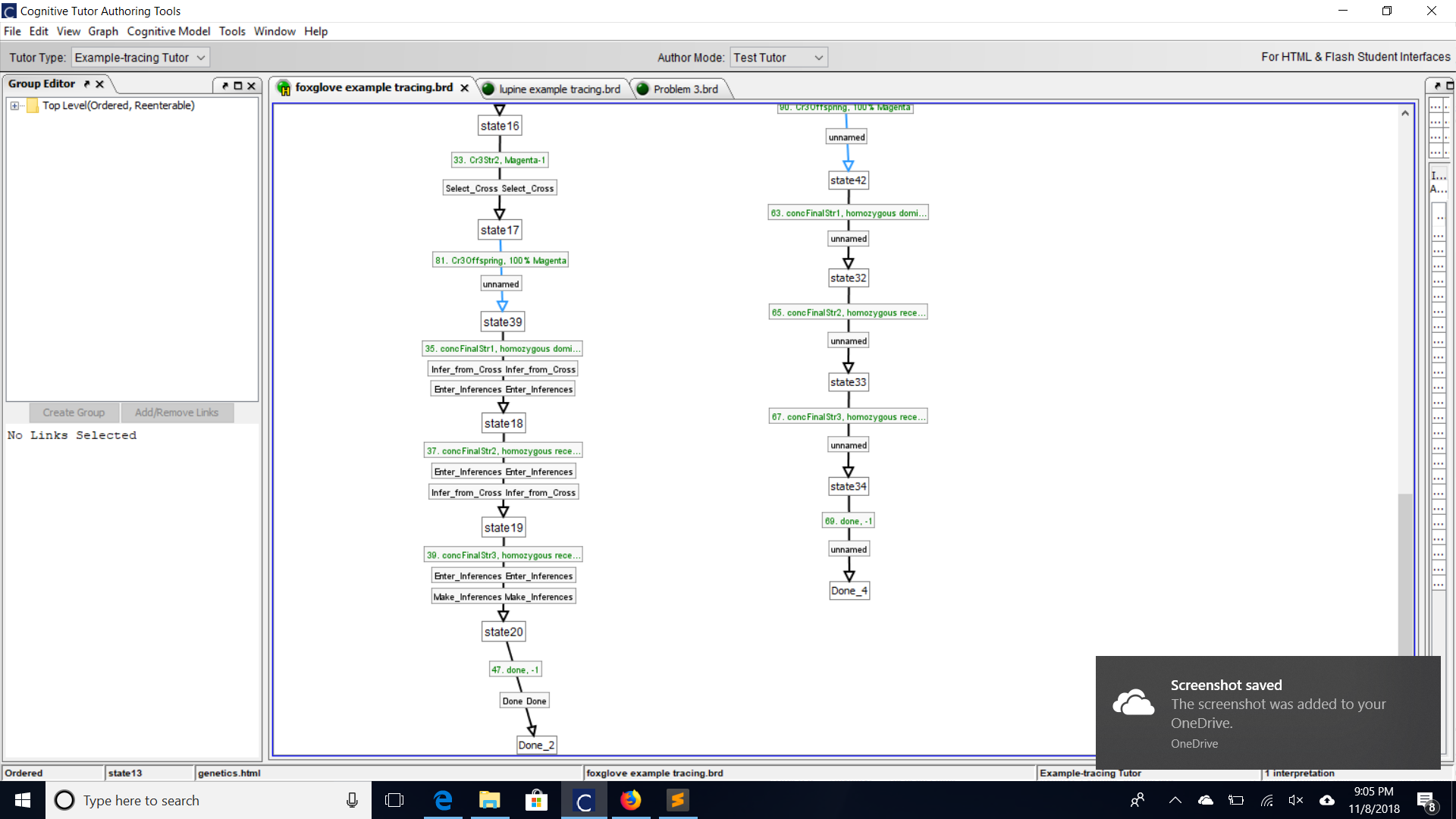


# Annotated graphs

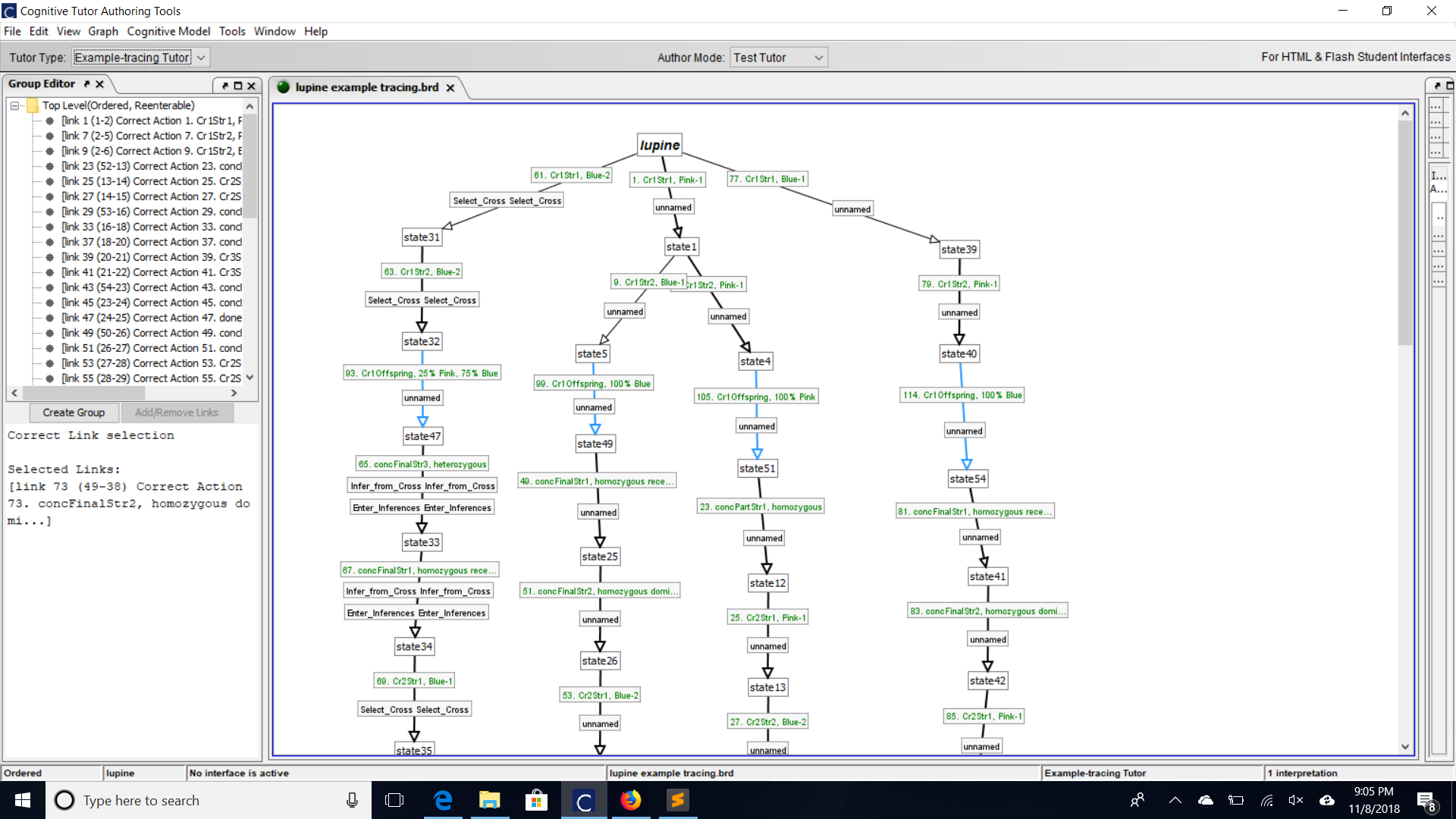
Graph 1

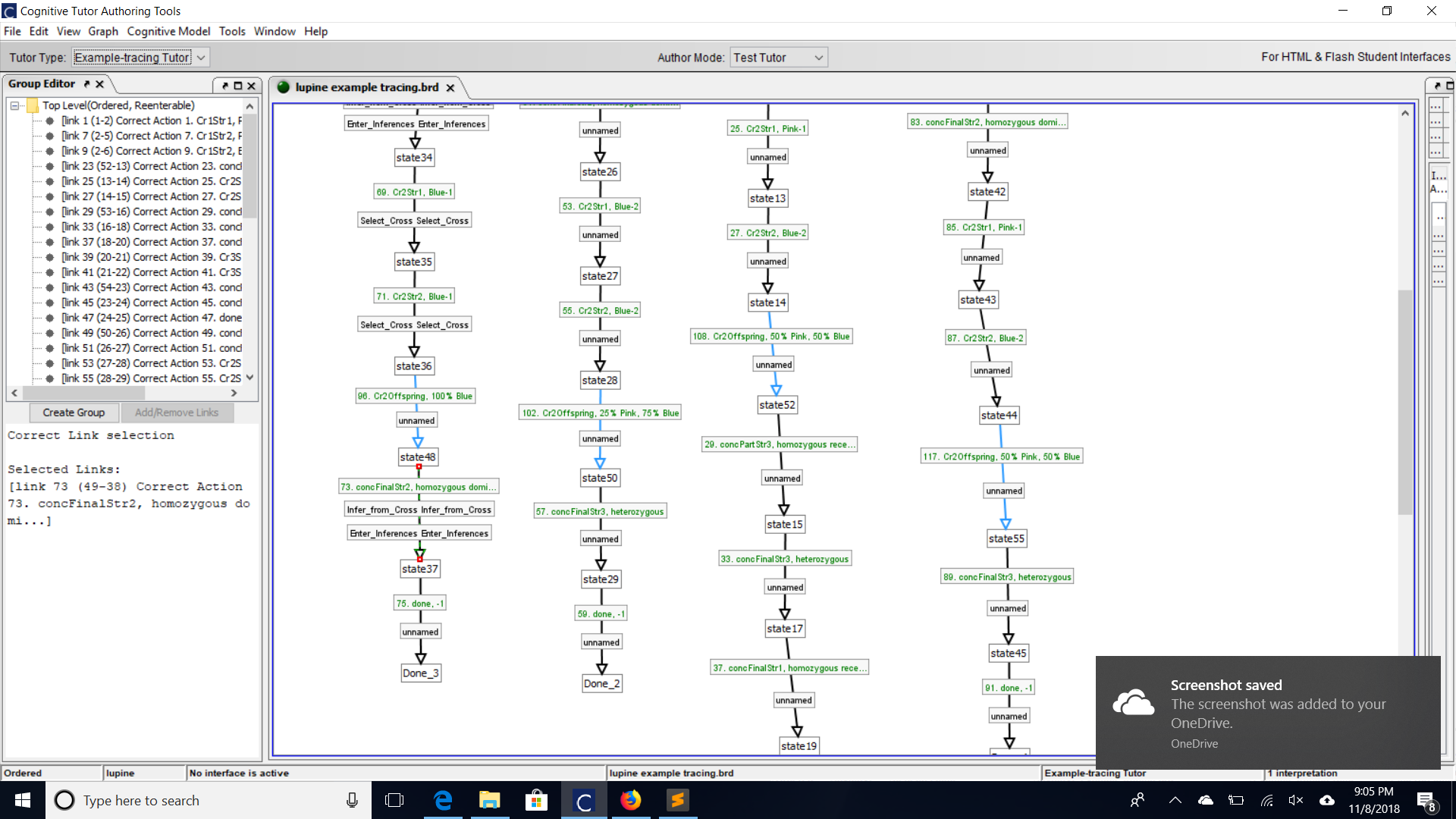






Graph 2





# Structure of working memory:



Problem:

* Name: String with the name of the problem
* subGoal: String with current subgoal
* firstStrainCurCross: String with the name of the first strain selected in the most recent cross
* curCross: String with the name of the most recently made cross
* interfaceRows: Array of Strings with the names of the interface rows
* dominantPhenotype: String representing the dominant phenotype
* recessivePhenotype: String representing the recessive phenotype
* phenotypes: Array of Strings with each unique phenotype

Strain:

* name: String with the name of the Strain (e.g. “Pink-1”)
* n: Int representing which number strain this is
* phenotype: String representing which phenotype this strain is
* partialConclusions: Array of Strings with the partial conclusions inferred about this strain (not necessarily entered yet)
* needToWritePartialConclusion: Boolean representing whether the tutor expects the student to enter a partial conclusion for this strain
* finalConclusion: String representing the final conclusion inferred about this strain (not necessarily entered yet)
* iePartialConclusions: name of the interfaceElement where user will enter partial conclusion
* ieFinalConclusion: name of the interfaceElement where user will enter final conclusion

Cross:

* name: String with name of cross
* Strains: Array of Strings which are the names of the strains in the cross
* offspringExt: String representing the resulting offspring of a cross (e.g. “100% Pink”)
* offspringInt: Array of Ints and Strings which represent the offspring (4 entries, with the % of offspring that were one phenotype as an Integer, then the String name of that phenotype, then the Int % and String name for the next phenotype e.g. [50, "pink", 50, "blue"])

interfaceElement

* name: String with the ID of the corresponding interfaceElement
* value: String with the value stored in the interface element

interfaceRow

* name: String with a name for the row
* ieStrains: String with the ID’s of the interface elements storing the Strain names
* ieResult: String with the ID of the text box that displays the offspring results

# Rules

Bootstrap:

* At the beginning of a problem: initialize the problem, strain, cross, interfaceElements, and interfaceRow facts

selectFirstStrain:

* If there is a Problem fact such that its current subgoal is “selectCross” and its first strain of the next cross has not yet been selected
* AND there is a strain (called s1)
* AND there is a cross (called c) such that the name of s1 is in the cross’s strain list
* AND there is an interfaceRow whose name is the first element of the problem’s interfaceRows list
* And there is an interfaceElement with no value set with its name in the interfaceRow’s ieStrains
  + THEN:
  + Update problem’s firstStrainCurCross to the strain name
  + Update the interfaceElement’s value to the strain name

selectSecondStrainDiff:

* If there is a Problem with its current subgoal as “selectCross” and its firstStrainCurrCross is set and the Problem has interfaceRows
* AND there is is a strain with a different name than the first strain from the current cross (call it s2)
* AND there is a Cross with strains matching the first strain from the problem’s current cross and s2
* And there is an interfaceRow whose name is the same as the first entry in the Problem’s interfaceRows
* And there are two interfaceElements with names corresponding to the row and one of them doesn’t have a value
  + THEN
  + Update the value of the blank interfaceElement and reveal the results of the cross with a TPA, then lock the text field with those cross results
  + Update the problem:
    - Set its fisrtStrainCurCross to null
    - Remove the interfaceRow used in this rule
    - Set the curCross to the cross matched on the left hand side
    - Change the subgoal to “inferFromCross”

selectSecondStrainSame

* Same as selectStrainDiff, but the names of the two strains must be different

sameStrain100

* If there is a Problem with subgoal “inferFromCross”
* AND there is a cross that is a self-cross that matches the problem’s curCross
* AND there is a strain with a name that matches the cross
  + THEN
  + Infer a partial conclusion for the strain that says the strain is homozygous
  + If this is the first partial conclusion for the strain, then write that the strain needsToWritePartialConclusion
  + Regardless, retract the cross, nullify the problem’s curCross, and change the subgoal to “makeInferences”

sameStrain75

* If there is a problem with subgoal “inferFromCross
* AND there is a self-cross that has offspring with 75% one phenotype and 25% another phenotype
* AND there is a strain with the same name as the one in the cross:
  + THEN
  + Infer a final conclusion that the strain is heterozygous
  + Infer that the dominant phenotype is the same as the strain
  + Infer that the recessive phenotype is the same as the one taken by 25% of the offsprring
  + Set the curCross to null
  + Retract the cross
  + Change the subgoal to “makeInferences”

diffStrainSamePheno75

* If there is a Problem with subgoal “inferFromCross”
* AND there is a cross of two different strains with the same phenotype and the resulting offspring have 75% of one phenotype and 25% of another phenotype
* AND there are two strains with names matching the two strains in the cross
  + THEN
  + Infer final conclusions for both strains that they are heterozygous
  + Infer that the dominant phenotype is the same as the strains in the cross
  + Infer that the recessive phenotype is the same as 25% of the offspring
  + Set the curCross to null
  + Retract the cross
  + Change the subgoal to “makeInferences”

diffStrainDiffPheno100

* If there is a problem with subgoal “inferFromCross”
* AND there is a cross of two different strains with different phenotypes and 100% of offspring are all one phenotype
* AND there are two strains with names matching the strains in the cross
  + THEN
  + Infer final conclusions for the strains: the strain whose phenotype is the same as the offspring is homozygous dominant and the other strain is homozygous recessive
  + Infer that the dominant phenotype is the same as the offspring
  + Infer that the recessive phenotype is the same as the homozygous recessive strain
  + Set the curCross to null
  + Retract the cross
  + Change the subgoal to “makeInferences”

diffStrainSamePheno50

* If there is a problem with subgoal “inferFromCross”
* AND there is a cross with two different strains with the different phenotypes with offspring 50% one pheontype and 50% the other phenotype
* AND there are two strains with names that match the names from the cross
  + THEN
  + Infer partial conclusions for both strains that they are either homozygous recessive or heterozygous
  + If either (inclusive) of the strains have not have a partial conclusion previously inferred then set that strain’s needToWritePartialConclusion to true
  + Set the curCross to null
  + Retract the cross
  + Change the subgoal to “makeInferences”

diffStrainSamePheno100

* If there is a problem with subgoal “inferFromCross”
* AND there is a cross of two different strains with the same phenotype and all offspring are the same phenotype
* AND there are two strains with names that match the cross
  + THEN (no inference to be made)
  + Set the curCross to null
  + Retract the cross
  + Change the subgoal to “makeInferences”

makeInferences

* If there is a problem with subgoal “inferFromCross” and we have inferred the dominant phenotype (it is assumed that we will then have inferred the recessive phenotype)
* AND there is a strain without a final conclusion written that has a partial conclusion
* THEN
  + If the strain’s partial conclusion homozygous:
    - Then infer a final conclusion that the strain is homozygous dominant if its phenotype matches the dominant phenotype, and homozygous dominant if its phenotype matches the recessive phenotype
  + If the strain’s partial conclusion is homozygous recessive or heterozygous:
    - Then infer a partial conclusion that the strain is heterozygous if its phenotype matches the dominant phenotype, and homozygous recessive if the strain’s phenotype matches the recessive phenotype

makeAdditionalInference

* If there is a Problem with subgoal “makeInfereneces” and we know the recessive phenotype
* AND there is a strain that has the phenotype but does not have a final conclusion inferred
  + THEN
  + Infer a final conclusion for this strain that it is homozygous recessive

inferFinalFromBothPartials

* If there is a problem with subgoal “makeInferences”
* AND there is a strain without a finalConclusion that has both partial conclusions inferred
  + THEN
    - Infer a final conclusion that the strain is homozygous recessive

doneInferencing

* If there is a problem with subgoal “makeInferences” and we have no other matching rules (low salience)
  + THEN
  + Set the subgoal to “enterInferences”

enterFinalInference

* If there is a problem with subgoal “enterInferences”
* AND there is a Strain with a final conclusion that has not been written
  + THEN write the inferred final conclusion

enterPartialInference

* If there is a problem with subgoal “enterInferences”
* AND there is a strain for which we need write its partial conclusion
  + THEN
  + Write its partial conclusion

doneWithMandatoryInput

* If there is a problem with subgoal “enteringInferences”
* AND there are no strains with final conclusions inferred but not written
* AND there are no strains with partial conclusions inferred but not written
* AND there are no other rule activations to try (low salience)
  + THEN
  + Change the subgoal to “selectCross”

Done

* IF there are no strains that don’t have a final conclusions written
  + THEN
  + Press the done button to complete the problem

# Answers to Questions on Working Memory

1. Working memory represents the subgoals as the problem’s subgoal attribute
2. The inferences are modeled as strings stored in each strain’s finalConclusions (stores one string), and partialConclusions (list of potentially multiple conclusions). These values are populated by the rules which fire during the “inferFromCross” and “makeInferences” subgoals.
3. Cross facts represent the two strains that were crossed to make the inference, including their offspring. This is helpful for matching the appropriate strains on the lefthand side, and for using the cross’ offspringInt to check the resulting offspring so that we can infer which of the cases in mendelBackwards matrix is represented by the cross on the rule’s lefthand side.
4. The model represents whether an inference has been made by storing it in a strain’s finalConclusion slot, or adding the inference to the strain’s partialConclusions array. The boolean values finalConclusionWritten and needToWritePartialConclusion are used to determine whether the final conclusion has been written, and whether there is a partial conclusion that needs to be written.
5. Making this distinction allows the model to generate all the inferences before entering any of them, and then to enter all the inferences in one subgoal. It also facilitates differentiating between the student’s thinking and actions in the cognitive model.
6. Interface elements are organized in individual interfaceElement facts, which are grouped in interfaceRow facts, which represent one row of several inputs for a cross by storing the names of those elements (the two strains, and a third TPA controlled text box for displaying the offspring results. Inference comboBoxes represented with interfaceElements that are not grouped into rows. The interfaceElements for crossing are referenced in the problemFact by storing the names of the interfaceRows in the problem’s interfaceRows attribute.
7. The offspringExt and offspringInt attributes of a cross both represent the information about its offspring (what % are each phenotype) but the Ext version is a string used for displaying to the user (via TPA) and the Int version is an array used for back-end calculations (so we don’t need to parse the string version to make calculations).

# Strengths and Weaknesses

Strengths: Our tutor is powerful and flexible. It allows students to take any reasonable path through the problem by making any permutation of crosses and making all the relevant conclusions.

Weaknesses: Our tutor offers very little scaffolding. There are no hints, and no feedback is given to explain right or wrong answers. In particular, it will likely be difficult for students to tell what the current subgoal is supposed to be and consequently whether they are supposed to make crosses or make inferences. Students would likely need significant coaching on use of the tutor, in addition to the actual processing of performing abductive reasoning on genetics problems to use this tutor, and even then, without knowing *why* their answers were wrong, it would be difficult for struggling students to learn from the tutor.

# Teamwork

Our team worked very productively together. Both of us wrote rules, and we performed the theoretical CTA together. We discussed our plans together before implementing our solutions, and did most troubleshooting together. We both have backgrounds in education and software development, so we were very well rounded.

# Challenges

On the task domain side, it was difficult to comprehend the inferences made by analyzing partial conclusions after the inferences that have been made from one cross have been made. In particular, we had difficulty recognizing that the case where one strain gets both partial conclusions added was even possible (we ran into this by accident and then realized that it caused a fatal bug that trapped you in the “enterInferences” subgoal without the ability to enter the inference) and then that in the case where the partial conclusions were different that you could make a final inference. We also nearly missed that if you know which phenotype is dominant and recessive, and you have a strain that has the recessive phenotype, that you can conclude that this strain is homozygous recessive, even if you’ve never used it in a cross. Both these situations required some outside the box thinking to catch as problems, and to solve.

On the technical implementation side, understanding the intended flow between rules was difficult, and especially the use of halt(). It was confusing to parse the conflict tree and understand where in the complex chains that halt was needed. We wound up using a somewhat strange implementation where halts() are called in the rules for entering inferences, but not in the doneWithMandatoryInput rule, so that doneWithMandatoryInput will flow right into selectFirstStrain.

# CTAT Notes

It was cool to finally develop a tutor that *really* required rules to be sufficiently robust. One annoying and bewildering bug that came up was that the globals used to define the strains in the problem files polluted the namespace, so when trying to refer to the strains in the lefthand side of rules by the name strain1, strain2, etc., very strange effects happened. This was difficult to detect, because we don’t often think to look back at the problem files, since the action is distributed in productionrules.nools and types.nools.

# New Problem: Green (and Yellow) beans

Mendel did experiments with beans, and this source with similar genetics problems used bean color as one such example:

<https://studyres.com/doc/14922486/genetics-practice-problems-packet>