1. Trivial.

2. Trivial. For both this problem and the previous, you have the option of either giving truck-loads of data or tweaking the parameters (e.g., \textit{inflate}).

3. This problem is impossible unless the ILP software allows you to significantly alter its default search algorithm. If your answer was ‘I tried it but couldn’t get it to work’ no credit would be given for your answer. The scientific way to explain the difficulty is to state that: (i) for association rules with the same consequent, only one of them may be produced by an ILP algorithm, since the search is performed via a sequential covering algorithm, and (ii) we need a simultaneous covering algorithm to be able to mine all rules. Later versions of PROGOL (notably the Aleph software) provide this functionality.

4. The only two possible legal solutions are:

   \begin{verbatim}
   willgraduate(X) :- hastaken(X,CS3414), hastaken(X,CS4604), hastaken(X,CS5604).
   willgraduate(X) :- hastaken(X,CS3414), hastaken(X,CS4604), hastaken(X,CS5614).
   willgraduate(X) :- hastaken(X,CS3414), hastaken(X,CS4604), hastaken(X,CS5804).
   ...
   willgraduate(X) :- hastaken(X,CS5614), hastaken(X,CS5804), hastaken(X,CS6604).
   \end{verbatim}

   and

   \begin{verbatim}
   willgraduate(X) :- hastaken(X,A), hastaken(X,B), hastaken(X,C),
   \hspace{1cm} A<>B, A<>C, B<>C.
   \end{verbatim}

   If you produced the second solution, you \textit{must} state the catch that the rule only holds when the domain of \(A, B,\) and \(C\) is restricted to the given set of courses. For this reason, the first rule is better as it probably was mined having other courses in the background knowledge. This exercise was meant to highlight the lesson that the learned rules will be cumbersome to read (and process), and may have poor generalization unless you have a good starting set of features. For instance, if the ILP software could constructively induce the feature ‘at least \(m\) of’ from the given features, then the solution would be elegant.

5. Trivial. You might have noticed that a clever tweaking of the ILP algorithm’s parameters (e.g., \(i, c\)) is beneficial for minimizing the length of the hypothesized rules and limiting the number of unifications.

6. The only way to learn this pattern is to learn what makes a ‘non prime’ and then recast prime numbers as the negation of this concept. A set is sometimes easier to define by its non-members:

   \begin{verbatim}
   notprime(A) :- number(A), divisor(A,B), B<>1, B<>A.
   prime(A) :- number(A), not notprime(A).
   \end{verbatim}

   If your rule was something like:

   \begin{verbatim}
   prime(A) :- numberofdivisors(A,2).
   \end{verbatim}

   you will get zero points, because you were explicitly forbidden to make the problem so simple. If you only learned a rule for \textit{notprime} but didn’t bother to (or notice that you can) learn \textit{prime} from \textit{nonprime}, you will get partial credit.