Question 1.

1. i.  
    p([1,3,6,4], X).  
   Call p(1, [3, 6, 4], X).  
   // is 3 < 1? no , take next rule   
   Call p(1, [6, 4], X)  
   // is 6 < 1? no , take next rule  
   Call p(1, [4], X)  
   // is 4 < 1? No, take next rule  
   Call p(1, [], X)  
   // reach terminate point, Unify X with 1  
   X = 1?   
   Yes  
   Exit all open calls.  
   ii.   
    p(L, 9).  
   Call p(L, [], 9)  
   Exit p(9, [], 9) // reach base case unify L with 9  
   Exit p([9], 9).
2. 1. r(4).   
      Call r(4)   
      Asert to end of your database p(4) // assertz works from end   
      Call r(3)   
      Assert to end of db p(3)  
      Call r(2)  
      Assert to end of db p(2)  
      Call r(1)  
      Assert to end of db p(1)  
      Call r(0). // base case   
      Exit all open cases, and return yes, but we have in our database p(1), p(2), p(3), p(4) as set of facts
   2. r(4).   
      Call r(4)   
      Asert to end of your database p(4) // assertz works from end   
      Call r(3)   
      Assert to end of db p(3)  
      Call r(2)  
      Assert to end of db p(2)  
      Call r(1)  
      Assert to end of db p(1)  
      Call r(0).   
      Assert to end of db p(0)  
      Call r(-1) and now will be stuck in an infinite loop as we will never reach base case.
3. 1. call(q([agree=thirdsing, cat=np, definite=yes], [case=subject, cat=D], L).  
      Check if agree=thirdsing @< case=subject, // yes  
      call(q([cat=np, definite=yes], [case=subject, cat=D], L).  
      Check if cat=np @< case=subjet  
      Fail above check  
      Redo call(q([agree=thirdsing, cat=np, definite=yes], [case=subject, cat=D], L). but now taking next rule.  
      Check if agree=thirdsing @< case=subject, // yes  
      Call (q([agree=thirdsing, cat=np, definite=yes], [cat=D], T)   
      // Current stack for L = [case=subject | T]   
      Check agree=thirdsing @< cat= D // yes  
      Call (q([cat=np, definite=yes], [cat=D], T) // since cat and cat are unifiable   
      // current stack for L = [case=subject, agree=thirdsing, cat=np, T]  
      Call (q[definite=yes], [], T) // base case  
      Return from previous call (q[definite=yes], [], definite=yes)  
      Now return from all open calls   
      End result   
      L = [case=subject, agree=thirdsing, cat=np, definite=yes]  
      D = np
   2. q([agree=thirdsing, cat=np, definite=yes], [cat=D, case=subject], L).  
      Call q([agree=thirdsing, cat=np, definite=yes], [cat=D, case=subject], [agree=thirdsing | T).  
      Check agree=thirdsing @< cat=D // yes  
      Call q([cat=np, definite=yes], [cat=D, case=subject], T).  
      // L stack = [agree=thirdsing, cat=np, T]  
      // we have np both places so we can use 2nd rule  
      Call q([definite=yes], [case=subject], T)  
      Check definite=yes @< case=subject  
      Fail   
      Redo q([definite=yes], [case=subject], T) // next rule  
      Check definite=yes @< case=subject  
      Fail   
      Redo Call q([cat=np, definite=yes], [cat=D, case=subject], T).  
      Check cat=np @< cat=D  
      Fail  
      Redo and fail again   
      Keep redoing and failing until we are back to agree=thirdsing   
      Call q([agree=thirdsing, cat=np, definite=yes], [cat=D, case=subject], [cat=D | T).  
      agree=thirdsing @< cat=D // yes  
      Call q([agree=thirdsing, cat=np, definite=yes], [case=subject], T).  
      agree=thirding @< case=subject // yes  
      But will go down a route which will compare cat to case and fail so I am skipping that and instead will show that we are trying next rule  
      Call q([agree=thirdsing, cat=np, definite=yes], [case=subject], [case=subject| T]).  
      agree=thirdsing @< case=subject // yes  
      Call q([agree=thirdsing, cat=np, definite=yes], [], T).  
      // reaching base cases and exiting from open calls will have this stack  
        
      L = [cat=D, case=subject, agree=thirdsing, cat=np, definite= yes).  
        
      SCREW THOSE 4 MARKS
   3. I have no idea, might blurb something about it enforce certain order

Question 2.

1. Context-free grammar tries to work out how to match rules of the right hand side of your rules, without taking into account the surrounding items.

Feature-based grammar allow us to attach more information to our grammar such as tenses, position, form, agreement, etc… it allow us to model complex sentences in more powerful way that represent the language accurately   
  
Example: She love he. He isn’t accounting for its position in the sentence that it came as an object and therefore should be ‘him’

1. Top-down: I have a rule, I will see if I can get the things needed to make this rule, do this for all rules until I reach a leaf (lexical edge)  
   Bottom-up: I have bunch of lexical edges, do I have the right hand side of any rule? If so replace it by left-hand side.   
     
     
   “I know it died”  
     
   Top down:  
   SOMETHIGN ALONG THESE LINES   
   Adding edge (0, 1, s, [np], [vp], 0)   
   Adding edge (1, 1, [np], [pronoun], 1) because matched start of edge (0, 1, s, np, vp)   
   Adding lexical edge(2, 1, pronoun, i, 2) because matched edge (1, 1, np, pronoun, 1)  
   Adding edge (1, 2, [vp], [verb], 3) because of edge (0, 1, s, [i], [vp], 0)  
   Adding lexical edge (2, 2, verb, know, 4) because of edge (1, 2, [vp], [verb], 3)   
   Adding edge (1, 2, vp, know, 5)   
   Adding edge (0, 1, s, i, know)   
   Fail and redo from beginning  
   Adding edge (0, 1, s, [np], [vp], 0)   
   Adding edge (1, 1, [np], [pronoun], 1) because matched start of edge (0, 1, s, np, vp)   
   Adding lexical edge(2, 1, pronoun, i, 2) because matched edge (1, 1, np, pronoun, 1)  
   Adding edge (1, 2, [vp], [verb], 3) because of edge (0, 1, s, [i], [vp], 0)  
   Adding lexical edge (2, 2, verb, know, 4) because of edge (1, 2, [vp], [verb], 3)   
   Adding edge (1, 2, vp, know, [s], 5) because of edge Adding lexical edge (2, 2, verb, know, 4)   
   Adding edge (1, 3, s, [np], [vp], 6) because matched edge (1, 2, vp, know, [s], 5)  
   Adding edge(2, 3, np, [prounoun], 7) because matched edge (1, 3, s, [np], [vp], 6)   
   Adding lexical edge (3, 3, pronoun, it, 8) because matched edge (2, 3, np, [prounoun], 7)  
   Adding edge(1, 4, vp, [verb], 9) because matched edge (1, 3, s, it, [vp], 6)  
    Adding lexical edge (2, 4, verb, died, 10) because matched edge(1, 4, vp, [verb], 9)  
   // there are 5 steps here that will match each of the loose edge with its root, at the end you will get   
   Adding edge (0, 1, i, know, it died, 16) because matched edge (1, 2, i, know, [s])  
     
   SCREW THOSE 10 MARKS
2. Fundamental rule of chart parsing: if you’ve got an X that needs [H | T] to make a complete X, you can combine them to make an X that needs T to complete itself.  
   This is used to save the trace route I have already tried so I don’t have to retrace back as I would do with bottom up. (Dynamic programming)  
     
   FK THE STEPS :( it is 1am and I am tired
3. Because of long distance dependency, if we only have rewrite rules we will have to backtrack a lot (failing and trying another rule) which will grow exponentially with number of rewrite rules we have, and become difficult to handle.

Question 3:

1. Providing a precise ways of specifying rules and facts and model complex sentences with simple logic.
2. Particular set of words, configured in specific way, always make the same contribution, though what you o with that contribution in a given context is up to you.  
     
    A. Structure ambiguity. I don’t know what the mode of combination is.   
   B. Lexical ambiguity. I don’t know what the parts are. Do you mean a jam as in traffic, or a jam as the glass jam, or jam as a party?   
   C. Scope ambiguity. I know the words, I know the structure but I still don’t know the meaning. Do I want to buy any type of jar of jam? Or is there a specific one I have in mind?
3. I have answered this on y1+1, the answer is bit long so I CBA sorry :(   
     
   Significance of term patient is: it provide the parser with some contextual knowledge; now we know that the effect of verb dies will fall on man.

Question 4:

1. A. here we have time in the past “entered” which isn’t in FOL  
   B. Using a “most powerful”, is a relative adjective, this will require us a knowledge of every person in the world, then we somehow measure their power and model that in FOL is impossible.   
   C. Here we have time in past “saw” which in’t in FOL
2. NOTE While he bound the man to a reference, it still doesn’t uniquely identify the man, depending on context this might be false.
3. Minutes work as a common ground between the speaker’s view of the minutes and mine, the rule had to specify that the minutes are from the speaker’s otherwise ti could have been the hearer or anyone else.   
   Now with the introduction of minutes, if we know a specific minute, we can deduce that in that minute there was a man, and that man entered a shop, the shop is part of the same context. Now we uniquely identified a man and a shop if we are given the correct context (minutes)
4. No idea

Question 5.

1. First order logic with no disjunction in the consequent is called Horn-clause logic.  
     
   It can’t model complex statements such as having disjunction in the consequent:  
   Something => dothis or that  
   Can’t deal with bizre stu
2. COPY PASTA FROM Y1+1: I would suggest looking at exam 2013 for a clearer answer  
   we will call using

prove(p => q or r => s)

which will call prove(P)

That will invoke the term P // aka what we called it trying to prove

P => Q (in our case this is p=> (q or r) => s)

P => Q will assert that p is true and tries to prove the reset of the terms

so will call prove(Q) // prove((q or r) => s))

This will call prove(P)

which will invoke the term =>

call ing P => Q // (q or r) => s

will assert that (q or r) is true in our database, next we try to prove s

prove(s) calling s will result in a failure since it isn't saved in our database

so we will call the next prove(P)

which tries to find R or S that brings our P "in this case s" to true.

we have q or r

so we prove that q or r holds (it is in our database)

then we do a call on r => s,

which will take us to last term

assert r in the database and prove s

then tries to prove q => s, which again will take us to last term and do its magic there.

1. It is constructive implication because we add left hand side of implication to our set of facts to see if we can prove right side. This is different from material implication where we replace the implication with negation and disjunction.  
     
   Stuff like: If the pigs fly I am the queen of England, would result in true in material implication but false in constructive one.
2. Semi-decidable means it that the program might not halt for some predicates, in first order logic it is impossible to tell for an arbitrary formula A is in formula B or not.   
   p(X) => q(X).  
   q(X) => p(X).  
   Here we will keep trying to prove right from proving left.   
   One way to solve it is to have labels which monitor all the things I have seen and halt if the current prove is trying to prove what is already in the label after X amount of loops.