6.871 SPRING 2004
End of Term Talks
Term Papers

The outline below can be used to organize both your term paper and your presentation at the end of the course. Not all of these questions will be equally relevant to every project, but you should make a serious effort to address all of them.

The most fundamental task is to make clear the answer to four central questions about your system:

1. What task does it do?
2. What is the problem solving paradigm, and why does it match your problem?
3. What does it *know* that enables it to do perform its task?
4. What did you learn in designing and building the system?

Note very carefully that the third question — "What does it know?" — is a very different question from "How does it work?" Imagine answering both of them for the Mycin system, for instance. What it *knows* has to do with medicine, organisms, drugs, etc. How it *works* has to do with rules, backward chaining, abstracting specific symptoms into symptom classes, etc.

Do not answer "What does it know?" by describing code or data structures. That is the answer to a different question.

The outline below provides a specific set of questions to address in both your talks and your papers. You may use this as a template for both, but if you decide to organize your talk or paper differently, be *sure* that you are addressing these questions explicitly.

What task does your system do?

- Answer in general first, then give a *real, specific,* <u>*completely concrete*</u> example of its input and its output.
  EG: Diagnoses infectious disease, given basic data about the patient.
  Specifically, for Jane Smith, a 25 year old female with a positive blood culture containing gram negative rods, . . . the system indicated that the organism might be a. . .

Walk through one real, annotated problem.

- That is, show the actual interaction with the user and intersperse explanatory comments so that we know what the system is doing and why, where the performance is interesting, innovative, awkward, dumb, etc.

What range of problems can the system handle?

- Characterize as best you can what kinds of problems the system can handle. Describe the ones it has actually solved and then describe what it can do beyond these. This is not always easy to do, so don't be surprised if this takes some careful thought; it is however an important exercise and an important thing to know.
- Describe a problem that your system can't handle, but is quite close to the sort it can deal with. That is, what relatively small change in the problem would make it beyond the ability of your system in an interesting way.

What does it know?
Show a complete tree (or other structure) of the system's knowledge base. Describe how your system represents the knowledge.

- For a backward chaining rule-based system, it is natural and informative to lay out the knowledge base as a tree structure with the overall goal at the top. It is enough to show which attributes are used to infer values for which other attributes.
  Use this or any other convenient and appropriate mechanism to display the system's entire knowledge base in a comprehensible fashion.

How does it work?

- Describe the overall approach to the problem, giving details of your system's problem solving strategy. Some of this may have been made clear by the annotated example, but try to address the question directly, even if it means repeating some material.

What went well?

- What did the system do well? How were you well-served by the knowledge representation and tools that you used? That is, what kinds of knowledge were easily captured? What kinds of reasoning were easily captured?

What went badly?

- What did the system do badly? How were you ill-served by the knowledge representation and tools that you used? What kinds of knowledge were not easily captured? Where did you have to force it? What kinds of reasoning were not easily captured? For both the knowledge and reasoning that were not easily captured, how else might/should you have proceeded?

What else did you learn from this?

Note that the most important thing is not whether you got the system to work in some spectacular fashion (though that is always good to see), but whether you learned something interesting from your efforts. You may have discovered for instance how badly rules capture the sort of reasoning you wanted to do. If so, be specific and technical about this. What didn't they do? What did you want to accomplish? What would you do differently next time and why do you think that would help?

Presentations should be *no more than 15 minutes*, **so aim for 12.**

Papers have no upper bound in length, but **must** include a few real rules (or frames, or whatever) in the body of the paper, and **must** have a *complete listing of the knowledge base* in an appendix. Do not put any extensive segment of code or rules into the body of the paper; you should describe the program in words, without reference to the code.

Remember that while presentations can be shared, *papers must be written independently.* You will be describing the same project, so some overlap in description (and the appendix) is to be expected, but you cannot turn in one paper between you or even share text between papers.

Also, given that you have known about the deadline for the final paper and project – the end of the last class – since the very first day of class this term, I am not inclined to grant extensions except for the most unanticipated of events.