DCS 440, Artificial Intelligence, Midterm 1, 1999
80 minutes, open book, open notes.
This exam has three problems on eight single-sided pages.
Answer each question in the space provided.
Use backs for further work but indicate when you do.

1. Unix systems often require you to create useful files in several steps. For example, the specification of this test was originally stored as text in a `tex` file. The printed version was created by first running a program called `latex` on the `tex` file to compute a description of where letters belong on a page; the description was stored in a new file, the `dvi` file. A program for the printer was constructed from this description using the command `dvips`; the printer program was stored in a `ps` file. Finally, another command, `lpr`, executed the printer program to create the master copy of the test. A common result is that your unix directories become cluttered with files that you do not need because you have used them for what they were created for and moreover you could make them again if you needed them again.

Imagine creating a software agent that would traverse your unix files and clean up after you. Here is a description of what the agent might do. It goes into a directory called `work` and finds that there are three files there: one called `report.tex`, one called `report.dvi` and one called `report.ps`. It concludes that the two files `report.dvi` and `report.ps` were stored temporary files that could be recreated from `report.tex` and deletes them.

[Questions begin on next page]

1a. Suppose we use the following kinds of objects to represent this situation. We have *files* themselves, *stems* which name files and *extensions* which give the kind of data stored in the file. Introduce enough symbols of each of these types to describe the contents of the `work` directory—three files: one called `report.tex`, one called `report.dvi` and one called `report.ps`)—by listing those symbols and their intended interpretations.

Symbol	Interpretation
f1	the file called `report.tex`
f2	the file called `report.dvi`
f3	the file called `report.ps`
rep	the stem `report`
tex	the extension `.tex`
dvi	the extension `.dvi`
ps	the extension `.ps`

1b. Suppose we have two binary relations `stem(F,S)` true when the name of file `F` begins with stem `S` and `extension(F,E)` true when the name of file `F` ends with extension `E`. Write atomic facts using these relations to describe the contents of the `work` directory.

```
stem(f1,rep).  extension(f1,tex).
stem(f2,rep).  extension(f2,dvi).
stem(f3,rep).   extension(f3,ps).
```

1c. Now we define a binary relation `derivable(I,O)` that is true if there is a program that computes file `O` as output given file `I` as input. Give two clauses for `derivable(I,O)` that apply in our scenario, in terms of the kinds of objects and relations already defined.

```
derivable(I,O) :-
                  stem(I,S), stem(O,S)
                  extension(I,tex), extension(O,dvi).

derivable(I,O) :-
                  stem(I,S), stem(O,S)
                  extension(I,dvi), extension(O,ps).
```

1d. Define a unary relation `delete(F)` that is true if a file should be deleted because it can be recreated automatically when needed.

```
delete(F) :- derivable(X,F).
```

1e. Consider two ways for our file-cleaning agent to make its decisions. In the first way, the agent repeatedly finds a file `f` for which `delete(F)` is true, deletes it, and reexamines the directory to assess the progress it has made. In the second way, the agent first finds all the files for which `delete(F)` is true and then deletes each in turn. Is there a reason to prefer one over another?

> Using the rules suggested as answers for problems 1c and 1d, only the second strategy will be successful in deleting all temporary files. The first strategy may delete an intermediate temporary file and then decide that files derived from it need to be preserved.
> Other considerations: if the directory is changing quickly and the agent must constantly strive to keep it free of temporary files, something like the first strategy may be necessary; otherwise the first strategy will require more consultation of the directory and more querying of the database.

1f. This example makes a strong assumption. Discuss this assumption and its consequences with respect to the following scenario. Sandy gets email from Robin of the text of a paper they are collaborating on, and saves it in a file called `robin.tex`. Later, Robin sends Sandy an output file of the first chapter of a novel to read for fun; Sandy saves that as `robin.dvi`. Is it possible to represent this situation using the kinds of objects and relations described above? Indicate which of the clauses you wrote in **1c** and **1d** are true and which are false in this scenario.

> The assumption is that all files with the same stem represent the same collection of data; thus you can tell whether a file has been derived when it shares a stem with a source file and has a related extension (as in the answer to 1c). This would lead to the `robin.dvi` file being deleted (incorrectly) in this scenario.
> The scenario still involves files, stems and extensions so the right objects and relations are available. But now the first rule (deriving dvi files from tex files) is false because there is a case that satisfies the body of the rule but not the head. The second rule is true because no cases apply to it. The third rule is true – and in fact can continue to be regarded as true even in the more general scenario – if a file truly is derivable then it continues to be a good idea to delete it.

2a. Define a relation `prefix(S,L)` true if list `L` begins with a copy of list `S` (that is, with the same elements in the same order) but continues by possibly having additional elements. (Do not appeal to any other relations in writing this definition.)

```
prefix([],L).
prefix([A|S],[A|L]) :-
                        prefix(S,L).
```

2b. Using this definition (if possible), provide the answer(s) to the two queries that follow. Give the value for `S` as the answer in all cases; pick two answers and also describe the derivations or proofs of those answers in any convenient notation (e.g., the full proof notation, the abbreviated notation from class, the notation of SLD derivations from the text).

- `?prefix(S,[a,b])`

```
           S = [] (match 1)
S = []     ?prefix(S,[a,b])

           S1 = [] (match 1)
           ?prefix(S1,[b])
           S = [a|S1] (match 2)
S = [a]    prefix(S,[a,b])

           S2 = [] (match 1)
           ?prefix(S2,[])
           S1 = [b|S2] (match 2)
           ?prefix(S1,[b])
           S = [a|S1] (match 2)
S = [a,b]  prefix(S,[a,b])
```

- `?prefix([a,b],S)`

```
                       S2 unconstrained (match 1)
                       ?prefix([],S2)
                       S1 = [b|S2] (match 2)
                       ?prefix([b],S1)
                       S = [a|S1] (match 2)
S = [a,b|S2]           prefix([a,b],S)
```

2c. Recall that the prolog proof search space for a query can be infinite if there is an increasing series of incomplete proofs, and given one of these incomplete proofs, you can always apply a rule to obtain another of them. Otherwise—if the prolog search space is finite—prolog can compute all answers to a query and terminate.

If `S` is a ground term, can the prolog proof search space for `?prefix(S,L)` be infinite or will prolog always terminate after finding all answers?

At each step of backward chaining to build the proof, the first argument of `prefix` is reduced to a smaller ground term. So the proof search space is finite and Prolog always terminates.

If `L` is a ground term, can the prolog proof search space for `?prefix(S,L)` be infinite or will prolog always terminate after finding all answers?

At each step of backward chaining to build the proof, the first argument of `prefix` is reduced to a smaller ground term. So the proof search space is finite and Prolog always terminates.

3 Great-uncle Jasper has bequeathed his coin collection to Sandy and Robin. It contains just five coins; their values are \$5, \$7, \$8, \$10 and \$16, for a total of \$46. Sandy and Robin decide that their two shares of the collection should have equal value. They have to figure out who gets what—it is (in part) a search problem.

The search space for this problem—like any other search problem—consists of a set of states, a start state, a set of goal states, and a way of going from one state to another. Let's define this as follows:

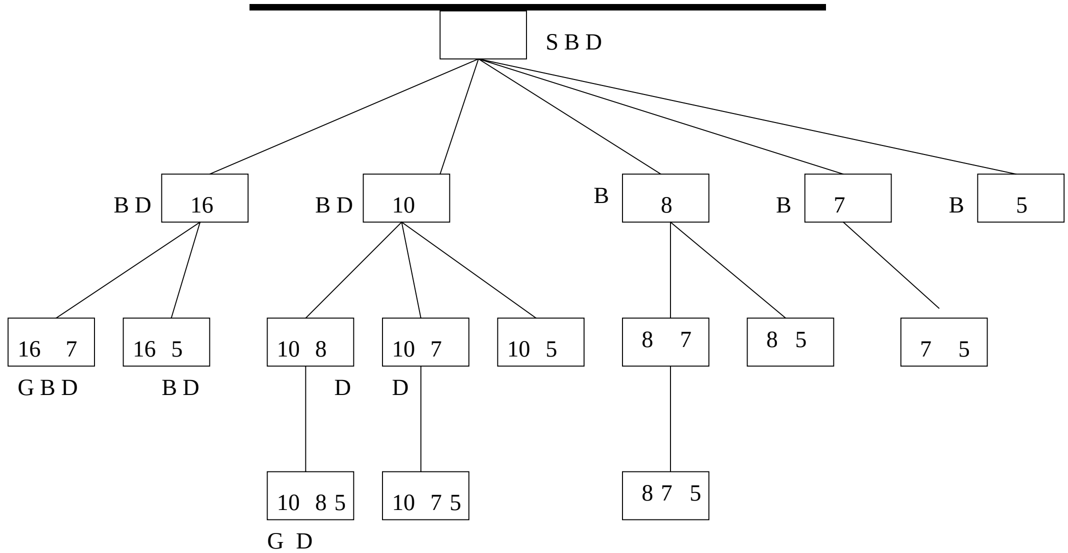
- A state consists of a set of coins to be given to Sandy. Since Sandy gets no more than half, the value of this set must be \$23 or less. Write a state as a list of values; for instance a state where Sandy gets the first two coins is:

$$\boxed{5, 7}$$

- In the start state, Sandy gets no coins.
- A goal state is a set of coins whose value is \$23.
- You go from one state to another by adding to Sandy's take another coin that Sandy hasn't been given yet and that is not worth more than any coin Sandy already has. (In other words, Sandy gets more valuable coins before less valuable ones.)

[Questions begin on next page]

3a Draw the search space Sandy and Robin have in dividing Jasper's coins (worth \$5, \$7, \$8, \$10 and \$16), according to this definition. Leave plenty of room beside each node to answer the remaining questions.



3b Label the start state with an S and label any goal states with a G.

3c Assuming neighbors are added to the search frontier in top to bottom/left to right order (according to your diagram), use a B to label the first eight nodes expanded in breadth-first search. Use a D to label the first eight nodes expanded in depth-first search.

3d Describe informally what would happen if we change the set of neighbors, so that we could give Sandy *any* other coin at any step. Would the number of goal states change? Would the search problem be easier or harder? Explain.

Let's suppose that we continue to represent a state by the list of coins assigned in order. Then the number of states and the number of paths will both explode with this change. This will tend to make this kind of search problem more difficult, especially in cases where there are few or no solutions and more of the space has to be explored in order to obtain an answer. But the solutions found with the expanded space will not change—any list of coins can be sorted to find a corresponding solution in the original search space.