University of Arizona, Department of Computer Science

CSc 372 — Assignment 5 — Due noon, Thu Oct 27 — 5%

Christian Collberg
October 18, 2011

1 Introduction

The purpose of this assignment is for you to get familiar with list manipulation in Prolog, as well as learning some common Prolog programming patterns, such as *generate-and-test*.

The following rules apply for this assignment:

1. You may freely introduce auxiliary predicates if that makes your program cleaner.
2. You will be graded primarily on **correctness**, **elegance**, **clarity**, and **documentation**.
3. Every predicate should be commented. At the very least, the comments should state what the predicate does, which arguments it takes, and what result it produces.
4. You may freely use standard Prolog list-manipulation predicates such as `member`, `append`, and `delete`.
5. This assignment is graded out of 100. It is worth 5% of your final grade.
6. Input files to the programs in this assignment can be found on the course website.
7. **Unless specifically noted, don't use Prolog's cut (!) operator!**
8. You should make your predicates as simple and elegant as possible. **You will have points taken off if you use abominations such as the ones below.**
   (a) It's bad Prolog style to use excessive equals (`=`)-operators:

   ```
   % BAD!                  % GOOD!
   foo(A,B) :- A = B.      foo(A,A).
   ```

   (b) Don't use semi-colons for **or**, instead use multiple predicates:

   ```
   % BAD!                      % GOOD!
   foo(A) :- bar(A); zap(A).   foo(A) :- bar(A).
                               foo(A) :- zap(A).
   ```

   (c) Don't use the `is`-operator except when you want to evaluate an arithmetic expression:

   ```
   % BAD!                  % GOOD!
   foo(A,B) :- A is B.     foo(A,B) :- A is B+1.
   ```

2 Useful Predicates

You may find the following built-in Prolog predicates useful.

The predicate `name(Atom,List)` splits up an atom into a string (list of ASCII characters), and converts a string back into an atom:

```
| ?- name(hello,L).
L = [104,101,108,108,111]
| ?- name(L, [104,101,108,108,111]).
L = hello
```

Prolog's built-in operator `\+` ("not") succeeds when its argument fails:

```
| ?- \+ member(a,[]).
yes
| ?- \+ member(a,[a]).
no
```

The *cut* operator `!` acts like a *barrier*: once execution has passed the `!` operator, it will never backtrack past it again.

```
fruit(apple).
fruit(orange).
fruit(pear).

| ?- fruit(L).
L = apple ? ;
L = orange ? ;
L = pear
yes

| ?- fruit(L),!.
L = apple
yes
```

The cut-operator should be avoided at all cost in Prolog programming. There are a few cases when its use is unavoidable, for example to improve performance.

The `fail` predicate will always fail. It is often used to generate all possible solutions to a query:

```
tastiness(apple,2).
tastiness(pear, 6).
tastiness(orange,10).

yummy :-
   fruit(L),
   tastiness(L, T),
```

```
   T > 5,
   write(L), nl,
   fail.
yummy.

| ?- yummy.
orange
pear
```

The `yummy` predicate will "iterate" (using backtracking) back and forth between `fruit(L)` and `fail` until no more tasty fruits can be found. The second `yummy` predicate makes sure that `yummy` will always succeed (return `yes`).

3 l33t Translator [30 points]

```
pr010g \/\/45 ph1r57 |>351g/\/3|> 70 |>0 <0/\/\pu74710/\/41 11/\/gu1571<5,
ph0r 3x4/\/\p13 70 7r4/\/51473 b37\/\/33/\/ 3/\/g115h 4/\/|> phr3/\/<h. 7h15
\/\/0u1|> 54v3 b07h <u17ur35 phr0/\/\ 4<7u411y h4v1/\/g 70 134r/\/ ph0r31g/\/
14/\/gu4g35. 45 4 <0/\/53qu3/\/<3, pr010g h45 /\/\4/\/y pr0p3r7135 7h47 /\/\4|35
17 u53phu1 70 v4r10u5 7r4/\/514710/\/ 745|5.
```

Or, in plain English. . .

Prolog was first designed to do computational linguistics, for example to translate between English and French. This would save both cultures from actually having to learn foreign languages. As a consequence, Prolog has many properties that makes it useful for various translation tasks.

`l33t` (pronounced *elite*) is a "language" used by `k00l d00d5` (cool dudes) in online conversations, such as IM, chat rooms, EQ, etc. There are many dialects, depending on how much of a `ph@ l33+ h4x0r` (fat elite hacker) you are.

You can read more here: `http://en.wikipedia.org/wiki/Leet`.

Let's write an English to l33t translator! You should use this translation table (stored in the file `r00l5.pl`) from English letters to `l33t`:

```
translate("a", "4").
translate("b", "b").
translate("c", "<").
translate("c", "k").
translate("d", "|>").
translate("e", "3").
translate("f", "ph").
translate("g", "g").
translate("g", "9").
translate("h", "h").
translate("i", "1").
translate("j", "j").
translate("k", "|").
```

```
translate("l", "1").
translate("m", "/\\/\\").
translate("n", "/\\/").
translate("o", "0").
translate("p", "p").
translate("q", "q").
translate("r", "r").
translate("s", "5").
translate("t", "7").
translate("t", "+").
translate("u", "u").
translate("v", "v").
translate("w", "\\/\\/").
translate("x", "x").
translate("y", "y").
translate("z", "z").
translate(" ", " ").
translate(".", ".").
translate("!", "!").
translate("?", "?").
translate("-", "-").
translate(";", ";").
translate(":", ":").
translate(",", ",").
```

To simplify the translation we only translate one character at a time, ignoring such transformations as `at`$\mapsto$`@`.

1. `l33t` only deals with lower-case letters since h4x0r5 are way too busy to use the shift key. Therefore write a predicate `tolower` that converts all the upper-case letters in a string to lower case, and leaves the other characters untouched:

   ```
   | ?- tolower("HELLO",L), name(N,L).
   L = [104,101,108,108,111]
   N = hello ?

   | ?- tolower("HeLLo D00d",L), name(N,L).
   L = [104,101,108,108,111,32,100,48,48,100]
   N = 'hello d00d' ?
   ```

2. Write a translator that takes an English string as input and produces one or more `l33t` translations:

   ```
   | ?- [r00l5,l33t]. % load the translation table and your program
   | ?- english2l33t("cat",E).
   E = [60,52,55] ? ;
   E = [60,52,43] ? ;
   E = [107,52,55] ? ;
   E = [107,52,43] ? ;
   ```

3. Since Prolog represents strings as lists of ASCII character numbers the output gets a bit hard to read. Write the top-level `l33t` predicate that uses the predicates above to a) convert the input to lower case, convert the string to l33t, and c) clean up the output by converting the string to an atom:

```
l33t(E,M) :- ...

| ?- l33t("CaT",E).
E = '<47' ? ;
E = '<4+' ? ;
E = k47 ? ;
E = 'k4+' ? ;
```

My solution is 25 lines long.

4 A Facebook Database [30 points]

Assume that we have a database of who is friends with whom, such as might be used by `facebook.com`:

```
friend(christian,margaret).
friend(christian,jas).
friend(christian,todd).
friend(christian,ji).
friend(christian,geener).

friend(todd,christian).
friend(todd,susan).

friend(susan,todd).

friend(jas,christian).
friend(jas,geener).
friend(jas,clark).

friend(geener,christian).
friend(geener,jas).
friend(geener,ji).

friend(clark,pat).

friend(pat,mike).
friend(pat,clark).

friend(margaret,christian).

friend(ji,christian).
friend(ji,geener).
```

This database is stored in the file `friends.pl`.

We would like to be able to answer the question: "who is X friends with, through at most two intermediate people, and what is the friendship path?" For example:

```
| ?- fb(christian,clark,L).
```

```
L = [christian,jas,clark] ? ;
L = [christian,geener,jas,clark] ? ;
(1 ms) no

| ?- fb(christian,mike,L).
(1 ms) no

| ?- fb(christian,mark,L).
(1 ms) no

| ?- fb(christian,christian,L).
no

| ?- fb(christian,geener,L).
L = [christian,geener] ? ;
L = [christian,jas,geener] ? ;
L = [christian,ji,geener] ? ;
(1 ms) no
```

Note that the path from `christian` to `mike` is too long ([christian,jas,clark,pat,mike]) for them to be friends, and that `christian` is friends with `geener` through three different paths.

My solution is 12 lines long.

5 Battleships [40 points]

In the game of battleships, the two players each maintain two boards, one with their own ships (the *ships board*) and one with the ships of the opponent and the shots taken against them (the *shots board*). The goal is to discover the location of the opponent's ships by shooting rounds into their grid. See Figure 1 for an example.

In this part of the assignment you will write a very simple implementation of the battleship game. Each player is initally represented by a fact like this:

```
player(
   'Bob1',
   reader,
    boards(
       ["........",
       ".******.",
       "........",
       ".*...*..",
       ".....*..",
       ".....*..",
       ".....*..",
       "........"]
,
      ["........",
       "........",
```

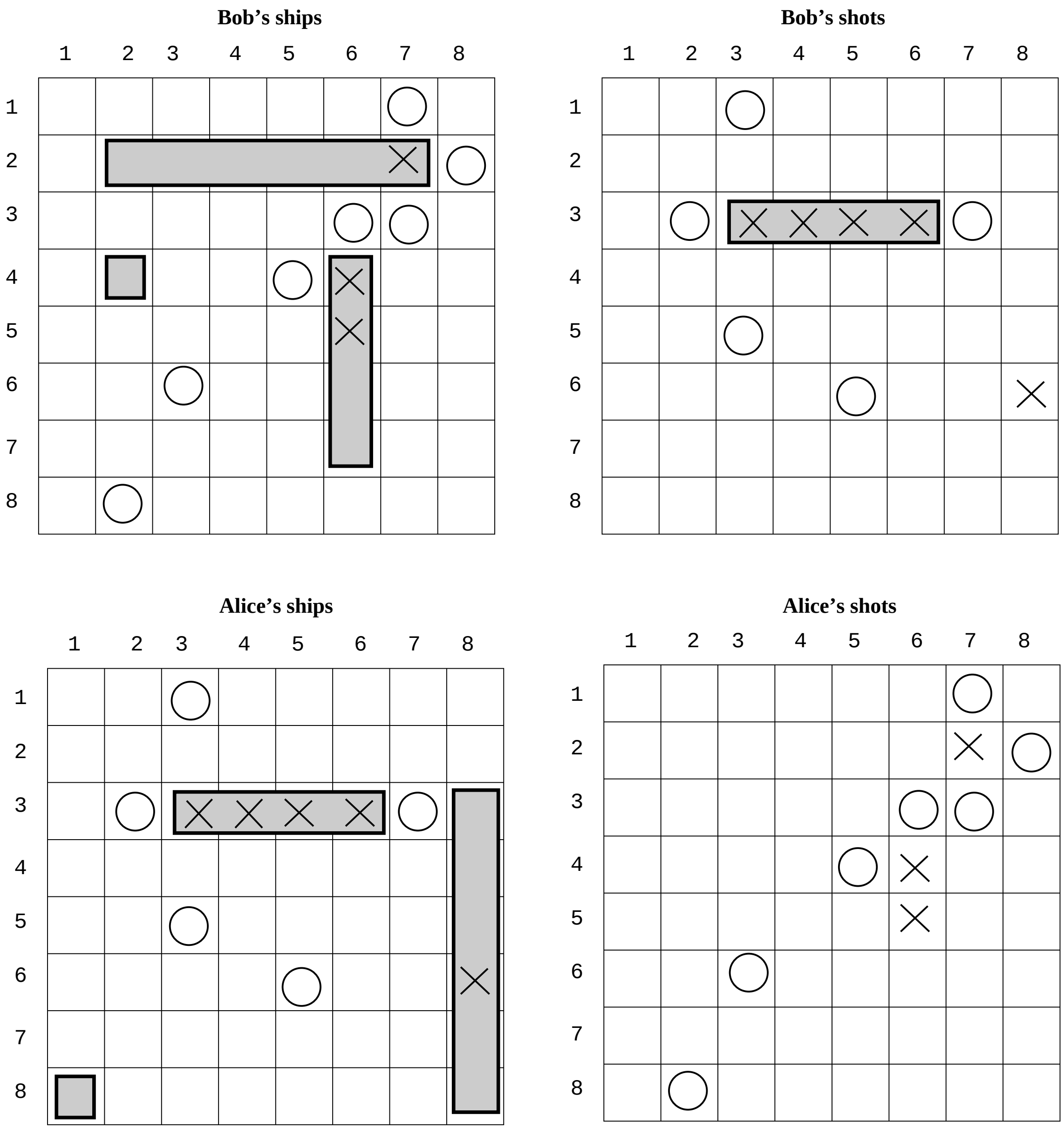

Figure 1: Alice and Bob playing battleships. In this example Bob has been able to discover and sink one of Alice’s ships, in locations $< 3, 3 > - < 3, 6 >$.

```
        "........",
        "........",
        "........",
        "........",
        "........",
        "........"]
    )
).
```

The second argument is the "strategy" the player uses. In our case, there are only two, `reader` which means a human player enters the moves on the terminal, and `random` which means that the machine chooses moves randomly. The program is written such that it is easy to add new strategies. The third argument to the `player` fact above is a `boards(`*`ships board`*`, `*`shots board`*`)`.

1. Various predicates for printing out a board have been defined for you:

```
printBoards(PlayerA,AsShips,AsShots) :-
   write(PlayerA), write(' Ships Board:'), nl, printBoard(AsShips),
   write(PlayerA), write(' Shots Board:'), nl, printBoard(AsShots).

printBoards(PlayerA,PlayerB,AsShips,AsShots,BsShips,BsShots) :-
   printBoards(PlayerA,AsShips,AsShots),
   printBoards(PlayerB,BsShips,BsShots).

| ?- player('Bob1',_,boards(Ships,Shots)),printBoards('Bob',Ships,Shots).
Bob Ships Board:
........
.******.
........
.*...*..
.....*..
.....*..
.....*..
........
Bob Shots Board:
........
........
........
........
........
........
........
........
```

2. To update the boards after each round, we need two predicates `get`, and `set`. `get` takes row and column numbers (indexing starts at 1) and a board as input and returns the entry in that cell as output. Dot (.) represents an empty entry, O (the letter "Oh") a miss, X a hit, and * the location of a ship. `put` similarly takes row and column numbers, a board, and a new entry as input, and returns a new board:

```
get(Row,Col,Bin,TheEntry) :- ...

set(Row,Col,Bin,TheEntry,Bout) :- ...

| ?- player('Bob1',_,boards(Ships,_)), get(2,3,Ships,Entry).
Entry = 42

| ?- player('Bob1',_,boards(ShipsIn,_)),
     "X"=[X],
     set(2,3,ShipsIn,X,ShipsOut),
```

```
     printBoard(ShipsOut).
........
.*X****.
........
.*...*..
.....*..
.....*..
.....*..
........
```

NOTE: `get` and `set` shouldn't be recursive predicates! They should solve the problem using backtracking only! You may use the `append` and `length` predicates! You can also use pattern matching and `is`, of course.

3. Using `set` and `get` above, implement the `shoot` predicate,

   `shoot(Row,Col,AsShots,AsShotsOut,BsShips,BsShipsOut).`

   `Row` and `Col` are the coordinates of the shot. `AsShots` is the shots board of the player doing the shooting before the shot, and `AsShotsOut` is the updated board after the shot. `BsShips` is the ships board of the person being shot at, and `BsShipsOut` is the updated board after the shot.

   After a cell has been shot at, it's updated like this:

`"*"`	$\Rightarrow$	`"X"`
`"."`	$\Rightarrow$	`"O"`
`"O"`	$\Rightarrow$	`"O"`
`"X"`	$\Rightarrow$	`"X"`

   I.e., if the cell being shot in is a `*`, change it to a `X`. If it's a dot, change it to an `O`, etc. Here's an example:

```
shoot(Row,Col,AsShots,AsShotsOut,BsShips,BsShipsOut) :- ...

oneshot("*","X").
oneshot(".","O").
oneshot("O","O").
oneshot("X","X").

| ?- player('Bob1',_,boards(_,BobShots)),
     player('Alice1',_,boards(AliceShips,_)),
     shoot(2,6,BobShots,BobShotsOut,AliceShips,AliceShipsOut),
     printBoard(BobShotsOut),nl,printBoard(AliceShipsOut).
........
.....X..
........
........
........
........
........
........
```

```
........
.....X..
.....*..
.....*..
.....*..
..*.....
........
.******.
```

The top board is what Bob sees after shooting one shot, and the bottom board is what Alice sees after the same shot. She can see that one of her length-4 ships has been hit in its "northernmost" cell, whereas Bob, of course, can only tell that he's hit *something*.

4. We've implemented two trivial player strategies for you. The first selects a random cell to shoot at, the second asks the user to input cell coordinates:

```
move(Player,random,PlayerShots,Row,Col) :-
   length(PlayerShots,L),
   random(1,L,Row),
   random(1,L,Col),
   write(Player), write(' plays '), write(Row), write(','), write(Col), nl.

move(Player,reader,_,Row,Col) :-
   write('Enter row: '), read(Row),
   write('Enter col: '), read(Col),
   write(Player), write(' plays '), write(Row), write(','), write(Col), nl.
```

5. The final thing to implement is a recursive routine to implement each round of play:

```
done(Ships) :- ...

% Add various base cases here to determine when A wins, B wins, and
% when it's a draw.
round(PlayerA,AsStrategy,AsShips,AsShots,PlayerB,BsStrategy,BsShips,BsShots) :-
   write('--------------------'), write(PlayerA), write('--------------------'), nl,
   ...
   printBoards(PlayerA,PlayerB,AsShips,AsShots1,BsShips1,BsShots),
   write('--------------------'), write(PlayerB), write('--------------------'), nl,
   ...
   printBoards(PlayerA,PlayerB,AsShips1,AsShots1,BsShips1,BsShots1),!,
   round(PlayerA,AsStrategy,AsShips1,AsShots1,PlayerB,BsStrategy,BsShips1,BsShots1).
```

`PlayerA` and `PlayerB` are the names ('Bob' or 'Alice') of the players. `AsStrategy` and `BsStrategy` are one of the atoms `random` or `reader`. `AsShips`/`AsShots` and `BsShips`/`BsShots` are the ships and shots boards of the two players at the beginning of the round, respectively.

The `done(Ships)` predicate is used to determine if the game is over, and if so, who has won. It takes a player's ships board as input and succeeds if it contains no *-characters. I.e., if Alice has successfully replaced all *-characters by X-characters in Bob's ships board, then she has won! If both have been able to shoot each other's ships, it's a draw.

NOTE: The `round` predicate should only succeed once, so you may have to add various *cuts* (!) to stop it from backtracking.

6. Finally now, we can put everything together:

```
battleships(Alice,Bob) :-
   player(Alice,AlicesStrategy,boards(AlicesShips,AlicesShots)),
   player(Bob,BobsStrategy,boards(BobsShips,BobsShots)),
   round('Alice',AlicesStrategy,AlicesShips,AlicesShots,
         'Bob',BobsStrategy,BobsShips,BobsShots).

ships :-
   battleships('Alice1','Bob1').

% Bob enters 7,6, as his first (and only) move, and wins.
bobWins :-
   battleships('Alice2','Bob2').

% Alice and Bob both enter 7,6, resulting in a draw.
aDraw :-
   battleships('Alice3','Bob3').
```

Here's a final example:

```
| ?- bobWins.
--------------------Alice--------------------
Alice plays 6,4
Alice Ships Board:
........
........
........
........
........
........
.....*..
........
Alice Shots Board:
........
........
........
........
........
...O....
........
........
Bob Ships Board:
........
.******.
........
.*...*..
.....*..
...O.*..
.....*..
........
Bob Shots Board:
```

```
........
........
........
........
........
........
........
........
-------------------Bob-------------------
Enter row: 7.
Enter col: 6.
Bob plays 7,6
Alice Ships Board:
........
........
........
........
........
........
.....X..
........
Alice Shots Board:
........
........
........
........
........
...O....
........
........
Bob Ships Board:
........
.******.
........
.*...*..
.....*..
...O.*..
.....*..
........
Bob Shots Board:
........
........
........
........
........
........
.....X..
........
Bob wins!
```

If no one wins the program should print “it is a draw!”.

6 Submission and Assessment

The deadline for this assignment is noon, Thu Oct 27. It is worth 5% of your final grade.

You should submit the assignment electronically using `d2l.cs.arizona.edu`

Don't show your code to anyone, don't read anyone else's code, don't discuss the details of your code with anyone. If you need help with the assignment see the instructor or the TA.