CSC173 Module 1
Success Facilitation Survey
with Answers and FFQ feature.

Chris Brown

September 30, 2007

Please write your name on the bluebook. This is a closed-book SFS. There are 80 possible points (one per minute). Stay cool and write neatly. Answers are on the Web site.

1 Propositional Calculus, Karnaugh Maps, Circuits (20 min)

A. (10 min) Make a truth table for this PC formula:
$(z \Rightarrow (\neg x \wedge y)) \vee (z \Rightarrow (\neg x \wedge \neg y))$.

(Be very careful here ... this has to be right or nothing below works!)

B. (6 min) Make a Karnaugh map for this "function" (that is, for the column under that middle connecting $\vee$.)

C. (4 min) Convert the Karnaugh map into a simple circuit (hint: two gates).

Answer:

A. For that middle $\vee$ column I get

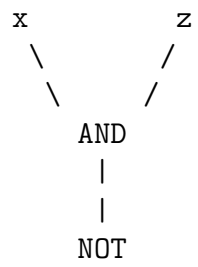
```
x y z   ...  V  ...
--------------------
1 1 1        0
1 1 0        1
1 0 1        0
1 0 0        1
0 1 1        1
0 1 0        1
0 0 1        1
0 0 0        1
```

B. One version of the Karnaugh Map (up to row and column rotation and reversal): The prime implicants are the top row ("not z"), and the left half ("not x").

```
       x y     x y    x y   x y
       0 0     0 1    1 1   1 0
z  0   1       1      1     1
   1   1       1      0     0
```

C. Circuit

From the Karnaugh map we get $\neg z \vee \neg x$, which by deMorgan is $\neg(z \wedge x)$:



2 PC Inference (15 min)

Given the following assertions in the database:

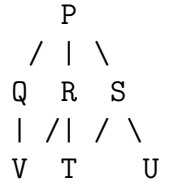
$\neg(Q \wedge R \wedge S) \vee P$
$(T \wedge V) \Rightarrow R$
$\neg(T \wedge U) \vee S$
$V \Rightarrow Q$
V
T
U

Prove P is true. You can do this either by: converting to Horn clauses and doing a forward-chaining or backward-chaining proof (say which you're doing) using (generalized) *modus ponens*, or by converting to Conjunctive Normal Form and doing a resolution proof. In the latter case, just to be traditional, assert also the negative of what you're trying to prove and derive the null clause.

Answer (sample): For Horn clauses, we get (, is "and"):

$Q, R, S \Rightarrow P$
$T, V \Rightarrow R$
$T, U \Rightarrow S$
$V \Rightarrow Q$
V, T, U

And your proof graph (either fwd or bkwd chaining), where arcs are inferences, looks like:



For resolution, you have to convert to CNF, for clauses of the form:
$(\neg Q \vee \neg R \vee \neg S \vee P)$
...
$(\neg V \vee Q)$
V, T, U
$\neg P$ (negate what you want to prove)

And you cancel the heck out of everything using resolution...

3 English to FOPC (15 min)

In your answer, use $F(x, y)$ for "x is the father of y", $D(x, y)$ for "x disapproves of y", $H(x, y), L(x, y)$ for "x Hates y" and "x Likes y", M (a constant) for "me", T (a constant) for "Tom", $A(x, y)$ for "x asks y to the prom", $r(x)$ (a function) for "the ring of x", and $R(x, y, z)$ for "x returns y to z".

Translate these English sentences into FOPC using the definitions above.

A. (5 min) Some fathers disapprove of their children.

B. (5 min) Some people hate people that some people like.

C. (5 min) Unless Tom asks me to the prom I'll return his ring.

Answer:

A. (NOT): $\exists x \forall y (F(x, y) \wedge D(x, y))$.
I thought I had this right but no. Max noticed that this asserts that x is everyone's father, among other things. I like his answer better:
$\exists x [\exists y (F(x, y)) \wedge \forall z (F(x, z) \Rightarrow D(x, z))]$.

B. $\exists x \exists y \exists z (H(x, y) \wedge L(z, y))$.

C. $\neg A(T, M) \Rightarrow R(M, r(T), T)$.

4 Prolog (30 min)

A (15 min) **Nondeterministic Programming**

A.1. (12 min) This prolog program computes all Pythagorian integer triples (X, Y, Z such that $X^2 + Y^2 = Z^2$) —Partial execution shown below. Explain how each predicate, thus the whole program, works.

A.2. (3 min) Notice those trivial answers with 0-length sides?? How can you get rid of these boring solutions and only get positive X, Y, Z?

```
pythag(X,Y,Z) :-
  inttriple(X,Y,Z),
  Sumsq is X*X + Y*Y,
  Sumsq is Z*Z.

inttriple(X,Y,Z) :-
  is_integer(Sum),
  minus(Sum,X,Sum1),
  minus(Sum1,Y,Z).

minus(Sum, Sum, 0).
minus(Sum, D1, D2) :-
  Sum > 0,
  Sum1 is Sum - 1,
  minus(Sum1, D1, D3),
  D2 is D3 + 1.

is_integer(0).
is_integer(N) :-
  is_integer(N1),
  N is N1 + 1.
------execution-----
?- pythag(X,Y,Z).

X = 0
Y = 0
Z = 0 ;

X = 1
Y = 0
Z = 1 ;
....
X = 6
Y = 0
Z = 6 ;

X = 4
Y = 3
Z = 5 ;
...
```

B. (15 min) **Backtracking and the cut (!)**

Given these clauses in the database (you `consulted` them):

```
change(you,i).
change(are, [am, not]).
change(french, dutch).
change(do, no).
change(X,X).

alter([],[]).
alter([H|T], [X|Y]) :- change(H,X), alter(T,Y).
```

Now you ask:

```
?- alter([you,are,a,computer],X).
```

B.1. (5 min) What do you see? (hint: it starts `X =`).

B.2. (5 min) Fine. But now you hit ";" several times, and you see three other 'answers', two partially right and one the same as the original first argument of `alter`. What causes this behaviour?

B.3. (5 min) How can you change your database so as not to generate these bogus answers?

Answer:

A.1. `pythag` uses the predicate `inttriple` to generate possible triples of integers X, Y, Z, then checks to see they meet the Pythagorean criterion. `inttriple` guarantees that *all* triples of integers are eventually generated. For that it calls `is_integer`, which generates the integers 0, 1, 2, ... in order starting with its base case and then creating the next integer given the previous one using its rule. These integers define the sum of the sides of the set of triangles we need to consider before moving on to consider the family of one-bigger triangles. Then using `minus`, a non-deterministic subtraction predicate, twice, `inttriple` finds all sides X, Y, Z such that the integer $= X + Y + Z$. `minus` computes the difference of its first two arguments and returns it in the third argument. Backtracking into `minus` (twice) is what forces `minus` to split up the integer (the triangle's 'circumference') into all possible sets of three sides (note we send in variables that it instantiates by search). This is how `append(X,Y,list)` can find all ways to divide a list into two parts, only it's used twice to find three variables.

A.2. You might want to start `is_integer` at 3, not 0, to avoid super-trivial answers. For the rest, add the conditions `X>0, Y>0, B>0` into the RHS of the `pythag` rule. Or, with a little care, you can modify `inttriple` or even `minus` to check for unwanted zeros.

B.1. First you see

```
X = [i, [am, not], a, computer] ;
```

Which is what you want.

B.2. But then if you hit ; several times, you get:

```
X = [i, are, a, computer] ;

X = [you, [am, not], a, computer] ;

X = [you, are, a, computer] ;

No
```

We've seen this in class. The problem is

```
alter([H|T], [X|Y]) :- change(H,X), alter(T,Y).
```

interacting with (backtracking into)

```
change(X,X).
```

This "don't do a change" rule is getting backed up into, giving the answers you don't want. B.3. You need to stop this back-up! The natural thing to do is

```
alter([H|T], [X|Y]) :- change(H,X),!, alter(T,Y).
```

but this also works:

```
dchange(you,i) :- !.
dchange(are, [am, not]) :- !.
dchange(french, dutch) :- !.
dchange(do, no) :- !.
dchange(X,X).

dalter([],[]).
dalter([H|T], [X|Y]) :- dchange(H,X), dalter(T,Y).
```

Either way, you get the right thing, *e.g.*

```
?- dalter([you, are, a,computer],X).
X = [i, [am, not], a, computer] ;
No
```