

## COSC 2P93 Study Sheet

Note: The final exam is substantially longer than this, and includes additional types of tasks, such as DFG, mystery programs, and much larger programs.

The following is presented to be better prepared for variable unification and list manipulation.

Give the final results of the following attempts at variable unification (if it fails, say why):

a)  $H=3+9$ .

$H=3+9$  (Remember, since **is** was not used, it merely assigns the structure of  $3+9$  to  $H$ )

b) Tallgeese is mobilesuit(0).

Error. (Using **is** means it tries to evaluate **mobilesuit** as a mathematical function/operation)

c)  $[1,2,3]=[H,M,N|T]$ .

$H=1, M=2, N=3, T=[]$  (The **tail** can be an empty list; a fact that can also lead to recursion bugs)

d)  $[1,2,3]=[H,M,N,T]$ .

False. (A **tail** can be an empty list, but this  $T$  is an **element**, which can't be an empty list)

e)  $\text{assert}(\text{match}(\text{stick}), \text{retract}(\text{match}(X)))$ .

$X=\text{stick}$ .

f)  $\text{assert}(\text{match}(\text{stick}), \text{retract}(\text{match}(X)), \text{retract}(\text{match}(X)))$ .

False. (Since the final **retract** fails to find a match, the whole query fails)

### Programming Task:

Swap first and last elements in a list.

eg.

?-  $\text{transfer}([a,b,c,d,e,f], X)$ .

$X = [f,b,c,d,e,a]$ .

```
transfer([H|T], [BackH|BackT]) :-  
    scoot(T, H, BackH, BackT).
```

```
scoot([First, Second|Rest], H, BackH, [First|BackT]) :-  
    scoot([Second|Rest], H, BackH, BackT).
```

```
scoot([Last], H, Last, [H]).
```

### Programming Task:

Write a predicate that, when given a list of numbers, raises each successive element to a higher power.

eg.

?- multiplaction([1,2,3,4],X).

X = [1,4,9,256].

```
multiplaction([H|T],[H|Returned]):-  
    process(T,T,Returned).
```

```
process([H|T],[OriginalH|OriginalT],[PartialH|Returned]):-  
    multiplyLists([H|T],[OriginalH|OriginalT],[PartialH|PartialT]),  
    process(PartialT,OriginalT,Returned).  
process([],_,[]).
```

```
multiplyLists([H|T],[ByH|ByT],[Mult|Returned]):-  
    Mult is H*ByH,  
    multiplyLists(T,ByT,Returned).  
multiplyLists([],[],[]).
```

The idea here is to break it up in terms of functionality. `multiplyLists` receives two lists, and returns one. The first argument is the working list that needs to be multiplied. The second argument is a portion of a copy of the original list of numbers, also to be multiplied. Thus, `multiplyLists` handles multiplication of list segments.

`process` handles the behaviour of invoking multiplication on progressively smaller portions of the list, and aggregating the result back into a list to return.

### Constraint Logic Programming:

(Note:  $V$  is a unification of domains. Thus,  $1..3V5..7$  means "1 to 3, or 5 to 7")

List the possible values for X, Y, and Z for the following:

$[X,Y,Z]$  ins  $1..3V5V7..9$ ,

$Y\# = X+1$ ,

$Z\# = Y+2$ .

X: 2

Y: 3

Z: 5

$[X,Y,Z]$  ins  $1..3V5V7..9$ ,

$Y\# = X+1$ ,

$Z\# > Y$ .

X:  $1..2V7$  (X can't be 5, because that would make Y 6)

Y:  $2..3V8$

Z:  $3V5V7..9$  (note: you could also list all the triples, but there are quite a few)