

Code: CSC 480

Name: Hithesh Shanmugam

Assignment 3

Part 1: Proportional logic and proofs

(1) Knaves and Knights:

If you assume A is telling the truth then B is a knave and is lying. B says "neither I nor A are knaves" but if he is a knave as A claims then this is a lie.

Alternatively, if you assume B is telling the truth when he says, "neither I nor A are knaves," then they must both be knights. But A said, "B is a knave," and if they are both knights then this is a lie, and knights, of course, never lie. So, B cannot be telling the truth.

Let's start as knight(B)

Says (B, "knight(A) /\ knight(B)") // neither of them is knave so we use both as knights.

By (2) and MP we conclude (knight(A) /\ knight(B))

By AE knight(A)

Says (A, "knave(B)")

By (2) and MP we conclude that knave(B)

Therefore, A is a knight and B is a knave.

(2) Knaves and Knights:

If we assume that A is telling the truth then both of them should be knights but in the other case if A is lying then both of them are not knaves.

If we consider what B is telling "We are not the same kinds" here there is a possibility that either one of them is knight or knave. If both are knights as A said then B should tell the same. So, A is lying and B is telling the truth.

Knight(A)

Says (A, knight(A) /\ knight(B))

By (2) and MP knight(A) /\ knight(B)

By AE knight(B)

Says (B, knave(A) /\ knight(B))

By (2) and MP knave(A) /\ knight(B)

By AE knave(A)

Therefore, A is a knave and B is a knight.

(3) Knaves and Knights:

Here we didn't hear what A said but the question is she a knight or knave? If A is a knight, he would say that he is a knight or if A is a knave, he still would have said that he is a knight.

If we take what B said i.e., "A said she was a knave" which is not true because in either cases A will be saying that she is knight so B is lying.

Now if we consider what C said i.e., "Don't believe B he is lying" we found that B is definitely lying by the above case so C must be saying the truth.

Therefore, B is a knave and C is a knight. As for A we know the answer he would have said and B said "A said she was a knave" so from that we can't say what is A. So, it might be a knave or knight.

(4) Knowledge base:

We have the KB as,

KB1: $(B \vee \sim D) \Rightarrow A$

KB2: $C \vee \sim D$

KB3: $C \Rightarrow B$

KB4: D

To prove: $A \wedge B$

Let's negate the proof value $\sim(A \wedge B)$

Applying DE Morgan's law $\sim A \vee \sim B$

Let us consider the KB1: $(B \vee \sim D) \Rightarrow A$

$\sim(B \vee \sim D) \vee A$ implementing the negation inside $(\sim B \vee D) \vee A$

Applying distributive law, $(A \vee \sim B) \wedge (A \vee D)$

We can write this as two KB $(A \vee \sim B), (A \vee D)$

$(A \vee D) (\sim A \vee \sim B)$ this gives us $D \vee \sim B$

Now if we consider KB3: $C \Rightarrow B$ this can be derived as $\sim C \vee B$

$(D \vee \sim B) (\sim C \vee B)$ this gives us $D \vee \sim C$

$(D \vee \sim C)$ KB2: $C \vee \sim D$ this cancels out and we can conclude that $A \wedge B$ has been proved.

Part 2: First order logic

Let $M(x)$ indicates that x is a member

Let $E(x,y)$ indicates that x is equal to y

Let $R(x,y)$ indicate that x and y are related.

(5) First order predicates:

“sally and ellen are members” this can be changed as

$$\forall x \quad E(x,sally) \vee E(x,ellen) \rightarrow M(x)$$

“ellen is related to bill” this can be changed as

$$\forall x,y \quad E(x,ellen) \vee E(y,bill) \rightarrow R(x,y)$$

“Anyone related to a member is also a member” this can be changed as

$$\forall x,y \quad R(x,y) \wedge M(x) \rightarrow M(y)$$

“Being related is symmetric” this can be changed as

$$\forall x,y \quad R(x,y) \rightarrow R(y,x)$$

“bob is not a member” this can be changed as

$$\forall x \quad E(x,bob) \rightarrow \neg M(x)$$

(6) CNF and Clause form:

$A \rightarrow B$ is equivalent to $\neg A \vee B$.

$A \wedge B$ is equivalent to $\neg A \vee \neg B$.

(a) $\forall x \quad E(x,sally) \vee E(x,ellen) \rightarrow M(x)$ from above equations we can write this equation can be written as $\forall x \quad \neg(E(x,sally) \vee E(x,ellen)) \vee M(x)$

(b) $\forall x,y \quad E(x,ellen) \vee E(y,bill) \rightarrow R(x,y)$ from above equations we can write this equation can be written as $\forall x,y \quad \neg(E(x,ellen) \vee E(y,bill)) \vee R(x,y)$

(c) $\forall x,y \quad R(x,y) \wedge M(x) \rightarrow M(y)$ from above equations we can write this equation can be written as $\forall x,y \quad \neg R(x,y) \vee \neg M(x) \vee M(y)$

(d) $\forall x,y \quad R(x,y) \rightarrow R(y,x)$ from above equations we can write this equation can be written as

$$\forall x,y \quad \neg R(x,y) \vee R(y,x)$$

(e) $\forall x \quad E(x,bob) \rightarrow \neg M(x)$ from above equations we can write this equation can be written as

$$\forall x \quad \neg E(x,bob) \vee \neg M(x)$$

(7) Resolution-refutation:

To Prove: Bill is a member of organisation

Now the combined statement will be,

$$\begin{aligned} & \forall xy \quad (\neg(E(x,sally) \vee E(x,ellen)) \vee M(x)) \wedge \\ & (\neg(E(x,ellen) \vee E(y,bill)) \vee R(x,y)) \wedge \\ & (\neg R(x,y) \vee \neg M(x) \vee M(y)) \wedge \\ & (\neg R(x,y) \vee R(y,x)) \wedge \\ & (\neg E(x,bob) \vee \neg M(x)) \end{aligned}$$

To prove our goal we can substitute $E(x,Bill)$ to be true and $M(x)$ to be false and at the same time will not lead to any satisfying assignment of the set of all clauses connected with \wedge .

Let's consider the clause $(\neg(E(x,ellen) \vee E(y,bill)) \vee R(x,y))$ this can be proved to be true iff Ellen and Bill are related.

The clause $(\neg R(x,y) \vee \neg M(x) \vee M(y))$ this will be true iff both the x and y are either a member of the organisation or not a member of organisation.

Now this clause $(\neg(E(x,sally) \vee E(x,ellen)) \vee M(x))$ for this clause to be true iff Sally and Ellen are member of the organisation.

We know Bill is related to Ellen and Ellen is a member of organisation, in a case if Bill is not a member of organisation, then these clauses won't be true simultaneously. So, Bill must be the member of organisation.

(8) Resolution-refutation:

To prove: Someone to whom sally is not related

Now the combined statement will be,

$$\begin{aligned} & \forall xy \quad (\neg(E(x,sally) \vee E(x,ellen)) \vee M(x)) \wedge \\ & (\neg(E(x,ellen) \vee E(y,bill)) \vee R(x,y)) \wedge \\ & (\neg R(x,y) \vee \neg M(x) \vee M(y)) \wedge \\ & (\neg R(x,y) \vee R(y,x)) \wedge \\ & (\neg E(x,bob) \vee \neg M(x)) \end{aligned}$$

To prove our goal we need to substitute $R(x,sally)$ to be true and we can check the $R(sally,x)$ to be true for every other relatives.

We know that bob is not a member ($\neg E(x, \text{bob}) \vee \neg M(x)$) and we have a relation as "Anyone related to a member is also a member" ($\neg R(x, y) \vee \neg M(x) \vee M(y)$) this is true iff anyone is related to a member.

We know that sally and ellen are members of the organisation. ($\neg(E(x, \text{sally}) \vee E(x, \text{ellen})) \vee M(x)$) this is going to be true iff both are members.

We also know that ellen and bill are related and they are members of the organisation from question 7.

So clearly bob is not related to sally from this knowledge base.

(9) Resolution refutation:

Now we have the change in the knowledge base as "There is someone who is not a member of organisation." Let's change this into CNF (Clause form)

$$\forall x, y \quad E(x, y) \rightarrow \neg M(x, y)$$

$$(\neg E(x, y) \vee \neg M(x, y))$$

Now we have that someone is not a member but that someone is definitely not ellen or bill because we know ($\neg(E(x, \text{sally}) \vee E(x, \text{ellen})) \vee M(x)$) this is going to be true iff both are members.

And from 7 ellen and bill are related and they are members of the organisation.

So, we cannot really say sally is related to someone or not.

(10) Generalized Modus Ponens:

From analysing the data given we can write as

$$(a) \text{ exchange}(x, \text{bob}) \leftarrow \text{in-circle}(x, \text{bob})$$

$\text{in-circle}(x, \text{bob}) \leftarrow \text{linked}(z, \text{bob}) \wedge \text{friend}(x, z)$ here z is linked to bob through either a friend x or a friend other than x

From the Horn Knowledgebase given to us we can say as follows,

Considering $\text{linked}(z, \text{bob})$ we have matching facts such that $\text{linked}(\text{kim}, \text{bob})$ and $\text{linked}(\text{don}, \text{bob})$ from this we could say that z could either be kim or don.

Now let us consider $\text{friend}(x, z)$ here there is only one that matches the fact such that $\text{friend}(\text{kim}, \text{don})$

From the fact $\text{friend}(x, z)$ it can mean x is a friend of z or it can also mean z is friend of x.

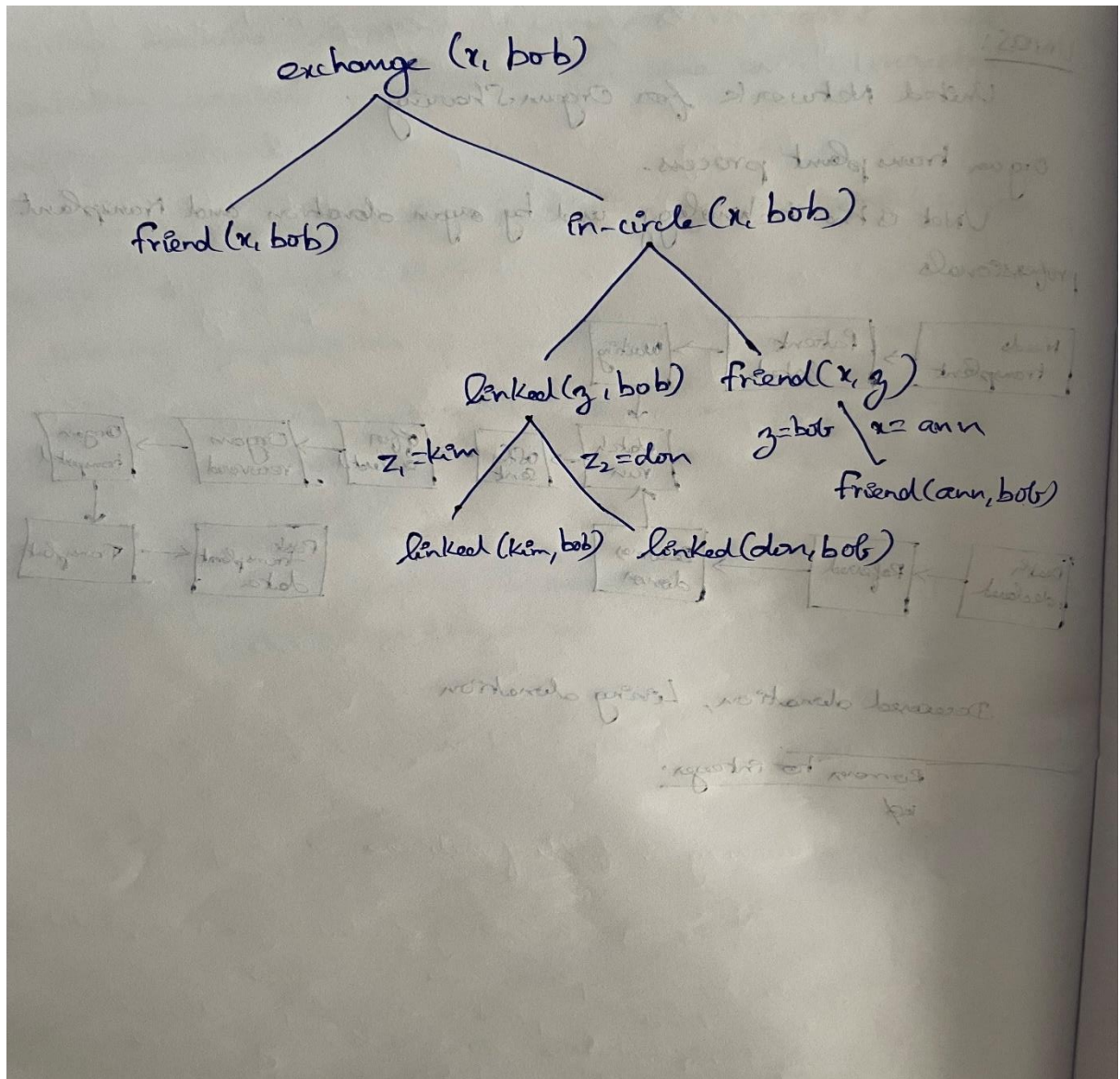
From the above facts we can come to that both kim and don can be values of x.

$$(b) \text{ exchange}(x, \text{bob}) \leftarrow \text{friend}(x, \text{bob})$$

From this and $\text{friend}(\text{ann}, \text{bob})$, x value is ann.

So, the values of **x** can be **kim, don and ann**.

Those are the values of **x** but in the $\text{exchange}(x, \text{bob})$ the only possible value would be kim.



Part 3: Exploring Prolog

(11) Prolog program:

We have the knowledge base as

1. $\text{friend}(\text{ann}, \text{bob})$
2. $\text{friend}(\text{kim}, \text{don})$
3. $\text{linked}(\text{kim}, \text{bob})$
4. $\text{linked}(\text{don}, \text{bob})$

5. `friend(x,y) => exchange(x,y)`

6. `in-circle(x,y) => exchange(x,y)`

7. `linked(z,y) /\ friend(x,z) => in-circle(x,y)`

In Prolog the lines 5 and 6 can be written as,

`exchange(X,Y) :- friend(X,Y).`

`exchange(X,Y) :- in-circle(X,Y).`

For line 7 we have to do CNF to find the prolog form

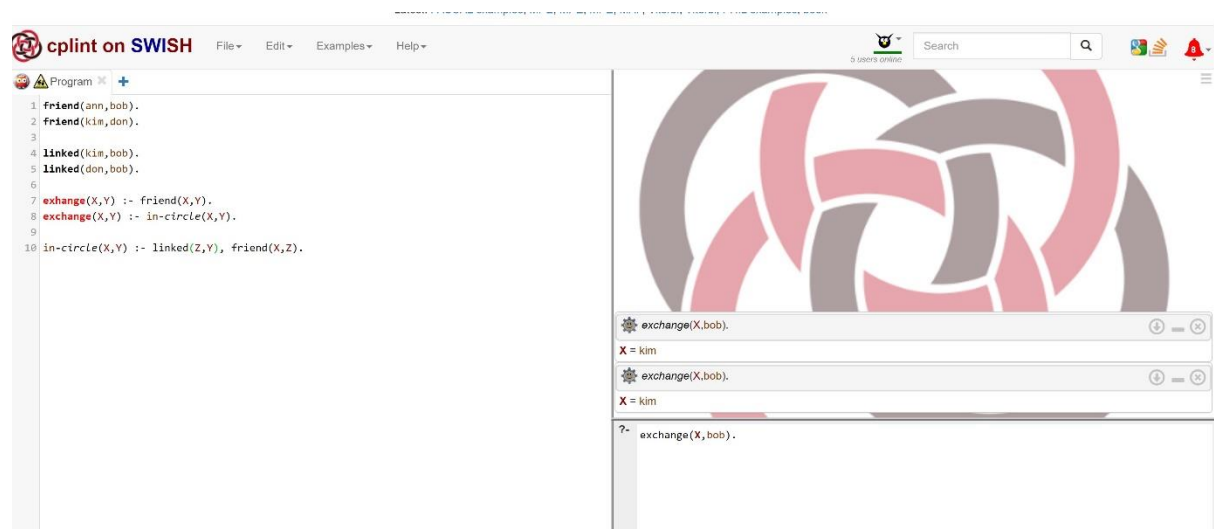
`linked(z,y)/\friend(x,z)->in-circle(x,y)`

$\sim((\text{linked}(z,y) \wedge \text{friend}(x,z)) \vee \text{in-circle}(x,y))$

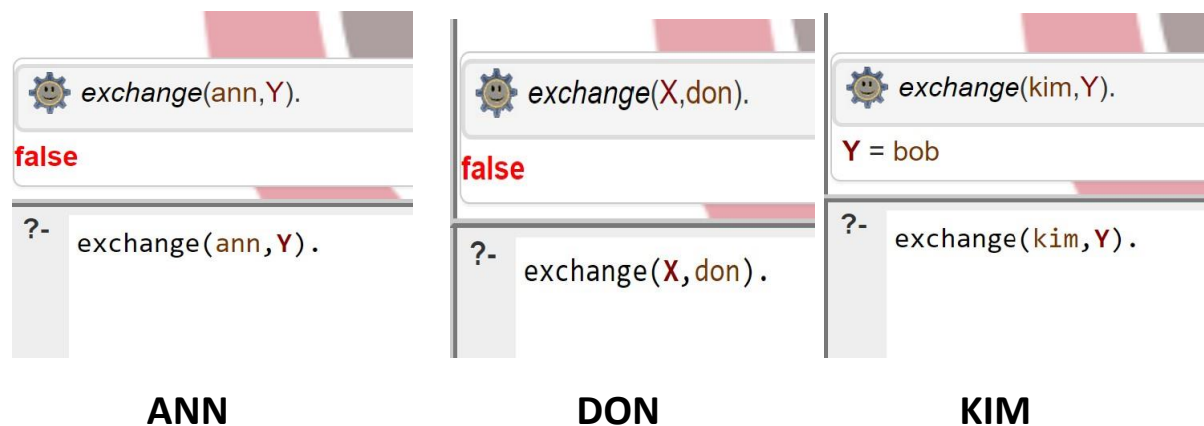
Applying DE Morgan's law $(\sim \text{linked}(z,y) \vee \sim \text{friend}(x,z)) \vee \text{in-circle}(x,y)$

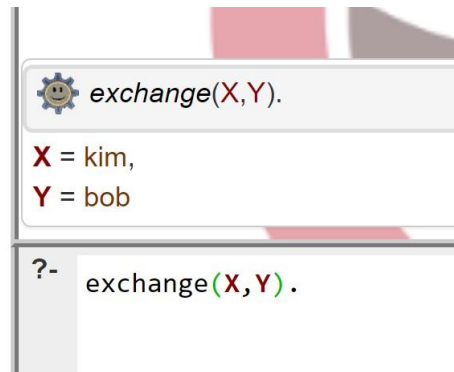
$\sim \text{linked}(z,y) \vee \sim \text{friend}(x,z) \vee \text{in-circle}(x,y)$ this can be written in prolog form as,

`in-circle(X,Y) :- linked(Z,Y), friend(X,Z).`



BOB





Exchange

(12) Prolog programming:

Code: (for both a and b)

`on(i,h).`

`on(f,e).`

`on(g,f).`

`on(b,a).`

`on(c,b).`

`on(d,c).`

`left(e,h).`

`left(g,i).`

`left(f,i).`

`left(a,e).`

`left(b,f).`

`left(d,g).`

`left(c,g).`

`blocks-on(X,Y) :- on(X,Y).`

`blocks-on(X,Y) :- on(X,Z), blocks-on(Z,Y).`

`stack-left(X,Y) :- left(X,Y).`

`stack-left(X,Y) :- left(X,Z), stack-left(Z,Y).`

`blocks(X,Y) :- on(X,Z), left(Z,Y).`

`above(X,Y) :- blocks-on(X,Z), blocks-on(Z,Y).`

(a) On and left: Queries:

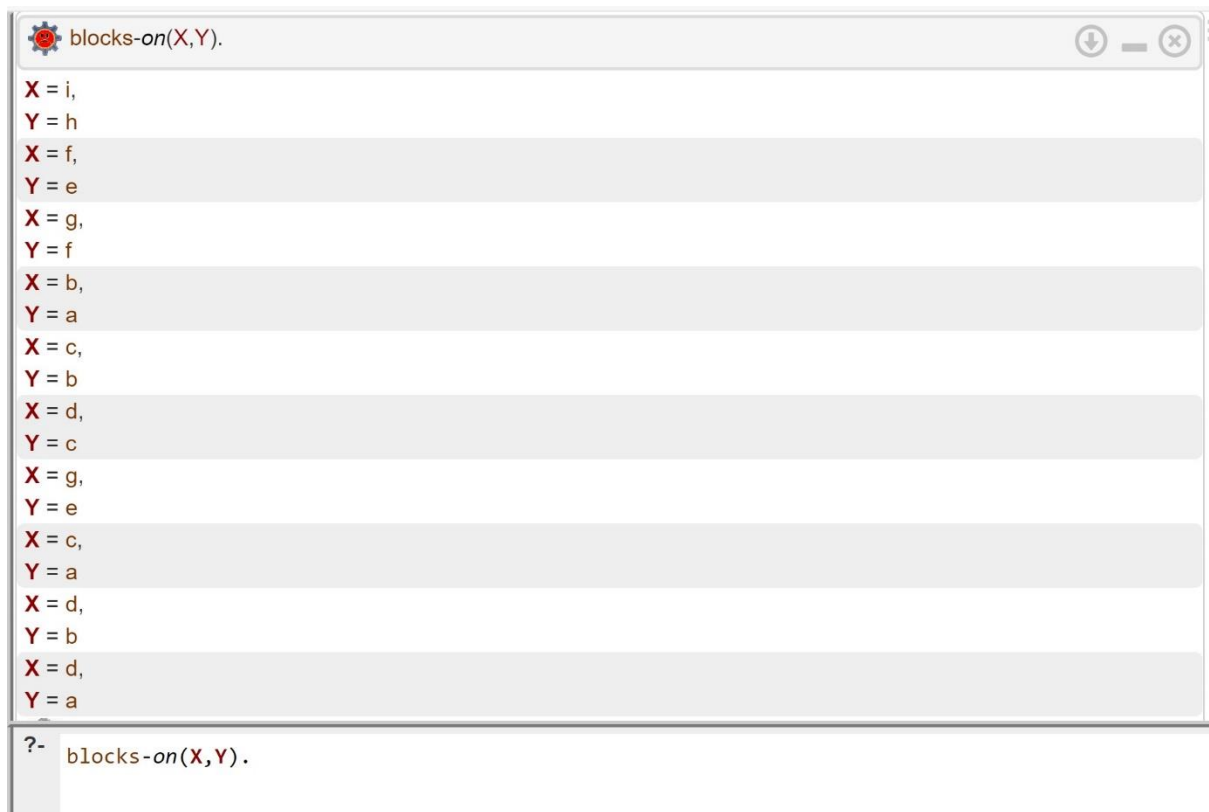
o what blocks are on block b? (`blocks-on(X,b).`)



o what blocks is block a on? (`blocks-on(a,Y).`)



o what blocks are on other blocks? (blocks-on(X,Y).)



```
blocks-on(X,Y).  
X = i,  
Y = h  
X = f,  
Y = e  
X = g,  
Y = f  
X = b,  
Y = a  
X = c,  
Y = b  
X = d,  
Y = c  
X = g,  
Y = e  
X = c,  
Y = a  
X = d,  
Y = b  
X = d,  
Y = a  
?- blocks-on(X,Y).
```

o what blocks are on blocks that are immediately to the left of other blocks? (blocks(X,Y).)



```
blocks(X,Y).  
X = f,  
Y = h  
X = g,  
Y = i  
X = b,  
Y = e  
X = c,  
Y = f  
X = d,  
Y = g  
?- blocks(X,Y).
```

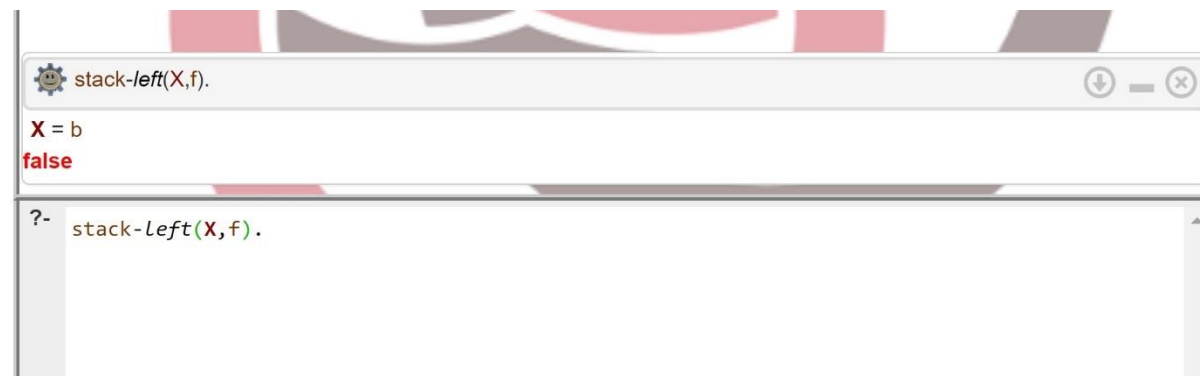
(b) above and stack-left:

o what blocks are above other blocks? (above(X,Y).)



```
above(X,Y).  
  
X = g,  
Y = e  
X = c,  
Y = a  
X = d,  
Y = b  
X = d,  
Y = a  
X = d,  
Y = a  
  
?- above(X,Y).
```

o what blocks are in a stack immediately to the left of stack for f? (stack-left(X,f).)



```
stack-left(X,f).  
  
X = b  
false  
  
?- stack-left(X,f).
```

(13) List in Prolog: Instead of two different add and delete I did it in one and the queries are listed as below,

Code:

L=[]

delete_at(X,L,K,NewL) :- X.

delete_at(X,[X|Xs],1,Xs).

delete_at(X,[Y|Xs],K,[Y|Ys]) :- K > 1, K1 is K - 1, delete_at(X,Xs,K1,Ys).

L=[]

add_at(X,L,K,NewL) :- X.

add_at(X,L,K,NewL) :- delete_at(X,NewL,K,L).

Queries:

delete_at(X,[a,b,c,d],1,NewL).

NewL = [b, c, d],

X = a

Next	10	100	1,000	Stop
------	----	-----	-------	------

?- delete_at(**X**, [a,b,c,d],1,**NewL**).

delete_at(X,[a,b,c,d],4,NewL).

NewL = [a, b, c],

X = d

Next	10	100	1,000	Stop
------	----	-----	-------	------

?- delete_at(**X**, [a,b,c,d],4,**NewL**).

add_at(xyz,[a,b,c,d],1,L).

L = [xyz, a, b, c, d]

Next	10	100	1,000	Stop
------	----	-----	-------	------

?- add_at(xyz, [a,b,c,d],1,**L**)

add_at(pqrs,[a,b,c,d],5,L).

L = [a, b, c, d, pqrs]

Next	10	100	1,000	Stop
------	----	-----	-------	------

?- add_at(pqrs, [a,b,c,d],5,**L**).

In order to know the size so that we could add or delete at the end (by finding the end we can add 1 to it and replace K in the query with that)we can use this code:

Code:

```
L=[]
```

```
length([], 0).
```

```
L=[]
```

```
length([H|T], N) :- length(T, N1), N is N1+1.
```

Query:

```
length([a,b,c,d],N).
```

```
N = 4
```

```
?- length([a,b,c,d],N).
```

This code can add or delete element at any position the list.