Intro to AI

Exam 2012-2013

1) a) DFS would have problems with such a graph because it would get stuck in an infinite loop trying to reach E. We can observe that in our graph we have a cycle : B-C-D. Since DFS does not stop until the goal node or a leaf is reached, this might mean that DFS could end up in an infinite cycle if say it would always choose B to be the child of D that it explores first.

Iterative Deepening would not have such a problem, because it deals with this by having depth levels at which it stops. Therefore, at level depth = 3, when we reach node D, we check B and we stop there because we have reached our maximum depth and backtrack to D. Then, we chose the next child of D, which is E and we find our solution.

Hence, Iterative Deepening rules out the possibility of infinite looping due to cycles, because it sets maximum levels at which it can explore. By combining the BFS and DFS strategies it manages to check all nodes.

b)
// *Not sure about this one*
We need to transform the HoldsAt into Happens

HoldsAt(AboveChasm, t) :- ∃ e, t' [ Happens(e, t') ^ Initiates(e, AboveChasm, t') ^ t'<t ^ not Clipped(t', AboveChasm, t).

Clipped(t', AboveChasm, t) : - ∃ e, t'' [ Happens(e,t") ^ t'<t"<t ^ Terminates(e,AboveChasm,t") ]

Initiates(Drop, Safe, t)) <- HoldsAt(GotRing, t) , HoldsAt(AboveChasm, t).
Initiates(GoChasm, AboveChasm, t').

Initially(GotRing, t0)
not Initially(AboveChasm, t0).
not Initially(Safe, t0).

GOAL : HoldsAt(Safe, tEnd).

Compute the CNF of the above. And then find all the Happens statements that are True after the satisfiability planning program runs. ( // Should I actually write the CNF also specify t0 < t' <tEnd ?)

c) S0 = initial situation

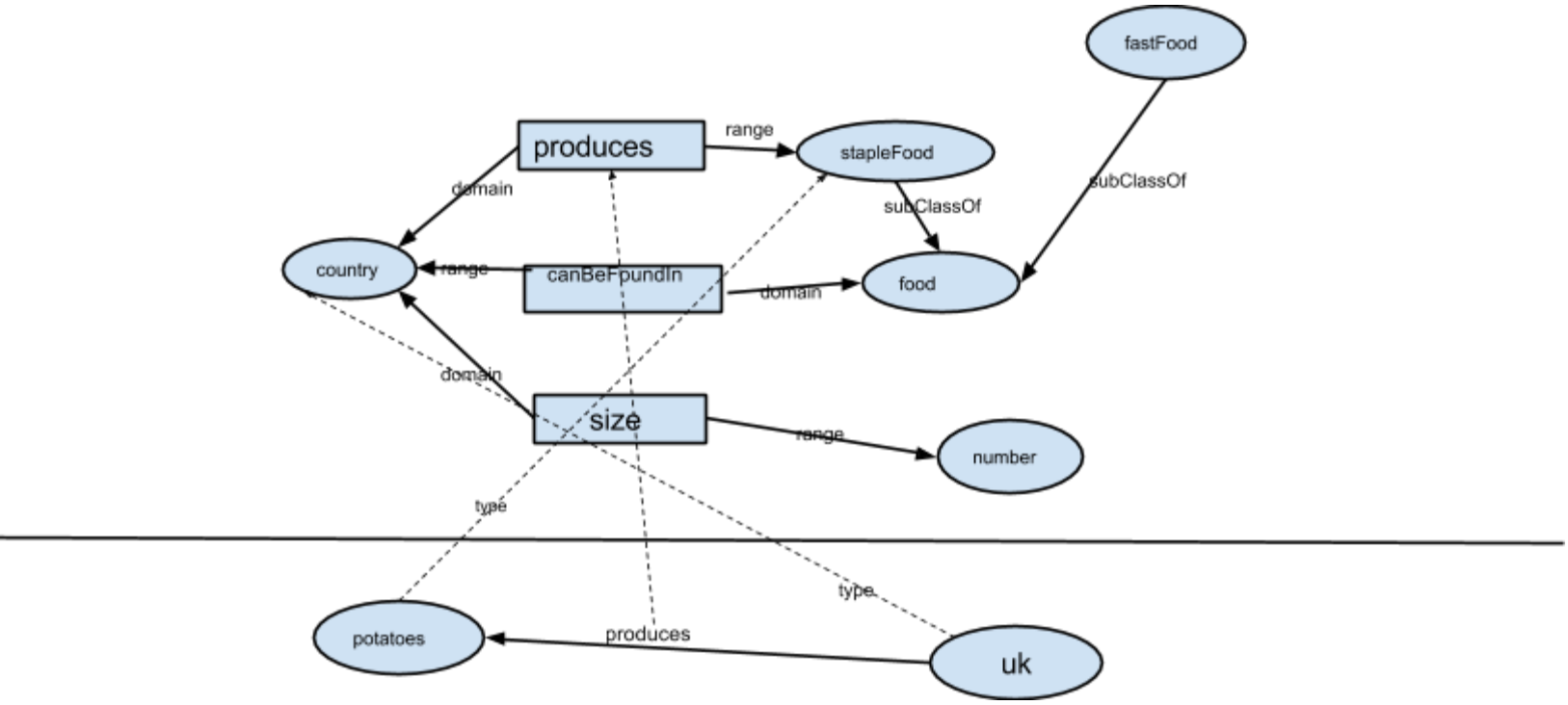
Holds(GotRing, S0)

not Holds(AboveChasm, S0).
not Holds(Safe, S0).

Holds(Safe, Result(Drop, S)) <- Holds(GotRing, S) , Holds(AboveChasm, S).
Holds(AboveChasm, Result(GoChasm,S)).

Goal : Holds(Safe, EndState)
The plan is action = GoChasm

2. a)




b) i) domain(produces, country), range(produces, stapleFood),
range(isProducedBy, country), domain(isProducedBy, stapleFood).

ii) sameClassAs(population, size).

iii) subClassOf(selfSufficientCountry, allValuesFrom(produces, stapleFood)).

[ C subClassOf allValuesFrom(P,D) === C(X),P(X,Y)->D(Y) === d(Y) :- c(X), p(X,Y).
P: property, D: class, allValuesFrom(P,D): all x s.t. y instance of D whenever P(x,y) ]

c) We start off with(orange):

stapleFood(potatoes) :
from food(X) <- stapleFood(X) : food(potatoes) {X\potatoes}

country(uk)

produces(uk, potatoes) :
from canBeFoundIn(X,Y) <- produces(X,Y) : canBeFoundIn(potatoes, uk) {X\uk, Y\potatoes}
from stapleFood(Y) <- produces(X,Y) : stapleFood(potatoes) { Y\potatoes}
from country(X) <- produces(X,Y) : country(uk) { X\uk}

canBeFoundIn(potatoes, uk) :
from food(X) <- canBeFoundIn(X,Y) : food(potatoes) { X\potatoes}

//Should I write all the additional information like country(uk) again from canBeFoundIn ??

OR

facts E = stapleFood(potatoes) ^ country(uk) ^ produces(uk,potatoes)
rules Pr = (¬stapleFood(X) or food(X) ) ^
( ¬fastFood(X) or food(X)) ^
( ¬produces(Y,X) or canBeFoundIn(X,Y)) ^
( ¬produces(X,Y) or country(X)) ^
( ¬produces(X,Y) or stapleFood(Y)) ^
(¬canBeFoundIn(X,Y) or country(Y)) ^
(¬canBeFoundIn(X,Y) or food(X)) ^
( ¬size(X,Y) or country(X)) ^
(¬size(X,Y0 or number(Y))

the same as above only in the form

stapleFood(potatoes) , food(X) <- stapleFood(X) ( mgu = {X\potatoes} }
food(potatoes) {X\[potatoes}
etc.

d) <-food(potatoes)
| from food(X) <- stapleFood(X) {X\potatoes}
stapleFood(potatoes)
|
BOX {X\potatoes}

<-food(potatoes)
| from food(X) <- stapleFood(X) {X\potatoes}
<- stapleFood(potatoes)
| from stapleFood(Y) <- produces(X',Y) {Y\potatoes, X\X'}
<-produces(X',potatoes)
| from produces(uk, potatoes) \{X'\uk}
BOX {X\potatoes, Y\potatoes, X'\uk}

<-food(potatoes)
| from food(X) <- canBeFoundIn(X,Y) {X\potatoes}
<-canBeFoundIn(potatoes, Y)
| from canBeFoundIn(X',Y') <- produces(Y',X') {X'\potatoes}
<- produces(Y,potatoes)
| from produces(uk,potatoes) {Y\uk}
BOX with {X\potatoes, X'\potatoes, Y\uk}

<-food(potatoes)
| from food(X) <- fastFood(X) {X\potatoes}
<-fastFood(potatoes)
|
FAIL

THE RESOLUTION VIEW : Use the CNF above and the negated goal : ¬food(potatoes).

e) The advantages of forward chaining is that you find all the information(all the goals in all the ways) in one go, compared to the backward chaining where you have to have a goal specifically in mind(you also have to backtrack if you want to find all the solutions to that goal). Moreover, backwards chaining can produce failures to the goal(extra computations). The disadvantage of forward chaining compared to the backwards one is that you might get superfluous information that you do not need if you are looking for something specific.