Project 1

Exercise 1 – Resolution

I have implemented to_clause predicate that gets a formula and apply the rules for Skolemization (rename the variables, creates new variables according to the existential quantifiers, pushes negation closer to the atom, takes into consideration and/or relationship, etc). Moreover there are some rules that are made to reduce the formula as much as possible, by removing redundant rules (and(a,a)=a, n(n(a))=a, etc).

In order to implement Skolemization, I implemented rename_vars predicate that rename the existential quantifiers. This take into consideration the relationship between forevery and exists as described here https://en.wikipedia.org/wiki/Skolem_normal_form. To unify these variables, I implemented unify_func predicate that tries to unify the variables from the function or fail. Also, substitute predicate was designed in order to apply the unifications.

In order to speed-up the backtracking process, I made use of the cut feature as much as possible.

The formats accepted are:

1) Formulas with variation of: n, and, or, forevery, exists, func, params, const, var.
2) CNF format (as described in the project requirements).

There are 2 predicates for resolution:
resolution_from_CNF(CNF).
resolution_from_KB(KB,Q).

In order to improve the time performance, we make a pre-cleaning of the CNF format such that it won't have duplicates inside clauses and are reduced as much as possible (n(n(a))->a, etc). During each step of the resolution, when the two chosen clauses are merged(after removing p and p negate), the possible duplicates are removed so that the next set of clauses don't contain any.

1.a

At a party for adults, the guests are asked to choose between three colors by following the rules
KB:

1. Every man or woman is an adult.
2. Every adult must choose between three colors: red, blue, black.
3. No man likes red color (an so he will never choose it).
4. Every married person will not choose black.
5. Sebi is married man.

Q: Sebi choose the blue color.

1.b

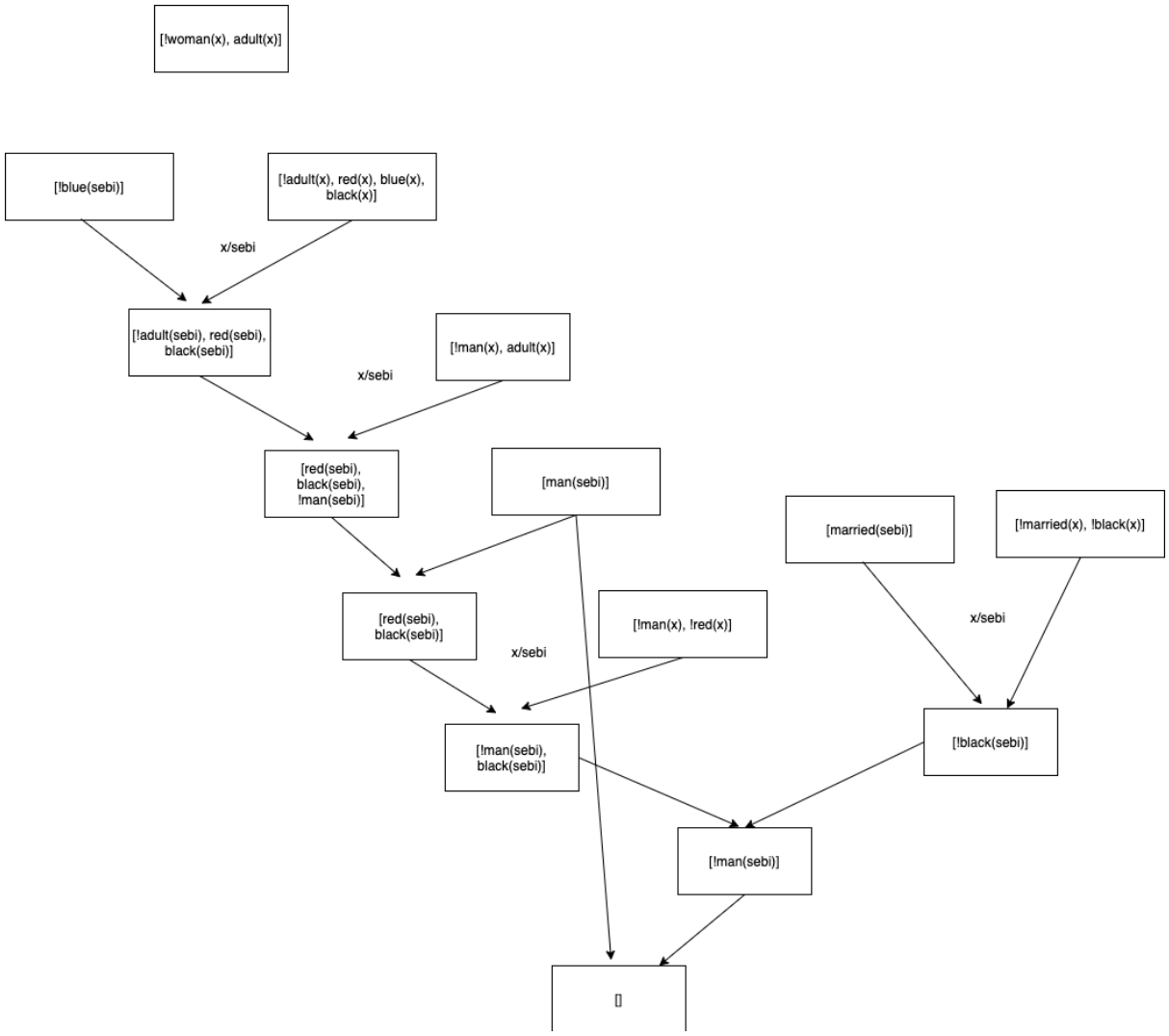
KB:
1.for every x: (man(x) or woman(x)) →adult(x)
2.for every x: adult(x) → (red(x) or blue(x) or black(x))
3.for every x: man(x) → !red(x)
4.for every x: married(x) → !black(x)
5.man(sebi).
6.married(sebi).

Q: blue(sebi)

Convert to CNF:
1 [!man(x), adult(x)]
2 [!woman(x), adult(x)]
3 [!adult(x), red(x), blue(x), black(x)]
4. [!man(x), !red(x)]
5.[!married(x), !black(x)]
6.[man(sebi)]
7.[married(sebi)]

Q: [blue(sebi)] => not(Q): [!blue(sebi)]

CNF = [[!man(x), adult(x)], [!woman(x), adult(x)], [!adult(x), red(x), blue(x), black(x)], [!man(x), !red(x)], .[!married(x), !black(x)], man(sebi)], married(sebi)], [!blue(sebi)]]



1.c

Conversion of the KB using the rules implemented in prolog.

KB
forevery(x, inclusion(or(func(man, x), func(woman, x)), func(adult, x)))
forevery(x, inclusion(func(adult, x), or(func(red, x), or(func(blue, x), func(black, x)))))
forevery(x, inclusion(func(man, x), func(n(red), x)))
forevery(x, inclusion(func(married, x), func(n(black), x)))
func(boy, sebi)
func(married, sebi)

Q func(blue, sebi).
!Q func(n(blue), sebi).

1.d)

i).resolution_from_CNF([[n(a),b],[c,d],[n(d),b],[n(c),b],[n(b)]]).
false

ii) resolution_from_CNF([[n(b),a],[n(a),b,e],[e],[a, n(e)],[n(a)]]).
false

iii) resolution_from_CNF([[n(a),b],[c,f],[n(f),b],[n(c),b],[n(c)]]).
true

iv) resolution_from_CNF([[a,b], [n(a),n(b)]]).
true

Observation: Since resolution was done on the negation of the Question, we must revert the result to establish if the Q is true of false.

Some simulation on the examples from the course.

$$KB \left\lceil \begin{array}{l} \forall x.GradStudent(x) \supset Student(x) \\ \forall x.Student(x) \supset HardWorker(x) \\ GradStudent(sue) \end{array} \right.$$

$$KB \models HardWorker(sue)$$

C3 Slide 28

KB Converted

1.forevery(x,inclusion(func(gradStudent,x), func(student,x)))
2.forevery(x,inclusion(func(student,x), func(hardWorker,x)))
3.func(gradStudent, sue)

Query (given as as List of Formulas + a Question): func(hardWorker, sue)

Simulation

resolution_from_KB([forevery(x,inclusion(func(gradStudent,x), func(student,x))), forevery(x,inclusion(func(student,x), func(hardWorker,x))), func(gradStudent, sue)], func(hardWorker, sue)). - **false.**

convert_from_KB_to_CNF([forevery(x,inclusion(func(gradStudent,x), func(student,x))), forevery(x,inclusion(func(student,x), func(hardWorker,x))), func(gradStudent, sue), n(func(hardWorker, sue))], R).
R = [[func(n(gradStudent), var(x_60)), func(student, var(x_60))], [func(n(student), var(x_109)), func(hardWorker, var(x_109))], [func(gradStudent, const(sue))], [func(n(hardWorker), const(sue))]].

!!Note that we applied Skolemization method to replace the variables.

To Do

KB

1. Anybody who passes some exam studies or is brilliant or is lucky.
2. Anybody who takes a 5 passes some exam.
3. No FMI student is lucky.
4. Anyone who drinks beer does not study.

5. (Question) Every FMI student who takes a 5 and drinks beer is brilliant.

C3 Slide 34

KB

passExam => brilliant or lucky : forevery(x, inclusion(func(passExam, x), or(func(brilliant, x), or(func(study, x), func(lucky, x)))))
brilliant => n(lucky) : forevery(x, inclusion(func(brilliant, x), func(n(lucky), x)))
lucky => n(brilliant): forevery(x, inclusion(func(lucky, x), func(n(brilliant), x)))
take5 => passExam : forevery(x, inclusion(func(take5, x), func(passExam, x)))
fmiStud => n(lucky) : forevery(x, inclusion(func(fmiStud, x), func(n(lucky), x)))
drinkBeer => n(study) : forevery(x, inclusion(func(drinkBeer, x), func(n(study), x)))

Question

fmi and take5 and drinkBeer => brilliant
forevery(x, inclusion(and(func(fmi, x), and(func(take5, x), func(drinkBeer)), func(brilliant, x)))

Conversion To CNF

resolution_from_CNF([[n(passExam), brilliant, lucky], [n(brilliant), n(lucky)],[n(lucky), n(brilliant)], [n(take5), passExam], [n(fmi), n(lucky)], [n(drinkBeer), n(study)], [n(fmi), n(take5), n(drinkBeer), brilliant]]) – False.

Exercise 2 - SAT Solver DP

Simulation:

[[toddler],[n(toddler),child],[n(child),n(male),boy],[n(infant),child],[n(child),n(female),girl],[female],[girl]]
min app: [[toddler/true],[n(male)/true],[n(infant)/true],[child/true],[n(female)/false],[girl/true]]
max app: [[child/true],[girl/true],[toddler/true],[n(male)/true],[female/true]]

[[toddler],[n(toddler),child],[n(child),n(male),boy],[n(infant),child],[n(child),n(female),girl],[female],[n(girl)]]
min app-> false
max app-> false

[[n(a),b],[c,d],[n(d),b],[n(c),b],[n(b)]]
min app-> false
max app-> false

[[n(b),a],[n(a),b,e],[e],[a,n(e)],[n(a)]]
min app-> false
max app-> false

[[n(a),n(e),b],[n(d),e,n(b)],[n(e),f,n(b)],[f,n(a),e],[e,f,n(b)]]
min app: [[b/true],[n(d)/true],[n(e)/true],[n(a)/true],[f/true]]
max app: [[e/true],[n(a)/true],[f/true]]

[[a,b],[n(a),n(b)],[n(a),b],[a,n(b)]]
min app-> false
max app-> false

Observation: Only some variables were assigned boolean values (the other ones can have any value).

The method for chosing the atom were based on:

- The most frequent p in clauses.
- The least frequent p in clauses.

An obvious observation is that by choosing the most frequent atom p with, the algorithm finishes faster (as there will be lots of removed clauses after each step).