Udrea Iulia-Maria
Group 407
09.12.2024

Project 1 – KRR

1. Subject 1 – Resolution

a). For this subject, I chose the following knowledge base, written in natural language:

1. Anyone who exercises and eats well is fit.
2. Anyone who is vegetarian eats well.
3. Anyone who is fit can hike.
4. Anyone who goes to the gym exercises or is a trainer.
5. No trainer has an office job.

Q: Anyone who has an office job, is vegetarian and exercises can hike.

This knowledge base has the following representation in FOL:

1. $\forall x.\left(\text{Exercises}(x) \wedge \text{EatsWell}(x) \supset \text{Fit}(x)\right)$
2. $\forall x.(\text{Vegetarian}(x) \supset \text{EatsWell}(x))$
3. $\forall x.(\text{Fit}(x) \supset \text{CanHike}(x))$
4. $\forall x.(\text{GoesToGym}(x) \supset (\text{Exercises}(x) \vee \text{Trainer}(x)))$
5. $\forall x.(\text{Trainer}(x) \supset \neg\text{HasOfficeJob}(x))$

Q: $\forall x.(\text{HasOfficeJob}(x) \wedge \text{GoesToGym}(x) \wedge \text{Vegetarian}(x)) \supset \text{CanHike}(x)$

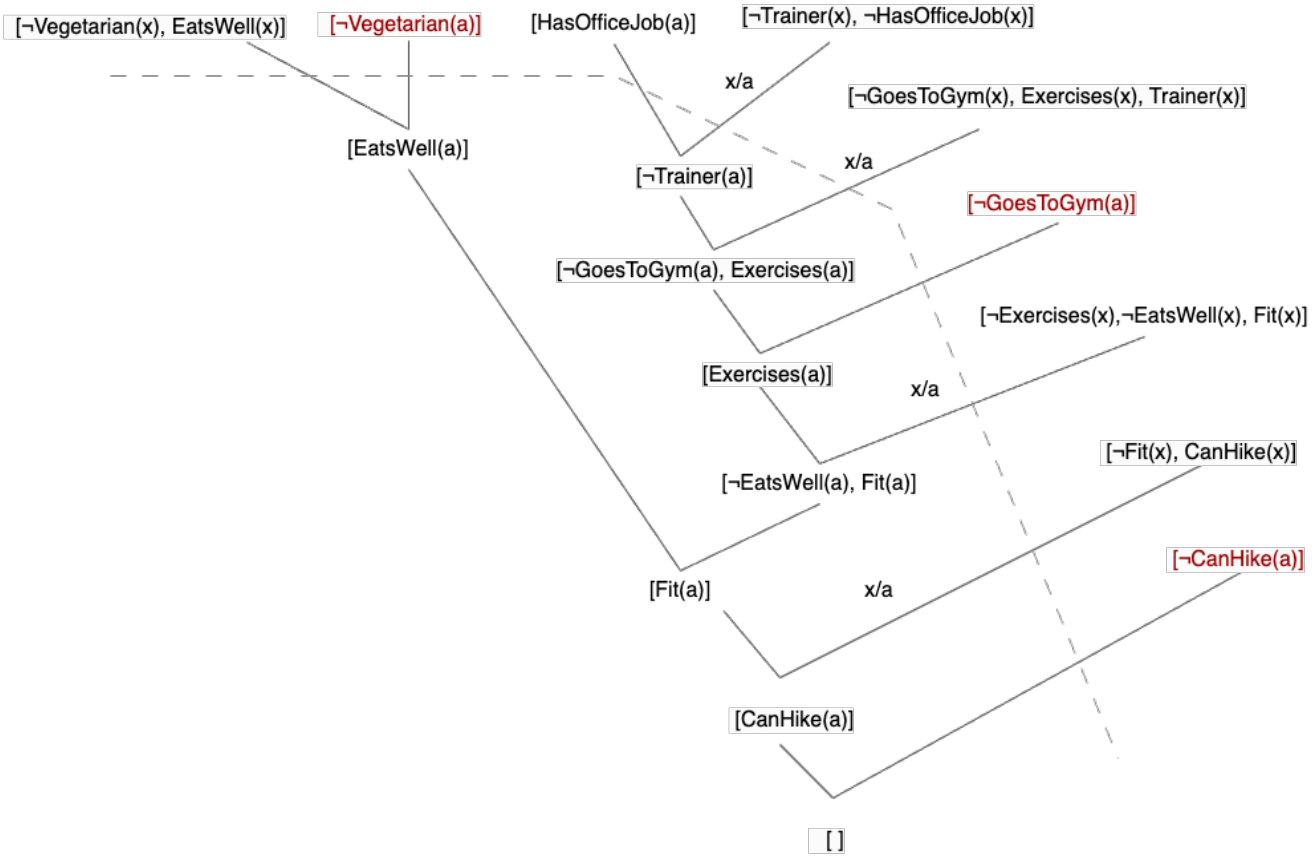
b). To demonstrate that the given question is logically entailed from the KB using the resolution method, we must prove that KB U $\{\neg Q\}$ is unsatisfiable. First, it is necessary to transform the knowledge base and the negated question into Conjunctive Normal Form (CNF).
Using the equivalence $\alpha \supset \beta$ can be rewritten as $(\neg\alpha \vee \beta)$, the KB in FOL is rewritten as follows:

1. $[\neg\text{Exercises}(x), \neg\text{EatsWell}(x), \text{Fit}(x)]$
2. $[\neg\text{Vegetarian}(x), \text{EatsWell}(x)]$
3. $[\neg\text{Fit}(x), \text{CanHike}(x)]$
4. $[\neg\text{GoesToGym}(x), \text{Exercises}(x), \text{Trainer}(x)]$
5. $[\neg\text{Trainer}(x), \neg\text{HasOfficeJob}(x)]$

Negating the implication $\neg(\alpha \supset \beta)$, which is rewritten as $\neg(\neg\alpha \vee \beta)$ and then simplified to $\alpha \wedge \neg\beta$, we obtain: $\neg Q$: $\exists x.\ \text{HasOfficeJob}(x) \wedge \text{GoesToGym}(x) \wedge \text{Vegetarian}(x) \wedge \neg\text{CanHike}(x)$

To replace the existential variable, we introduce a as a Skolem constant, resulting in the following clauses: $[\text{HasOfficeJob}(a)]$, $[\text{GoesToGym}(a)]$, $[\text{Vegetarian}(a)]$, $[\neg\text{CanHike}(a)]$

Now, we can proceed to apply the resolution method.



We obtained the empty clause, which means that the knowledge base is unsatisfiable, and therefore the knowledge base logically entails the question.

c). **Short description of the code**: The code reads data from a file and uses grammars to transform the input text, formatted according to the examples provided in the assignment, into a list of clauses that can be processed by the resolution algorithm. The algorithm supports variables and can unify them with constants. A dynamic predicate is used to check whether a newly generated clause has already been derived; if so, it is not added again to the knowledge base. Additionally, a predicate with three parameters (clause1, clause2, literal) is employed to track the clauses and literals on which resolution has already been applied, preventing redundant operations.

When running the algorithm on the provided examples and the custom example, the result is **UNSATISFIABLE**, indicating that the questions are **logically entailed** from the knowledge bases.

d). The results obtained by running the code on the provided examples are as follows:

Index	Clause	Algorithm Result
i	(¬a or b) and (c or d) and (¬d or b) and ¬b and (¬c or b) and e and (f or a or b or ¬f)	UNSATISFIABLE
ii	(¬b or a) and (¬a or b or e) and (a or ¬e) and ¬a and e	UNSATISFIABLE
iii	(¬a or b) and (c or f) and ¬c and (¬f or b) and (¬c or b)	SATISFIABLE
iv	(a or b) and (¬a or ¬b) and c	SATISFIABLE

The implemented code detects tautologies, subsumed clauses, and pure clauses. For instance, in set (i), atom e is identified because $not(e)$ does not appear in any other clause. Consequently, the clause $[e]$ can be eliminated. Additionally, in the same set (i), the clause $(f$ or a or b or $\neg f)$ contains both the atom f and its negation $\neg f$, which constitutes a tautology. Therefore, this clause can also be eliminated.

2. Subject 2 – SAT solver

The implementation reads data from a file and uses the grammars from the previous problem to transform the input into the desired format. For this task, the two strategies are: choosing atoms from **the shortest clause** and selecting **the most frequently occurring atom**. The comparison between these strategies will focus on the the outcomes produced by the solver, the number of atoms evaluated before satisfiability is determined and the runtime for each approach.

Example i).

```
toddler and (¬toddler or child) and (¬child or ¬male or boy) and
(¬infant or child) and (¬child or ¬female or girl) and female and girl
```

The following table summarizes the results obtained for each strategy on this example, including the steps and execution time. Both strategies exhibit similar execution times; however, in this case, the **Shortest Clause** strategy identifies a solution with fewer atoms compared to the alternative strategy.

Strategy	Result	Steps	Time
Shortest Clause	YES {toddler/true, child/true, female/true, girl/true, not(male)/true}	5	5ms
Most Frequent Atom	YES {child/true, girl/true, toddler/true, male/true, female/true, boy/true}	6	5ms

The subsequent tables analyze how each method traverses the set of propositions to reach a result. For each step, the tables display the initial set of propositions, the selected atom and its truth value (true or false), and the resulting set of propositions after eliminating clauses and atoms according to the algorithm. For the **Shortest Clause** method, we obtain the following:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	toddler/true	[[child], [female], [girl], [not(child), not(female), girl], [not(child), not(male), boy], [not(infant), child]]
2	Resulting Set 1	child/true	[[female], [girl], [not(female), girl], [not(male), boy]]
3	Resulting Set 2	female/true	[[girl], [not(male), boy]]
4	Resulting Set 3	girl/true	[[not(male), boy]]
5	Resulting Set 4	not(male)/true	[]

For the **Most Frequent Atom** method, we obtain the following results:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	child/true	[[female], [girl], [toddler], [not(female), girl], [not(male), boy]]
2	Resulting Set 1	girl/true	[[female], [toddler], [not(male), boy]]
3	Resulting Set 2	toddler/true	[[female], [not(male), boy]]
4	Resulting Set 3	male/true	[[boy], [female]]
5	Resulting Set 4	female/true	[[boy]]
6	Resulting Set 5	boy/true	[]

Example ii).

```
toddler and (¬toddler or child) and (¬child or ¬male or boy) and
(¬infant or child) and (¬child or ¬female or girl) and female and ¬girl
```

Both methods produce the result **NOT** in the Davis Putnam algorithm meaning that there is no combination of assignments that can satisfy all the clauses. For this example, the **Most Frequent Atom** strategy was faster in determining unsatisfiability, even though the number of atoms evaluated is the same, as shown in the following table.

Strategy	Result	Steps	Time
Shortest Clause	NOT	8	11ms
Most Frequent Atom	NOT	8	9ms

The following table illustrates the steps taken by the algorithm when using the **Shortest Clause** strategy. Whenever the resulting set contains an empty clause for a specific assignment, it indicates that this assignment cannot be part of the solution. The method is applied recursively until it is determined that all variables selected by the algorithm, regardless of their truth values, eventually lead to an empty clause.

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	toddler/true	[[child], [female], [not(child), not(female), girl], [not(child), not(male), boy], [not(girl)], [not(infant), child]]
2	Resulting Set 1	child/true	[[female], [not(female), girl], [not(girl)], [not(male), boy]]
3	Resulting Set 2	female/true	[[girl], [not(girl)], [not(male), boy]]
4	Resulting Set 3	girl/true	[[], [not(male), boy]]

Step	Starting Set	Atom/Assignment	Resulting Set
5	Resulting Set 3	girl/false	[[], [not(male), boy]]
6	Resulting Set 2	female/true	[[], [not(girl)], [not(male), boy]]
7	Resulting Set 1	child/false	[[], [female], [not(girl)], [not(infant)]]
8	Initial Set	toddler/false	[[], [female], [not(child), not(female), girl], [not(child), not(male), boy], [not(girl)], [not(infant), child]]

The steps for the **Most Frequent Atom** strategy for example ii are as follows:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	child/true	[[female], [toddler], [not(female), girl], [not(girl)], [not(male), boy]]
2	Resulting Set 1	girl/true	[[], [female], [toddler], [not(male), boy]]
3	Resulting Set 1	girl/false	[[female], [toddler], [not(female)], [not(male), boy]]
4	Resulting Set 3	female/true	[[], [toddler], [not(male), boy]]
5	Resulting Set 3	female/false	[[], [toddler], [not(male), boy]]
6	Initial Set	child/false	[[female], [toddler], [not(girl)], [not(infant)], [not(toddler)]]
7	Resulting Set 6	toddler/true	[[],[female],[not(girl)],[not(infant)]]
8	Resulting Set 6	toddler/false	[[],[female],[not(girl)],[not(infant)]]

Example iii).

```
(¬a or b) and (c or d) and (¬d or b) and (¬c or b) and ¬b and e and
(f or a or b or ¬f)
```

This example is also unsatisfiable and lead to the **NOT** result. In this case, we observe that due to the presence of small clauses with few atoms in common, the **Shortest Clause** method reaches the result in fewer steps.

Strategy	Result	Steps	Time
Shortest Clause	NOT	10	12ms
Most Frequent Atom	NOT	12	14ms

Running the code on example iii) using the **Shortest Clause** strategy, we obtain the following steps:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	not(b)/true	[[c,d],[e],[f,a,not(f)],[not(a)],[not(c)],[not(d)]]
2	Resulting Set 1	e/true	[[c,d],[f,a,not(f)],[not(a)],[not(c)],[not(d)]]
3	Resulting Set 2	not(a)/true	[[c,d],[f,not(f)],[not(c)],[not(d)]]
4	Resulting Set 3	not(c)/true	[[d],[f,not(f)],[not(d)]]
5	Resulting Set 4	d/true	[[],[f,not(f)]]
6	Resulting Set 4	d/false	[[],[f,not(f)]]
7	Resulting Set 3	not(c)/false	[[],[f,not(f)],[not(d)]]
8	Resulting Set 2	not(a)/false	[[],[c,d],[not(c)],[not(d)]]
9	Resulting Set 1	e/false	[[],[c,d],[f,a,not(f)],[not(a)],[not(c)],[not(d)]]
10	Initial Set	not(b)/false	[[],[c,d],[e]]

For the **Most Frequent Atom** strategy, the steps are:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	b/true	[[], [c, d], [e]]
2	Initial Set	b/false	[[c, d], [e], [f, a, not(f)], [not(a)], [not(c)], [not(d)]]
3	Resulting Set 2	f/true	[[c,d],[e],[not(a)],[not(c)],[not(d)]]
4	Resulting Set 3	d/true	[[],[e],[not(a)],[not(c)]]
5	Resulting Set 3	d/false	[[c],[e],[not(a)],[not(c)]]
6	Resulting Set 5	c/true	[[],[e],[not(a)]]
7	Resulting Set 5	c/false	[[],[e],[not(a)]]
8	Resulting Set 2	f/false	[[c,d],[e],[not(a)],[not(c)],[not(d)]]
9	Resulting Set 8	d/true	[[],[e],[not(a)],[not(c)]]
10	Resulting Set 8	d/false	[[c],[e],[not(a)],[not(c)]]
11	Resulting Set 10	c/true	[[],[e],[not(a)]]
12	Resulting Set 10	c/false	[[],[e],[not(a)]]

Example iv).

`(¬b or a) and (¬a or b or e) and e and (a or ¬e) and ¬a`

In terms of the number of steps, both strategies are equally efficient, as they evaluate the same atoms but in a different order.

Strategy	Result	Steps	Time
Shortest Clause	NOT	4	4ms
Most Frequent Atom	NOT	4	4ms

For **Shortest Clause**, the order is:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	e/true	[[a],[not(a)],[not(b),a]]
2	Resulting Set 1	a/true	[[]]
3	Resulting Set 1	a/false	[[],[not(b)]]
4	Initial Set	e/true	[[],[not(a)],[not(a),b],[not(b),a]]

For **Most Frequent Atom**, the order is:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	a/true	[[],[b,e],[e]]
2	Initial Set	a/false	[[e],[not(b)],[not(e)]]
3	Resulting Set 2	e/true	[[],[not(b)]]
4	Resulting Set 2	e/false	[[],[not(b)]]

Example v).

`(¬a or ¬e or b) and (¬d or e or ¬b) and (¬e or f or ¬b) and (f or ¬a or e) and (e or f or ¬b)`

Compared to the previous example, which is shorter and contains smaller clauses, this example is solved quickly by both methods because all atoms appear in all clauses. This allows reductions to be applied universally. In this case, both strategies produce equally good results.

Strategy	Result	Steps	Time
Shortest Clause	YES {not(a)/true, e/true, f/true}	3	3ms
Most Frequent Atom	YES {e/true, b/true, f/true}	3	2ms

For the **Shortest Clause** strategy, the steps for this example are:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	not(a)/true	[[e,f,not(b)],[not(d),e,not(b)],[not(e),f,not(b)]]
2	Resulting Set 1	e/true	[[f,not(b)]]
3	Resulting Set 2	f/true	[ ]

The **Most Frequent Atom** method reaches the same result, but the set of propositions decreases more quickly compared to the **Shortest Clause** method in this case:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	e/true	[[f,not(b)],[not(a),b]]
2	Resulting Set 1	b/true	[[f]]
3	Resulting Set 2	f/true	[ ]

Example vi).

`(a or b) and (¬a or ¬b) and (¬a or b) and (a or ¬b)`

In this example, we can observe the largest difference in the number of steps between the two methods.

Strategy	Result	Steps	Time
Shortest Clause	NOT	12	7ms
Most Frequent Atom	NOT	6	4ms

Running the code on example vi) using the **Shortest Clause** strategy, we obtain the following steps:

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	a/true	[[b],[not(b)]]

Step	Starting Set	Atom/Assignment	Resulting Set
2	Resulting Set 1	b/true	[[]]
3	Resulting Set 1	b/false	[[]]
4	Initial Set	a/false	[[b],[not(b)]]
5	Resulting Set 4	b/true	[[]]
6	Resulting Set 4	b/false	[[]]
7	Initial Set	b/true	[[a],[not(a)]]
8	Resulting Set 7	a/true	[[]]
9	Resulting Set 7	a/false	[[]]
10	Initial Set	b/false	[[a],[not(a)]]
11	Resulting Set 10	a/true	[[]]
12	Resulting Set 10	a/true	[[]]

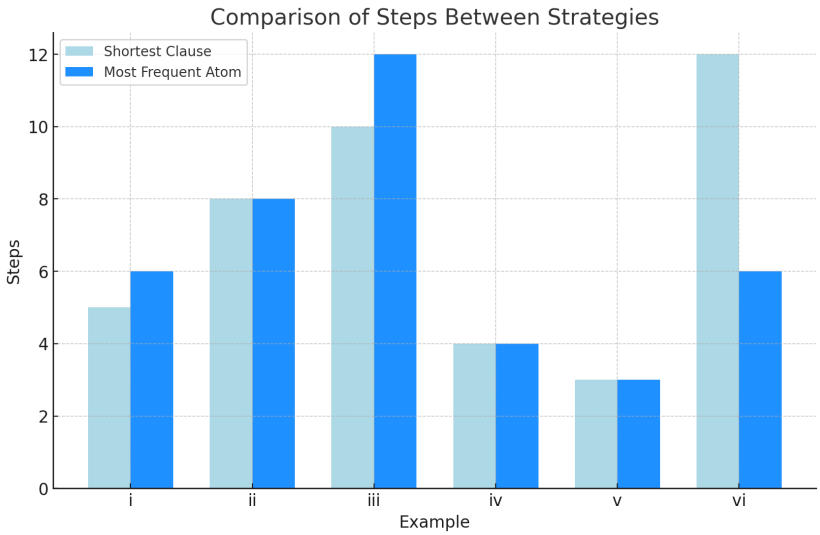
Although the **Shortest Clause** method already checks combinations of values for a and b, it repeats steps unnecessarily. By applying the **Most Frequent Atom** method, the result is obtained in fewer steps, as all value pairs are assigned only once.

Step	Starting Set	Atom/Assignment	Resulting Set
1	Initial Set	b/true	[[a],[not(a)]]
2	Resulting Set 1	a/true	[[]]
3	Resulting Set 1	a/false	[[]]
4	Initial Set	b/false	[[a],[not(a)]]
5	Resulting Set 4	a/true	[[]]
6	Resulting Set 4	a/false	[[]]

Conclusions about the strategies

The **Shortest Clause** strategy prioritizes unit clauses, which allows it to detect contradictions more quickly. This focus on smaller clauses often leads to faster identification of unsatisfiability in cases where such contradictions exist early in the process. However, the method may end up exploring more branches, which can result in a higher number of steps overall, especially in scenarios with a large number of interconnected clauses.

The **Most Frequent Atom** strategy, on the other hand, efficiently reduces the size of the knowledge base by eliminating multiple clauses at the same time. This makes it particularly effective for large knowledge bases with long clauses. However, its focus on frequent atoms might lead to overlooking atomic clauses, potentially causing the algorithm to explore unnecessary branches that could have been avoided by resolving unit clauses earlier.



Neither strategy is universally better; their relative efficiency depends on the specific structure of the clauses and the distribution of atoms.