Danmarks Tekniske Universitet DTU

# Belief Revision Assignment

**Authors – Group 19**

Bryndís Rósa Sigurpálsdóttir - s212423
Eydís Sjöfn Kjærbo - s212943
Juliette Marie Victoire Vlieghe – s213705
Michal Marek Kossakowski – s220393

Technical University of Denmark
Course 02180 – Introduction to AI

May 19, 2023

# Contents

# 1 Introduction to the problem of belief revision

Belief revision is a hot topic in theoretical computer science and logic, as well as a major research area in artificial intelligence. The ability to revise beliefs allows for the alteration of information repositories in response to new knowledge. These procedures enable the insertion of new information into a knowledge base without jeopardizing its integrity. There are two main approaches to belief revision: AGM style and reason-maintenance style. AGM style belief revision mandates that modifications to a belief base be as minor as possible, whereas reason-maintenance style belief revision records how beliefs support each other and uses this knowledge to render conflicting beliefs underivable. Beliefs in the AGM model are expressed by sentences in a formal language. Sentences do not capture all features of belief, but they are currently the best broad sense representation available. A set of such belief-representing sentences represents an agent's beliefs. This set is commonly believed to be closed under logical consequence, which means that every statement that follows logically from this set is already in the set. This paper will focus on the AGM belief revision approach, which contains three forms of belief change, expansion, contraction, and revision.[1].

The issue with belief revision is that when an intelligent agent receives new information that opposes prior beliefs and often takes priority over them, logical formulae that express those beliefs must be revised. However, the logical consideration alone does not tell you which beliefs to revise; this must be determined by other techniques. What complicates matters is that beliefs in a database have logical consequences. So, while giving up a belief, you must also determine which of its consequences to keep and which to discard [2, 3, 4].

The purpose of this report is to demonstrate how we attempt to design and implement a belief revision engine composed of an agent's beliefs and a mechanism that outputs the revised belief based on a propositional formula input. The following parts will be included in this paper. In Section 2, we will detail the design and implementation decisions we took based on the methodology presented in this course. Section 3 will go through everything we learned when constructing this project, as well as the conclusion and future work.

# 2 The belief revision agent

This section will cover the formalisms and design choices of the Belief revision agent. It will be divided into four sections, the first of which will describe the design and implementation of the belief base, the second of will document how we designed and implemented a method for checking logical entailment, the third will explain how we implemented the belief base contraction, and the fourth will discuss how we implemented the belief base expansion.

## 2.1 Belief base

We have designed our belief base following the framework adopted in the article "Applications of Belief Revision" by Mary-Anne Williams [3]. It is a formal framework which incorporates AGM postulates and follows the principle of Minimal Change. We have designed our belief base as a belief set containing beliefs stored as formulas in propositional logic. In addition, each belief also contains a priority value which, when faced with a choice, our belief revision agent can refer to. Each formula in the belief set is, therefore, assigned a priority value from 0 to 1. The entire belief revision engine was implemented using Python, with the addition of Sympy - a library for symbolic mathematics [5].

## 2.2 Logical entailment

Logical entailment occurs when one assertion logically follows from another.[6] For example, if $a \wedge b$ is in the belief base, then our belief base entails both a and b. In order to make this task easier we used the CNF formula from the sympy package. This formula accepts a statement and returns the CNF representation of that statement. In CNF form, a statement is a conjunction of clauses, and each clause is a disjunction of literals.

Our entailment method is primarily based on three techniques: IS-CONTRADICTION, IS-TAUTOLOGY, and CHECK-FOR-INCONSISTENCY. The method IS-CONTRADICTION determines whether a given sentence contains a contradiction and returns true if it does, false otherwise. A statement that is always wrong is said to be a contradiction. The method IS-TAUTOLOGY determines whether a given statement contains a tautology and returns true if it does, false otherwise. A tautology is a statement that is always true. The method CHECK-FOR-INCONSISTENCY takes in a list of statements and checks if they cause a contradiction, returning true if there is a contradiction, or false otherwise.

**Algorithm 1** Entailment

```
procedure ENTAILS(B, $formula$)
    if $formula = \emptyset$ then
        return True
    else if B.IS-CONTRADICTION($formula$) then
        return False
    else if B.IS-TAUTOLOGY($formula$) then
        return True
    else if not B.IS-EMPTY then
        Join statements in the belief base and the formula with $\wedge$.
        Change statement to CNF format
        Divide the CNF statement into parts of $\vee$ statements
        if more than one $\vee$ statement then
            return B.CHECK-FOR-INCONSISTENCY(list of $\vee$ statements)
    return False
function IS-TAUTOLOGY(B, $formula$)
    return B.IS-CONTRADICTION(TO-CNF($\neg$ $formula$))
```

When we call the method ENTAILS (Algorithm 1), we want to begin by inspecting the input. If the input is empty we can return that the belief base entails it. If the input contains a contradiction, our belief base should not entail it, however if the input is a tautology, the belief base should entail it. If the belief base is empty, we know that it does not entail the input because it is neither a contradiction nor a tautology. If this is not the case, we may see if combining the belief base and the ¬ formula with a conjunction results in a contradiction; if it does, the belief base entails the input. This is known as proof by contradiction. All other scenarios will imply that the belief base does not entail the input.

We also included the technique IS-TAUTOLOGY in algorithm 1. In this one-liner, we call the method IS_CONTRADICTION and determine whether the ¬ formula will result in a contradiction. Because tautologies are always valid, the negation of a tautology should always be invalid, resulting in a contradiction.

**Algorithm 2** Contradiction

```
procedure IS-CONTRADICTION(B, $formula$)
    if $formula = \emptyset$ then
        return True
    Change formula to CNF format
    Divide the CNF statement into parts of $\vee$ statements
    if more than one $\vee$ statement then
        return B.CHECK-FOR-INCONSISTENCY(list of $\vee$ statements)
    return False
```

In algorithm 2, we start by checking if the input is an empty set. This might occur when a contradiction is changed to a CNF format. As in algorithm 1, we combine the belief base and the ¬ formula with a conjunction and call the method CHECK-FOR-INCONSISTENCY to check if it results in a contradiction.

**Algorithm 3** Check for Inconsistency

```
procedure CHECK-FOR-INCONSISTENCY(B, $list$)
    for <i in $list$> do
        for <j in $list[i+1:]$> do
            if i = ¬j then
                return True
        if length of list > 2 then
            remove statement i from the list if it is a tautology
            check if there are long $\vee$ statements that cause inconsistency
            check if we can remove letters from a $\vee$ statement and alter the list
            check if everything in the $\vee$ statement is invalid, causing inconsistency
    return False
```

In the algorithm 3, we go through a few steps and check for inconsistencies; if we discover any, we stop and return our results. Our input consists of a list of disjunctions. We begin by going through the list and looking for any contradictions between disjunctions such as ¬ a and a. If this is not the case, we examine each disjunction individually.

We determine whether the disjunction is a tautology and delete it from the list if it is. We eliminate it since the list initially contained only one conjunction, and a valid statement in the conjunction will not cause it to become invalid, therefore this statement is irrelevant.

We then look for any long disjunctions that create contradiction, such as $a \vee b$, $a \vee \neg b$, $\neg a \vee b$ and $\neg a \vee \neg b$.

Then we check if we can remove literals from each disjunction, for example, if we have $a \wedge (\neg a \vee b \vee c)$, we can remove $\neg a$ from the disjunction because we know that $b \vee c$ has to be valid for our statement to be valid. If we can remove from the disjunction and the remaining disjunction isn't already in the list, We add the new disjunction to the list and call the method recursively with our new list.

Finally, we check to see if everything in the disjunction is invalid, resulting in a contradiction in the original conjunction.

If we find no contradictions during this process we can return that there are no contradictions in this list.

## 2.3 Expansion

Expansion represents the most basic modification to a knowledge base. It entails accepting information without removing previously accepted information, which may result in an inconsistent knowledge base [3].

In our approach, the expansion contractor EXPAND(B, $\phi$, newPriority) is implemented as follows: is the base of the belief set, $\phi$ is the belief that is being added to the base, and priority is the intended order of $\phi$. According to AGM definitions of expansion operators, every new belief is accepted even if it leads to an inconsistent belief set. As a result, in our method, we do not evaluate logical consistency with the existing base. Since we don't want $\phi$ repetitions in our belief base, we start by checking to see whether the $\phi$ occurs in the existing base. If that is not the case, we proceed to add it to the belief base with the $newPriority$. However, if the belief is already present in the base, we want to know what its priority order should really be. This is done to maintain consistency with the rest of the entrenchment, as we believe that beliefs with higher priority order should be the ones that are less likely to be retracted [7, 8]. This is achieved by utilizing the DEGREE(B, $\phi$) method, which effectively lookups of the belief's current priority. If $\phi$ already has a priority order equal to or greater than the incoming priority $newPriority$, no modifications to the belief base will be made. However, if $\phi$'s original priority order is lower than the incoming $newPriority$ order, $\phi$ is formally added to the base with the incoming new $newPriority$ order.

**Algorithm 4** Expansion

```
procedure EXPAND(B, $\phi$, newPriority)
    if not ENTAILS($\phi$) then
        Incoming belief does not exist in the base
        New belief is added to the base, with the newPriority.
    else
        Incoming belief is already in the base
        for belief in B do
            if belief.formula==$\phi$ then
                if belief.priority > newPriority then
                    The existing priority in the base is higher so we no nothing
                if belief.priority<=newPriority and newPriority>B.degree($\phi$)) then
                    The belief in the base it deleted
                    The belief is added again in the base with the newPriority
```

## 2.4 Contraction of belief base

We use the following definition of the degree of acceptance, adapted from [3]:

**Definition 1** (degree of acceptance)**.** The degree of acceptance of a set of ranked beliefs **B** is the following:

$$\begin{cases} 0 & \text{if } \mathbf{B} \not\models \phi \\ 1 & \text{if } \phi \text{ is a tautology, i.e } \neg\phi \text{ is not satisfiable} \\ j & \text{largest } j \text{ such that } \mathbf{B} \models \phi \text{ otherwise} \end{cases}$$

In other words, the degree of acceptance of a sentence is higher when we only need highly plausible beliefs to derive it.

We propose the following procedure for contraction (algorithm 5): if B does not entail the formula to contract, no further action s required. Otherwise, we reconstruct a belief base. For each belief, starting with the most plausible ones, we check if adding them to the belief base would result in a belief base that entails the formula to contract. In that case they are discarded, otherwise we add them.

**Algorithm 5** Contraction

```
procedure CONTRACT($\mathbf{B}$, $\phi$, $priority = 1$)
    if IS-TAUTOLOGY($\phi$) then
        Raise Priority Error: Cannot remove a tautology from the belief base.
    else if $B.degree(\phi) > priority$ then
        Raise Priority Error: priority is too low to perform contraction.
    else if $\mathbf{B} \vDash \phi$ then
        Consider the belief base $\mathbf{B}$ sorted by descending priority.
        Consider an empty belief base $\mathbf{B}'$.
        for belief in $\mathbf{B}$ do
            if $\mathbf{B}' \cup belief.formula \nvDash \phi$ then
                $\mathbf{B}'$.EXPAND($belief$)
        $\mathbf{B} \leftarrow \mathbf{B}'$
=0
```

This requires some modifications in case we input a conjunction: if we add $a \wedge b$ in the belief base and contract $b$, we don't want to remove the entire belief; we want to keep "a" so that the recovery principle is respected. For this purpose, we do the following (algorithm 6): we successively replace each atom by True and get the result. For example, for $\phi = a \wedge (b \vee c)$, by replacing $a$, $b$, $c$, we respectively get $(b \vee c)$, $a$, $a$ and remove the duplicate formula.

**Algorithm 6** Split

```
function SPLIT($\phi$)
    if not IS-CONJUNCTION($\phi$) then formulas = [$\phi$]
    else
        atoms = atoms(formulas)
        for a in atoms do
            Replace a by True and add the result in formulas
        remove duplicate formulas
    return formulas
```

So instead of merging one belief at the time, we split each belief and try to merge each sub-beliefs. For user's convenience, if all the sub-beliefs are valid, we keep the original belief instead. To keep the pseudocode simple and readable, we can assume that we split each belief a priori, however the actual code does not make this assumption and can handle conjunction.

## 2.5 Revision of the belief base

We propose the following procedure for revision (algorithm 7): we try to contract the negation of the formula from the belief base, which solves potential conflicts if the formula is plausible enough, then expand the belief base with the new belief. We decide not to add inconsistent formulas in the belief base, and we will therefore discard the formula given if it contradicts itself.

**Algorithm 7** Revision

```
procedure REVISION($\mathbf{B}$, $\phi$, $priority = 1$)
    if $is - contradiction(\phi)$ then
        Raise Error: The formula is not satisfiable.
    else
        try
            $\mathbf{B}$.CONTRACT($\neg\phi$, $priority = priority$)
            $\mathbf{B}$.EXPAND($\phi$)
        catch Priority Error
            Raise Priority Error: The formula conflicts with more plausible beliefs.
        end try
function IS-CONTRADICTION($\phi$)
    return $\neg$IS-SATISFIABLE($\phi$)
```

## 2.6 Testing algorithms with AGM postulates

In this chapter, we will demonstrate the results of tests performed on our implemented belief revision engine. We implemented dedicated tests for chosen AGM postulates for both contraction and revision. The tests were executed on sample data which was prepared individually for each postulate.

### 2.6.1 Testing contraction

- Closure

  Closure was tested by expanding an empty belief base and contracting it using the same formula: $(p \land q)$. If the resulting belief base entailed itself, meaning it was logically closed, then the test returned a positive result.

- Success

  Success was tested by expanding an empty belief base and contracting it using the same formula: $(p \land q)$. If the resulting belief base did not entail the given formula, the test returned a positive result.

- Inclusion

  Inclusion was tested by expanding an empty belief base with $p$ and $q$, and later contracting $p$. If the contracted belief base was a subset of the belief base before contraction, the test returned a positive result.

- Vacuity

Vacuity was tested by expanding an empty belief base with $p$ and contracting $q$ from it. If the belief base was the exact same set as before the contraction, the test returned a positive result.

- Extensionality

  Extensionality was tested by first defining two equivalent formulas: $(p \to q)$ and $(\neg p \lor q)$. An empty belief base was then expanded by the first formula and a copy of the expanded belief base was created. The first formula was contracted from the original belief base and the second - from the copy. If both belief bases where the exact same sets of beliefs, the test returned a positive result.

- Recovery

  Recovery was tested by expanding an empty belief base by $(p \land q)$. A copy of the expanded belief base was made. The same formula was then contracted from the copy and used again to expand it. If the copy of the belief base entailed the original, the test returned a positive result.

| AGM postulate | Closure | Success | Inclusion | Vacuity | Extensionality | Recovery |
|---|---|---|---|---|---|---|
| Test result | Positive | Positive | Positive | Positive | Positive | Positive |

Table 1: Test results of AGM postulates for contraction

Recovery proves to be a highly debatable postulate and when given the right formulas it can yeild implausible results. As an example, an empty belief base was given two true formulas: p and q. A copy of the expanded belief base was made and the formula p | q was then contracted from it and used again for expansion. The original belief base with formulas p and q did not entail the modified copy.

### 2.6.2 Testing revision

- Closure

  Closure was tested by revising formula $q$ in an empty belief base. If the belief base entailed itself, meaning it was logically closed, then the test returned a positive result.

- Success

  Success was tested by revising formula $q$ in an empty belief base. If the belief base entailed the given formula, then the test returned a positive result.

- Inclusion

  Inclusion was tested by revising one empty belief base by the formula $q$ and expanding another empty belief base with the same formula. If the revised belief base was a subset of the expanded one, then the test returned a positive result.

- Vacuity

  Vacuity was tested by first expanding en empty belief base by the formula $(p \vee q)$. A copy of the expanded belief base was made. The original belief base was revised with that formula and the copy was expanded with it. If the revised belief base was a subset of the expanded one, then the test returned a positive result.

- Consistency

  Consistency was tested by creating a consistent formula $(q \wedge p)$ and revising an empty (consistent) belief base with it. If the belief base was still consistent after revision, then the test returned a positive result.

- Extensionality

  Extensionality was tested by creating two equivalent formulas $p = (a \rightarrow b)$ and $q = (\neg a \vee b)$ and revising one empty belief base with $p$ and another with $q$. If both belief bases entailed one another, then the test returned a positive result.

| AGM postulate | Closure | Success | Inclusion | Vacuity | Consistency | Extensionality |
|---|---|---|---|---|---|---|
| Test result | Positive | Positive | Positive | Positive | Positive | Positive |

Table 2: Test results of AGM postulates for revision

Every implemented test for the chosen AGM postulates yeilded a positive result. We have succeeded in creating a belief revision agent that follows the AGM postulates of rational belief revision. We did not however manage to implement all AGM postulates. The ones that were omitted were for example conjunctive inclusion and conjunctive overlap for contraction, and superexpansion and subexpansion for revision.

# 3 Conclusion

## 3.1 What we've learnt along the way

We had a conceptual knowledge of the fundamentals of belief revision agent design before the beginning of this assignment, owing to the study material offered in this course. However, this project allowed us to go a step further and design our own algorithm for belief revision, which showed us how to implement the concepts in designing the belief revision agent.

This project taught us more about what has to be done to ensure that a new belief does not jeopardize the integrity of the belief foundation, as well as how difficult it was to implement some of the algorithm's components. One of the most challenging aspects of implementing the project was converting our understanding of the approaches to a manner that the computer could really utilize and accept. We also realized at one time how important it is to utilize debugging or just have someone else test the function you have been implementing; this greatly aids in the discovery of errors in the code.

We also discovered that working in groups is more efficient since two sets of eyes are looking at the same problem at the same time. This enhanced the component's design and code.

## 3.2 Conclusion and Future work

This report details how we designed and implemented a belief revision engine composed of an agent's beliefs and a mechanism that outputs the revised belief based on a propositional formula input.

We had the belief revision agent run through a sequence of stages as specified in the project description, and at each stage, we had to explain how we designed and implemented the method included in that stage. The first stage was the belief base, the second was the logical entailment, the third was the contraction of the belief base, and the final was the expansion of the belief base. In addition, we added a section in our report that described how we tested the algorithm. The belief base has a formal framework which incorporates AGM postulates and follows the principle of Minimal Chang in addition to that it also contains priority value. The logical entailment method is based on tree techniques is-contradiction, is-tautology, and check-for-inconsistency. In the expansion method every belief is accepted even though it could lead to inconsistency.

Most models of belief change, including our system, are represented as sentences in some formal language because it is the best general-purpose encoding currently available; however, because actual beliefs are not represented by sentences, it is important for future work to use something different for the beliefs in order to capture all aspects of a belief.

Also, the system we designed is not very user-friendly and assumes that the agent has a good knowledge of formal languages and logic; therefore, for future work, it would be preferable to assist the user in adding a new belief to the database with some guidelines and accept more diverse types of input from the user. Furthermore, for future work, it would be useful to include a technique for emptying the belief base as well as a method for determining the priority of a belief; this would also assist to make the designed agent more user-friendly.

## References

[1] A. Jensen and J. Villadsen, "Plan-belief revision in jason," *ICAART 2015 - 7th International Conference on Agents and Artificial Intelligence, Proceedings*, vol. 1, pp. 182–189, 01 2015.

[2] J. Delgrande, P. Peppas, and S. Woltran, "General belief revision," *Journal of the ACM*, vol. 65, pp. 1–34, 09 2018.

[3] M.-A. Williams, "Applications of belief revision," vol. 1472, 02 2000.

[4] J. Delgrande, P. Peppas, and S. Woltran, "General belief revision," *Journal of the ACM*, vol. 65, pp. 1–34, 09 2018.

[5] "Sympy library documentation." `https://www.sympy.org/en/index.html`.

[6] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2010.

[7] P. Gärdenfors, *Belief Revision: an introduction.* 05 1992.

[8] S. Dixon and W. Wobcke, "The implementation of a first-order logic agm belief revision system," in *Proceedings of 1993 IEEE Conference on Tools with Al (TAI-93)*, pp. 40–47, 1993.