Techincal University of Denmark

02180 Introduction to Artificial Intelligence

Assignment 2: Belief Revision

Authors

Edrin Molla - s242628
Gisle Garen - s242715
Ignacio Ripoll Gonzalez - s242875
Paulo Ricardo Beckhauser de Araujo - s242779

May 5, 2025

1 Introduction

Belief revision is a fundamental topic in artificial intelligence and knowledge representation, addressing the question of how an intelligent agent should update its beliefs in light of new information. The challenge arises when the new information conflicts with existing beliefs, potentially introducing inconsistency. A rational agent must then decide which beliefs to give up in order to restore consistency, while retaining as much of its original knowledge as possible.

In the formal framework of belief revision, the most widely studied model is the AGM theory, introduced by Alchourrón, Gärdenfors, and Makinson. The AGM model defines three core operations on a belief base: expansion, contraction, and revision. Expansion simply adds a new belief without considering consistency. Contraction removes a belief to eliminate inconsistency, and revision combines both processes by first contracting the negation of the new belief and then expanding the belief base with the new belief.

In this project, we implement a belief revision agent that operates over a belief base consisting of propositional logic formulas. Our approach follows the AGM model and is based on symbolic logic, using an Abstract Syntax Tree (AST) representation for formulas. The belief base stores formulas along with priority values, enabling us to control which beliefs should be retained or removed during contraction. Entailment is checked using a resolution-based algorithm implemented from scratch, without reliance on external logic solvers.

This report describes the design and implementation of each component of the belief revision agent, including the belief base structure, CNF conversion, entailment checker and contraction and expansion mechanism. We conclude with reflections on the development process and potential future improvements.

2 Belief Base

Let $B \subseteq \mathcal{L}_{PL}$ be a finite set of propositional formulas, where $\mathcal{L}_{PL}$ denotes the language of propositional logic. Each formula $\varphi_i \in B$ represents an explicit commitment of the agent, while the implicit commitments are given by the logical closure $\mathrm{Cn}(B)$. In line with the AGM framework, belief change operations (expansion, contraction, and revision) are performed on B to update the agent's informational state.

In our implementation, the belief base is realized as a class `BeliefBase`, which stores a collection of formulas along with integer-valued priorities. Formally, each belief is represented as a pair (φ_i, π_i), where φ_i is a formula and $\pi_i \in \mathbb{Z}$ is its priority. Priorities are used to resolve conflicts during contraction, ensuring more important beliefs are retained when information must be discarded.

All formulas are automatically converted to conjunctive normal form (CNF) upon entry. This normalization step ensures uniform representation and facilitates efficient resolution-based entailment checking. The internal belief list is sorted by priority (in descending order) after every insertion:

```
self.beliefs.sort(key=lambda p: p[1], reverse=True)
```

This sorting ensures that the highest-priority formulas are always considered first during contraction and revision.

As an example, consider the belief base:

$$B = \{(\neg p \vee q, 3),\ (p, 2),\ (r, 1)\},$$

where each tuple represents a formula and its corresponding priority. Internally, these formulas are stored in CNF and prioritized as:

$$[(\neg p \vee q,\ 3),\quad (p,\ 2),\quad (r,\ 1)]$$

Core Operations

- `add`(φ, π) Adds the formula φ with priority π to the belief base. The formula is first converted to CNF to standardize representation.
- `remove`(φ) Removes all instances of the formula φ from the belief base. This is used when a belief must be explicitly retracted.
- `clear`() Empties the entire belief base. This is typically invoked when contraction yields no consistent remainder.
- `get_beliefs`() Returns all formulas in the base, omitting priority metadata.
- `get_prioritized_beliefs`() Returns the full list of (φ_i, π_i) pairs for use in contraction or revision.

Partial Meet Contraction Support

To support partial meet contraction, the belief base provides a method `compute_remainders`(φ), which returns all maximal subsets of B that do not entail φ. These subsets are computed via exhaustive combination and filtering based on entailment checks using the resolution procedure.

Two utility functions complement this process:

- `select_remainders()` ranks remainders by their cumulative priority and selects the maximal ones.
- `intersect_selected()` computes the intersection of the selected remainders to produce the new contracted base.

Together, these mechanisms ensure that belief base updates respect both logical coherence and the agent's epistemic preferences.

3 Logical Entailment

Logical entailment, denoted $B \models \varphi$, expresses whether a formula φ logically follows from a belief base B. To determine entailment, we implemented a resolution-based procedure. This choice is motivated by the completeness and soundness of the resolution method for propositional logic in conjunctive normal form (CNF).

Resolution operates on clauses, which are disjunctions of literals. A literal is a propositional variable p or its negation $\neg p$. Clauses are represented as sets of such literals, and formulas are converted into CNF—conjunctions of clauses—before applying resolution. The entailment check follows the principle of *reductio ad absurdum*:

$$B \models \varphi \quad \text{iff} \quad B \cup \{\neg\varphi\} \models \bot$$

That is, if adding the negation of φ to the belief base leads to a contradiction (an empty clause via resolution), then φ is entailed.

Each formula in the belief base is converted to CNF using the `to_cnf()` method. For example, the formula

$$(p \to q) \land p$$

is transformed into the clauses

$$\{\neg p, q\}, \quad \{p\}$$

which are represented programmatically as sets of tuples:

```
frozenset({("p", False), ("q", True)}),  frozenset({("p", True)})
```

This transformation is handled by the function `extract_clauses()`, which parses CNF formulas into clause sets, ensuring each literal is either an atomic symbol or its negation.

Once the clauses from the belief base are extracted, the formula φ is negated and its CNF clauses are also added. The main resolution procedure, implemented in `resolution_entails()`, repeatedly applies binary resolution between pairs of clauses. Given two clauses C_1 and C_2 that contain complementary literals ℓ and $\neg\ell$, the resolvent is:

$$R = (C_1 \cup C_2) \setminus \{\ell, \neg\ell\}$$

This process is repeated until either:

- The empty clause $\square$ is derived, in which case $B \models \varphi$, or
- No new clauses are generated, indicating $B \not\models \varphi$.

A key subtlety in the implementation is filtering out tautological clauses—those that contain both a literal and its negation. Such clauses are always true and would otherwise interfere with the resolution process by producing misleading contradictions. To address this, we implemented a function `is_tautology()` that removes such clauses from the resolution process.

The implementation adheres to the theoretical properties of the resolution calculus. We verified it through synthetic tests, ensuring it respects entailment for both positive and negative cases. For instance, given:

$$B = \{p \to q, p\}, \quad \varphi = q$$

our resolution procedure correctly concludes $B \models q$.

The method is complete and sound for propositional logic and forms the foundation for verifying AGM postulates such as consistency and vacuity.

4 Contraction

The contraction operation is responsible for removing a belief φ from a belief base B in a rational way, such that the resulting belief base no longer entails φ while preserving as much information as possible. Our implementation follows the partial meet contraction strategy from AGM theory.

The contraction process first checks whether φ is entailed by B. If it is not, no contraction is needed. If it is, we compute the set of all maximal subsets of B that do not entail φ, known as the remainders. These subsets represent belief bases that no longer support φ.

Each belief in B is associated with a priority. We select from the remainders those subsets whose total priority is maximized. If multiple remainders achieve this maximum score, we compute the intersection of their indices. The new belief base consists only of those formulas appearing in this intersection.

If no such remainders can be found (e.g., the base only consists of tautologies or all entail φ), the entire belief base is cleared.

Formally, the algorithm satisfies the Success, Inclusion, Vacuity, Consistency and Extensionality postulates of AGM theory. The use of priorities adds a heuristic dimension, preferring to retain more valuable beliefs.

5 Expansion

In the AGM framework, expansion is the operation of incorporating a new formula φ into a belief base B without removing any existing beliefs, resulting in $B + \varphi$. Unlike contraction and revision, expansion imposes no requirement of consistency preservation. Its only postulate is Success, which requires that φ must belong to the expanded belief base:

$$\varphi \in B + \varphi.$$

Our implementation follows this principle precisely. The method `expand()` appends the formula φ to the current belief base along with a user-defined priority value. This priority is later used in contraction to guide the selection among competing beliefs, particularly during partial meet contraction.

Importantly, expansion does not check for logical entailment or consistency before incorporating the new belief. Thus, it is possible that $B + \varphi$ becomes logically inconsistent,

i.e., $B \cup \{\varphi\} \models \bot$. This behavior aligns with the AGM definition of expansion, which allows for inconsistency, distinguishing it from belief revision.

Mathematically, if the original belief base is $B = \{\varphi_1, \ldots, \varphi_n\}$, then the result of expansion is simply:

$$B + \varphi = B \cup \{\varphi\}.$$

This operation plays a vital role in the Levi Identity for belief revision:

$$B * \varphi = (B - \neg\varphi) + \varphi,$$

which defines revision in terms of contraction followed by expansion. In our agent, expansion serves as the final step in revision, ensuring that the new information φ is present in the updated belief base.

While our current implementation adopts a minimal approach, future refinements could include optional consistency checks or even automatic invocation of contraction when inconsistency is detected. Nevertheless, the current design preserves the theoretical purity of AGM-style expansion and provides a clear semantic boundary between belief addition and belief reconciliation.

6 Learnings

This project has provided a rich opportunity to explore the theoretical foundations and practical implementation of belief revision. Engaging directly with formal logical systems and the AGM postulates offered insights that extended beyond textbook understanding. By constructing an agent capable of expansion and contraction in accordance with formal criteria, we were able to observe how delicate and interdependent these belief operations are in practice.

A key learning outcome was the recognition of how logical entailment is not merely a syntactic issue but one that depends heavily on proper canonical transformations—specifically, conversion into conjunctive normal form. This process exposed a variety of edge cases that tested our understanding of formula manipulation and resolution.

Implementing resolution from scratch sharpened our appreciation for both its elegance and its limitations. Prioritization proved to be a critical tool for structuring belief importance in a rational framework, since its integration into contraction procedures helped balance logical necessity with practical goals.

We also gained valuable experience debugging discrepancies between logical theory and program behavior, underscoring the interpretative bridge developers must build between semantics and implementation. Overall, the project deepened our understanding of formal logic, computational reasoning, and the challenges of aligning algorithmic processes with rational agent models.

7 Conclusion

This project has demonstrated the implementation of a belief revision agent rooted in the AGM framework and propositional logic. Through the integration of resolution-based entailment checking, partial meet contraction, and prioritized expansion, we were able to construct a system that not only adheres to the core AGM postulates but also captures key practical considerations such as belief importance and computational tractability.

The agent reliably manages belief dynamics in a logically sound manner, respecting consistency while maintaining expressivity. Our approach emphasizes modularity and interpretability, and the resulting system allows for the controlled addition and removal of information within a rational agent framework.

Looking ahead, promising extensions include implementing full revision as defined by the Levi identity, incorporating plausibility orderings over possible worlds for a semantic perspective on belief change, and enhancing the resolution engine to scale with more complex knowledge bases. Another intriguing direction is the application of this agent in structured reasoning tasks such as strategic gameplay, for instance in a variant of Mastermind, where iterative feedback and revision mirror real-world decision cycles.

Overall, the project strengthened our understanding of formal logic, automated reasoning, and the nuanced requirements of knowledge management in intelligent systems.