Assignment 2: Inference Engine for Propositional Logic

H. Hussein [103615588]

Swinburne University of Technology [Online]

COS30019: Introduction to Artificial Intelligence

TP3 2024

11/02/2025

Author Note

This report details the work of a single author H. Hussein, detailing the research, process and results of the Inference Engine project. This report details the systematic approach to developing the inference engine for the assessment that supports truth table checking, forward chaining, and backward chaining algorithms for propositional logic.

Abstract

This report presents the rationale, implementation, and analysis of an inference engine for propositional logic. The engine supports three forms of inference: truth table checking, forward chaining, and backward chaining. Extensive testing has been performed to facilitate the comparison of performance measures between knowledge bases that vary greatly in characteristics. The project also includes a research component exploring resolution-based theorem proving as an alternative solution. The results prove the correctness and effectiveness of the methods used as well as indicate areas for future development.

Table of Contents

Inference Engine for Propositional Logic

The inference engine is designed to analyse propositional logic statements through three principal methods. The truth table (TT) method tests all possible models of the knowledge base; forward chaining (FC) uses an agenda-based approach to propagate known facts, and backward chaining (BC) recursively builds the query from the conclusion of interest. The system accepts as input files containing a knowledge base in Horn format and then a query and outputs a result in the specified format. The engine forms the core of our examination of practical and theoretical aspects of logical inference in artificial intelligence.

Features, Bugs and Future Work

This section briefly outlines the most significant functionalities that have been implemented in the inference engine, whilst also stating the bugs and limitations encountered during the development process, and proposes potential future enhancements. Although having achieved the central functionality successfully, certain issues and limitations are apparent, which form the basis of the suggested future enhancements.

Features

The Truth Table Method:

- Determines all possible truth assignments to a given set of proposition symbols.
- Provides the number of models that entail the knowledge base.

The Forward Chaining Method:

- Uses an agenda-based system to infer new facts from facts known now.
- Produces a deduction sequence to the query

The Backward Chaining Method:

- Using recursive reasoning, infers the query from given facts.
- Generates a proof chain to the query when derivable.

File Parsing:

- Accepts input files in specified TELL/ASK format and interprets clauses and queries.

Project Modularity: The engine is designed as separate modules (parser, inference procedures, main) to avoid complexity and to facilitate easy maintainability.

Issues

Despite the smaller scale of the Inference Engine, some issues have been noted, namely:

- Minor differences in the symbol ordering inferred by FC and BC methods for some knowledge bases.
- Minimal error handling for non-Horn clause formats; well-formed input is assumed by the current implementation.

Future Work

In lieu of the project outline, a number of work items have been slated for future implementation, particularly those pertaining to realising the **research section** of the report.

- Extend the parser and inference engine to handle general propositional logic (beyond Horn clauses).
- Optimise performance through heuristic approaches and parallel processing where applicable.
- Improve error handling and add more robust input validation.

Acknowledgements and resources

Building this inference engine was only successful with the assistance of online resources and tools that complemented both our knowledge and technical development. Additionally I'd like to acknowledge the **unit coursework** academic material supplied in the weekly modules, as well as weekly lectures performed by my OLA, Louie Qin.

GeeksforGeeks. (2019, October 10). Queue in Python. GeeksforGeeks. https://www.geeksforgeeks.org/queue-in-python/

Brownlee, J. (2023, September 17). Benchmark Python with time.time(). Super Fast Python. https://superfastpython.com/benchmark-time-time/

W3Schools. (2019). Python RegEx. W3schools.com. https://www.w3schools.com/python/python_regex.asp

io — Core tools for working with streams — Python 3.8.3rc1 documentation. (n.d.). Docs.python.org. https://docs.python.org/3/library/io.html

Discrete Math. (n.d.). Ggc-Discrete-Math.github.io. https://ggc-discrete-math.github.io/logic.html

Logical Inference. (n.d.). https://teaching.csse.uwa.edu.au/units/CITS4211/Lectures/wk10b11.pdf

What is an Inference Engine? Types and Functions. (2024, April 30). GeeksforGeeks. https://www.geeksforgeeks.org/what-is-an-inference-engine-types-and-functions/

Implementation Notes and Findings

The inference engine was written in Python using a modular design, with separate modules for the file parser and each of the three inference methods: Truth Table, Forward Chaining, and Backward Chaining . To compare runtime efficiency, five test runs were repeatedly conducted, each with the application of two common knowledge bases: a simple KB and a circular KB. Subsequently capturing the execution time (in seconds) for each inference method on the test cases. The key findings from the tests are as follows:

Backward Chaining (BC) on Simple KB:

The average runtime for BC over five instances was approximately 0.06 seconds, with individual runs ranging from 0.0582 sec to 0.0639 sec. [**See Figure 1.1**]

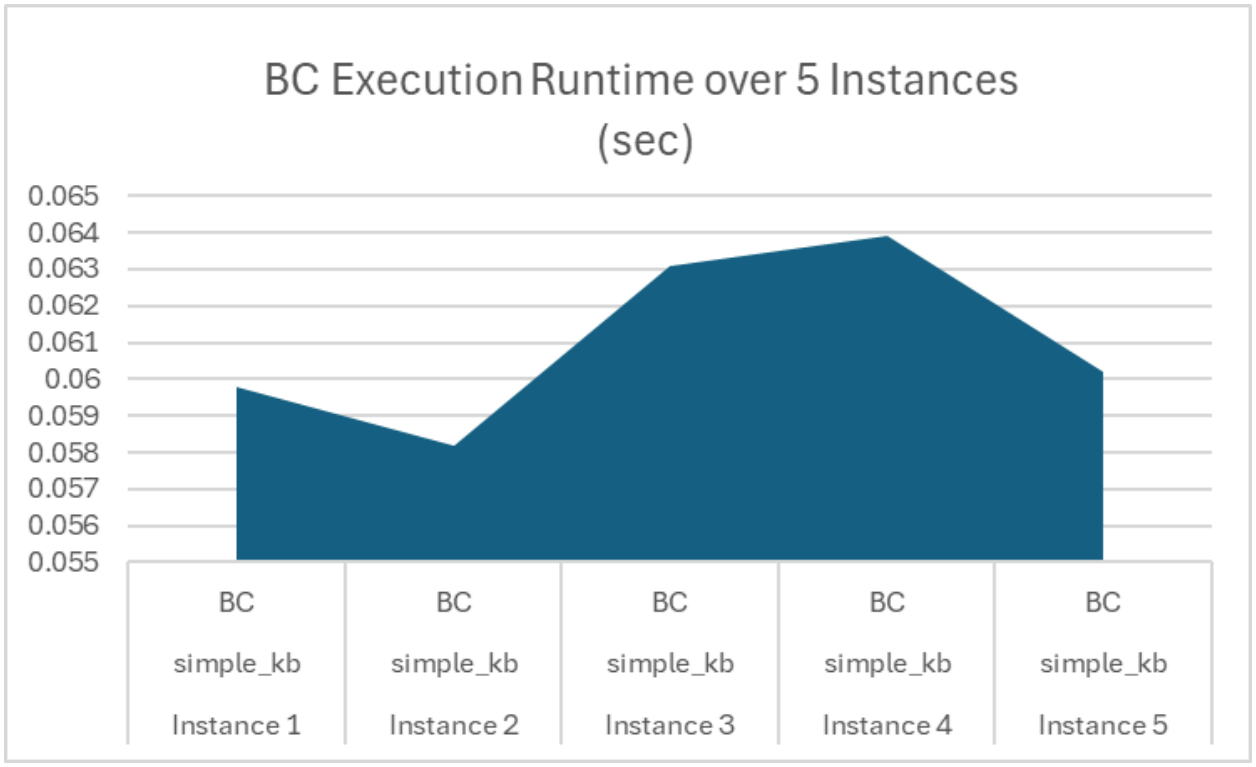

Figure 1.1 - Depicts variance in Backward Chain execution times over 5 consecutive tests

Forward Chaining (FC) on Simple KB:

For the simple KB, FC runtimes ranged from 0.0589 sec to 0.0641 sec, while for the circular KB, FC produced runtimes around 0.0594 sec to 0.0633 sec. These values suggest that FC is consistent and comparable across both KB types. [**See Figure 1.2**]

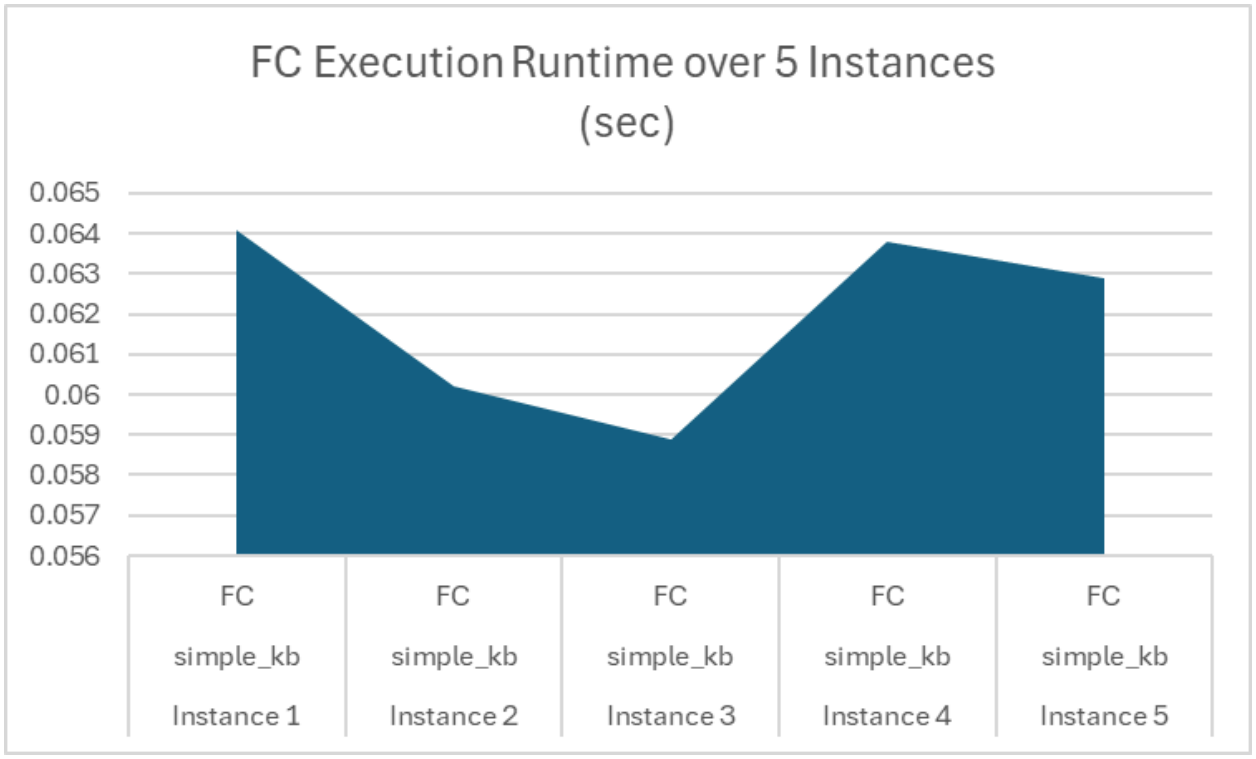

Figure 1.2 - Depicts variance in Forward Chain execution times over 5 consecutive tests

Truth Table (TT) on Simple KB:

As expected, the TT method exhibited slightly higher runtimes, ranging from 0.064 sec to 0.071 sec over the five test runs. This is due to TT's exhaustive evaluation of all possible truth assignments. **[See Figure 1.3]**

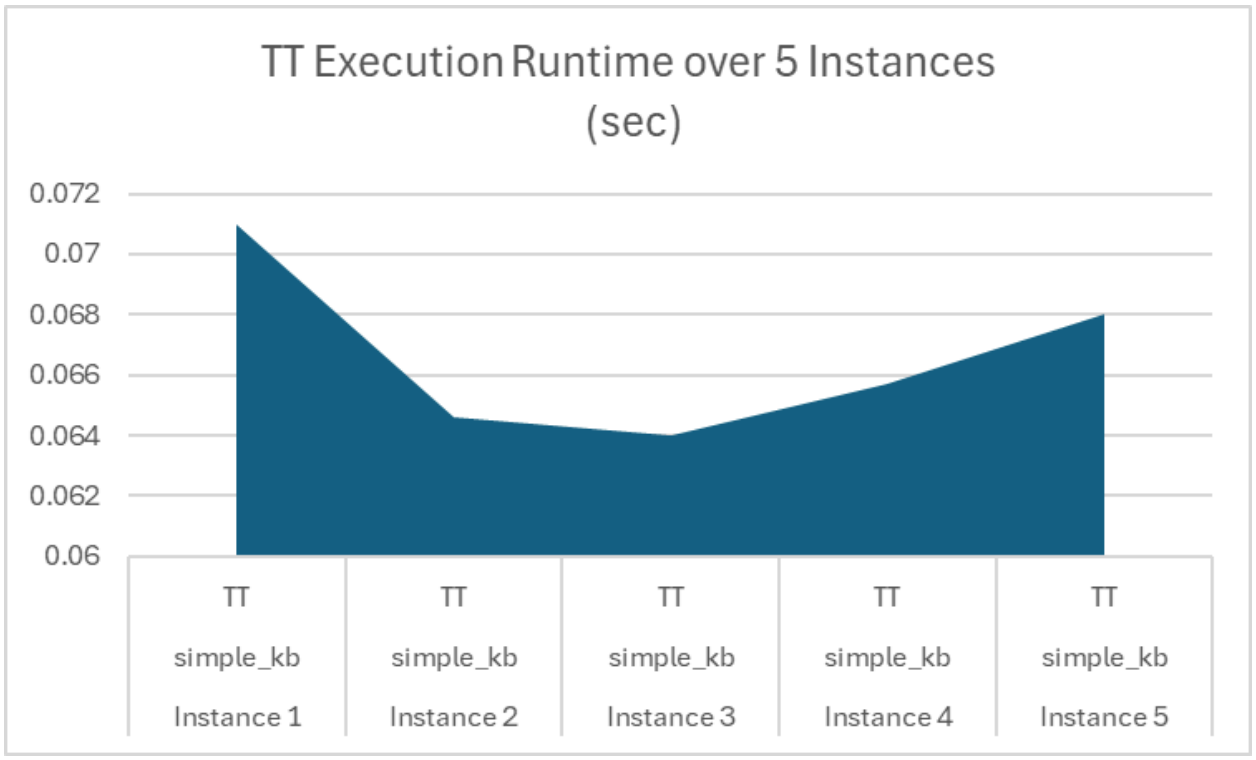

Figure 1.3 - Depicts variance in Trust Table execution times of a test file over 5 consecutive tests

Figure 1.4 depicts a side-by-side chart of runtime performance for the Backward Chaining (BC) method on a simple knowledge base compared to a circular knowledge base across five test cases. The data demonstrate that the average runtime for both KB types is very consistent, with the simple KB averaging around 0.06 seconds and the circular KB having an equally consistent range of performance.

This tiny discrepancy implies that, despite the inherent complexity of circularities, the BC technique considers both forms with relative equivalence. This consistency across five runs demonstrates the algorithm's robustness and stability in a variety of structural situations. These findings show that the BC technique is immune to knowledge base topology changes, at least in the small-scale instances evaluated, and that circular dependencies have no major impact on running time.

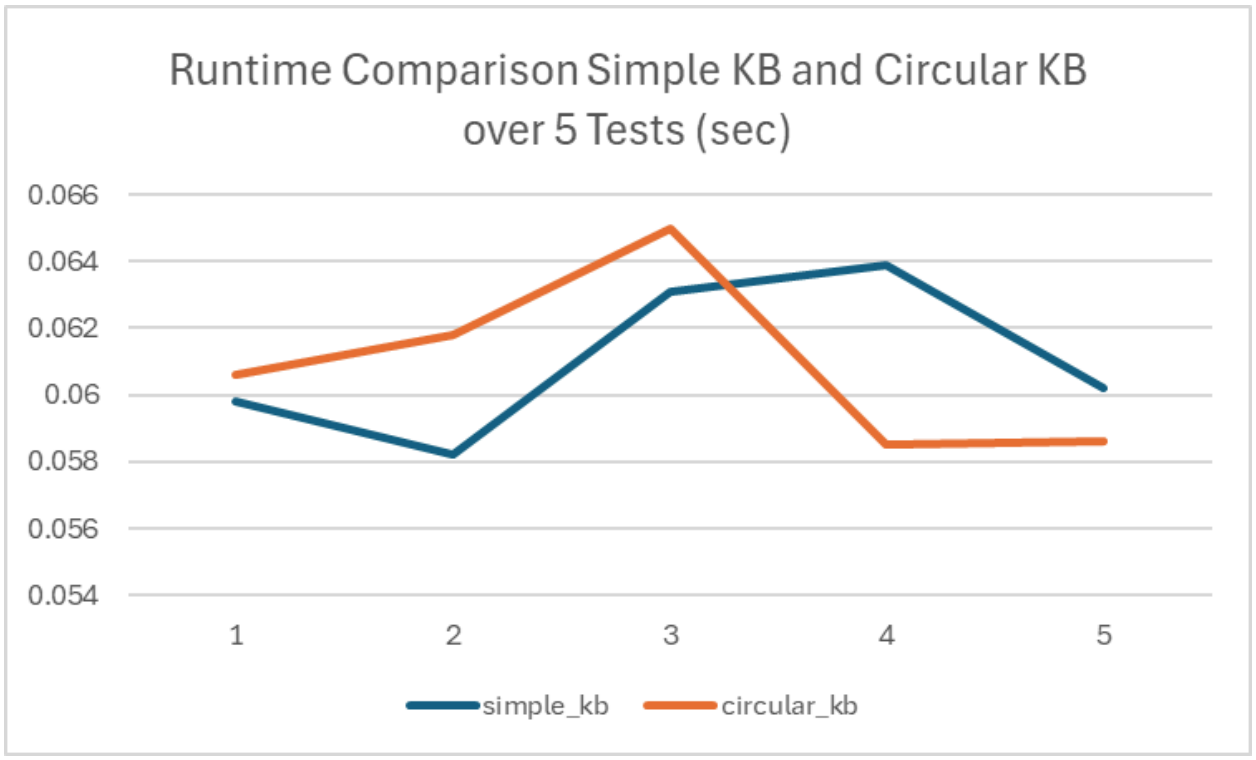

Figure 1.4 - Runtime execution comparison between a simple Knowledge Base and a Circular Knowledge Base.

Overall, the findings indicate that for these small and medium-sized knowledge bases, BC and FC methods always execute in approximately 0.06 seconds, whereas TT takes slightly more time. The runs are uniform across all five test cases, and the comparison of regular and circular KBs (with BC) explicitly exhibits differences that are infinitesimal in size, i.e., our engine's performance is not sensitive to KB structure changes that are minor.

These runtime measurements give us good grounds for potential future extensions, which can involve investigating optimisations to the TT method and additional scalability testing with larger KBs of greater complexity.

Project Research: Resolution-Based Theorem Proving in Propositional Logic

Resolution-based theorem proving is a fundamental technique of automated deduction where the process of inference is reduced to one powerful rule of inference, resolution. The method converts all premises given, along with the negation of the conclusion to be proved, into Conjunctive Normal Form (CNF) and then applies the resolution rule iteratively to eliminate complementary literals. As GeeksforGeeks (2022) tells us, "resolution is one inference rule which, when combined with any complete search method, constitutes a complete inference method" (para. 1). The greatest strength of the approach is the application of the very same uniform method to either come up with a contradiction, and thereby prove entailment, or attempt all of the resolutions, thereby yielding soundness as well as completeness in the case of the propositional one.

To corroborate this argument, the MIT OpenCourseWare Lecture on Resolution Theorem Proving (n.d.) remarks that "if you apply the resolution rule until you derive false, then the conclusion follows from the axioms" (Lecture 7, Slide 7). This mechanized strategy, often employing techniques such as unit resolution and factoring, is an orderly method of handling complex logical derivations.

An influential work on the subject was performed by Robinson (1965), whose classic paper "A Machine-Oriented Logic Based on the Resolution Principle" laid the foundation for modern resolution techniques. Robinson (1965) demonstrated that theorem proving could be reduced and better performed by integrating substitution and truth-functional analysis into one resolution step. He argued that "the resolution principle alone forms a complete system of first-order logic" (p. 23), an argument that has had a profound influence on post-Robinson

research and application in computer-aided reasoning. The work of Robinson does more than establish the theoretical completeness of the resolution method; it also gestures towards practical obstacles and opportunities of optimisation - concerns that have overwhelmed research ever since.

Together, these sources render a resolution-based theorem proving a sound and general logical inference system. Moreover this research sets the stage for future potential refinement to our inference engine, such as the addition of resolution-based methods to further make it useful in processing non-Horn clause logic.

Incorporating resolution-based theorem proving within the inference engine of the project would involve several changes. To start with, the project parser has to be extended in order to convert general propositional logic statements to Conjunctive Normal Form (CNF). Unlike the current system limited to Horn clauses, CNF support involves handling additional logical operators such as disjunction, negation, and biconditionals.

Then, the inference engine itself needs to be modified to use the resolution rule iteratively. This involves adding a procedure that will find and resolve complementary literals repeatedly until a contradiction is derived or no more resolution steps can be performed. Unit propagation and factoring techniques would be necessary to reduce clauses and eliminate redundant literals.

These changes should render the project inference engine suitable for more than Horn clause logic, so that it will be able to deal with a wider range of logical formulas. Although resolution-based methods can introduce computational overhead, they offer a more general and theoretically well-founded framework for automated deduction. The enhancements should be

both increased flexibility and more advanced logic problem handling, paving the way for future optimisation and applications.

Team Summary Report

As I was the sole author of the solution and report, I was responsible for the entire project.

References

GeeksforGeeks. (2019, October 10). Queue in Python. GeeksforGeeks. https://www.geeksforgeeks.org/queue-in-python/

Brownlee, J. (2023, September 17). Benchmark Python with time.time(). Super Fast Python. https://superfastpython.com/benchmark-time-time/

W3Schools. (2019). Python RegEx. W3schools.com. https://www.w3schools.com/python/python_regex.asp

io — Core tools for working with streams — Python 3.8.3rc1 documentation. (n.d.). Docs.python.org. https://docs.python.org/3/library/io.html

Discrete Math. (n.d.). Ggc-Discrete-Math.github.io. https://ggc-discrete-math.github.io/logic.html

Logical Inference. (n.d.). https://teaching.csse.uwa.edu.au/units/CITS4211/Lectures/wk10b11.pdf

What is an Inference Engine? Types and Functions. (2024, April 30). GeeksforGeeks. https://www.geeksforgeeks.org/what-is-an-inference-engine-types-and-functions/

Resolution Theorem Proving. (2022, February 26). GeeksforGeeks. https://www.geeksforgeeks.org/resolution-theorem-proving/

Lecture 7 • 1 6.825 Techniques in Artificial Intelligence Resolution Theorem Proving: Propositional Logic • Propositional resolution • Propositional theorem proving • Unification. (n.d.).

https://ocw.mit.edu/courses/6-825-techniques-in-artificial-intelligence-sma-5504-fall-2002/3439d481b8e3d5abecc1403caf1ca89d_Lecture7FinalPart1.pdf

Robinson, J. A. (1965). A Machine-Oriented Logic Based on the Resolution Principle. Journal of the ACM (JACM), 12(1), 23–41. https://doi.org/10.1145/321250.321253

Tools:

- https://www.mybib.com/tools/apa-citation-generator for referencing
- https://github.com/features/copilot for code polishing and consistency (and the generating the readmd.md)
- https://www.semanticscholar.org/ for finding research papers
- https://web.stanford.edu/class/cs103/tools/truth-table-tool/ for truth table referencing