Propositional Logic Inference Engine
Assignment 2 Report

Vu Minh Dung
s104775535

Trinh Khoi Nguyen
s104993405

GROUP 5

November 20th, 2024

—

COS30049: Introduction to AI

—

Dr. Aiden Nguyen

TABLE OF CONTENTS

1. Instructions

1.1 Program Execution

The inference engine can be run using the following command-line syntax:

```
python iengine.py <filename> <method>
```

Where:

- `<filename>` is the input text file containing the knowledge base and query
- `<method>` can be one of three algorithms:
  - TT: Truth Table Checking
  - FC: Forward Chaining
  - BC: Backward Chaining

Example Usages:

```
PS D:\Intro to AI\Assignment_2> python iengine.py test1.txt TT
YES: 3
PS D:\Intro to AI\Assignment_2> python iengine.py test1.txt FC
YES: a, b, p2, p3, p1, d, c, e, f
PS D:\Intro to AI\Assignment_2> python iengine.py test1.txt BC
YES: p2, p3, p1, d
```

1.2 Input File Format

The input file follows a specific structure:

- TELL section: Horn clauses separated by semicolons
- ASK section: A single proposition symbol to query

Example Input (test1.txt):

test1.txt

```
1  TELL
2  p2 => p3; p3 => p1; c => e; b & e => f; f & g => h; p2 & p1 & p3 => d; p1 & p3 => c; a; b; p2;
3  ASK
4  d
```

2. Introduction to Propositional Logic Inference Engine

2.1 Propositional Logic Fundamentals

Propositional logic is a formal system for reasoning about logical statements using logical connectives. It provides a mathematical framework for representing and analyzing logical relationships between propositions.

Knowledge Base Management

- Robust sentence parsing with support for multiple expressions separated by semicolons
- Symbol extraction and normalization
- Support for various logical operator notations (&&, ||, →, ↔, etc.)
- Automatic parentheses balancing and validation

2.2 Logical Connectives

Our implementation supports the following logical connectives:

- `~`: Negation (¬)
- `&`: Conjunction (∧)
- `||`: Disjunction (∨)
- `=>`: Implication (⇒)
- `<=>`: Biconditional (⇔)

```python
def _normalize_expression(self, expr: str) -> str:
    """Normalize logical expressions"""
    # Replace alternative symbols with standard ones
    replacements = {
        '!': '~',    # NOT
        '&&': '&',   # AND
        '||': '||',  # OR
        '→': '=>',   # IMPLIES
        '↔': '<=>',  # BICONDITIONAL
        'AND': '&',
        'OR': '||',
        'NOT': '~',
        'IMPLIES': '=>',
        'IFF': '<=>'
    }
```

Expression Handling

- Comprehensive logical expression parser
- Support for nested parentheses
- Clean handling of whitespace and formatting variations

2.3 Core Concepts

- Proposition: A declarative statement that is either true or false
- Knowledge Base: A set of logical sentences representing known information
- Entailment: A logical relationship where one proposition logically follows from another
- Model: A possible interpretation that makes a set of sentences true

3. Inference Methods

3.1 Truth Table Checking (TT)

Description:

- Checks entailment by examining all possible truth value combinations
- Works with all types of knowledge bases
- Comprehensive but computationally expensive

Algorithm Steps:

1. Generate all possible models (truth assignments)
2. Check if the knowledge base is true in each model
3. Verify if the query is true in all models where the KB is true

Complexity: O(2^n), where n is the number of unique symbols

3.2 Forward Chaining (FC)

Description:

- Works with Horn-form knowledge bases
- Starts with known facts and derives new facts
- Incrementally builds knowledge

Algorithm Steps:

1. Initialize agenda with known facts
2. Repeatedly apply inference rules
3. Add newly inferred facts to the knowledge base
4. Stop when query is found or no new facts can be derived

Complexity: O(n*m), where n is the number of rules and m is the number of facts

3.3 Backward Chaining (BC)

Description:

- Works with Horn-form knowledge bases
- Starts with the goal and works backward
- Attempts to prove the query by finding supporting facts

Algorithm Steps:

1. Start with the goal query
2. Find rules that can prove the goal
3. Recursively prove the prerequisites of those rules
4. Terminate when all prerequisites are satisfied or impossible

Complexity: O(n*m), where n is the number of rules and m is the recursion depth

4. Implementation Details

4.1 System Architecture

Include a **class diagram** showing the relationships among the core components:

- "KnowledgeBase" class: Stores and manage sentences and symbols.
- "LogicalExpression" class: Handles parsing of logical statements.
- "InferenceEngine" class: Implements the TT, FC, and BC algorithms.
- Mention dependencies between parsing, model evaluation, and inference.

4.2 Key Implementation Challenges

- Parsing complex logical expressions
- Efficient model generation
- Handling different knowledge base formats
- Implementing recursive inference algorithms

4.3 Algorithm-Specific Implementations

Expand on:

- **Truth Table Checking (TT):**
  - Explain the truth table generation using "itertools.product" for all combinations.
  - Clarify how entailment is verified by comparing truth assignments for KB and the query.
- **Forward Chaining (FC):**
  - Discuss the agenda-based mechanism and iterative derivation of facts.
  - Highlight how Horn clauses are checked and facts added to the inferred set.
- **Backward Chaining (BC):**
  - Describe the recursive approach to query goals and sub-goals.
  - Mention the use of a visited set to prevent circular reasoning.

Key Implementation Differences

1. **Memory Usage**
   - TT: Highest memory usage due to storing all possible models
   - FC: Moderate memory usage for agenda and count tracking
   - BC: Lowest memory usage due to recursive approach
2. **Completeness**
   - TT: Complete for all propositional logic
   - FC/BC: Complete only for Horn clauses
3. **Performance Characteristics**
   - TT: Exponential with number of symbols
   - FC: Linear with KB size
   - BC: Can be exponential in worst case but often better in practice

5. Testing Strategy

5.1 Test Case Categories

1. Simple single-rule scenarios
2. Multiple interconnected rules
3. Circular reasoning cases
4. Edge cases with complex logical structures
5. Performance benchmark cases

5.2 Test Case Examples

Generic_1.txt

```
TELL
(a <=> (c => ~d)) & b & (b => a); c; ~f || g;
ASK
~d & (~g => ~f)
```

Generic_2.txt

```
TELL
(p || e) => r; (r || s) => t; u & g;
ASK
(r & ~t) & (u & g)
```

Generic_3.txt

```
TELL
(p || q) & ~r; s <=> (p & q); t || ~s; ~p; t;
ASK
q
```

Generic_4.txt

```
TELL
(a <=> (c => d)) =>e; c; ~f||g; a; c; a=>d;
ASK
~e
```

Generic_5.txt

```
TELL
a & b => c;c & d => e;e & f => g;g => h;i & j => k;k & l => m;m & n => o;o & p => q;q => r;s & t => u;u & v => w;w & x => y;y => z;a; b;
ASK
z
```

Horn_1.txt

```
TELL
rainy => wet; wet => umbrella; cold & wet => stay_inside; windy & rainy => storm; cold; windy;
ASK
stay_inside
```

Horn_2.txt

```
TELL
p2=> p3; p3 => p1; c => e; b&e => f; f&g => h; p1=>d; p1&p3 => c; a; b; p2;
ASK
d
```

Horn_3.txt

```
TELL
i => k; k => j; f & g => h; i & j & k => d; j & k => c; a; b; i;
ASK
d
```

Horn_4.txt

```
TELL
a =>d; g => c; b & b => g; c; f; g;
ASK
d
```

Horn_5.txt

```
TELL
a => b; b =>c; c => d; d;
ASK
d
```

Test case outputs:

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_1.txt TT
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_1.txt FC
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_1.txt BC
NO
```

File Generic_1.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_2.txt TT
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_2.txt FC
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_2.txt BC
NO
```

File Generic_2.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_3.txt TT
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_3.txt FC
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_3.txt BC
NO
```

File Generic_3.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_4.txt TT
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_4.txt FC
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_4.txt BC
NO
```

File Generic_4.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_5.txt FC
YES: a, b, d, f, i, j, l, n, p, s, t, v, x, c, k, u, e, m, w, g, o, y, h, q, z, r
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_5.txt BC
YES: s, t, u, v, w, x, y, z
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Generic_5.txt TT
```

File Generic_5.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_1.txt TT
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_1.txt FC
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_1.txt BC
NO
```

File Horn_1.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_2.txt TT
YES: 3
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_2.txt FC
YES: a, b, p2, p3, p1, d, c, e, f
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_2.txt BC
YES: p2, p3, p1, d
```

File Horn_2.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_3.txt TT
YES: 3
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_3.txt FC
YES: a, b, i, k, j, d, c
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_3.txt BC
YES: i, k, j, d
```

File Horn_3.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_4.txt TT
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_4.txt FC
NO
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_4.txt BC
NO
```

File Horn_4.txt: Provided test case

```
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_5.txt TT
YES: 4
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_5.txt FC
YES: d
PS D:\Intro to AI\Assignment_2> python iengine.py Testing/Horn_5.txt BC
YES: d
```

File Horn_5.txt: Provided test case

6. Feature and Limitations

6.1 Features Implemented

Core Functionality

1. Knowledge Base Management
    - Robust sentence parsing with support for multiple expressions separated by semicolons
    - Symbol extraction and normalization
    - Support for various logical operator notations (&&, ||, →, ↔, etc.)
    - Automatic parentheses balancing and validation
2. Inference Methods
    - Truth Table (TT) enumeration
    - Forward Chaining (FC)
    - Backward Chaining (BC)
3. Logical Operators Support
    - NOT (~)
    - AND (&)
    - OR (||)
    - IMPLICATION (=>)
    - BICONDITIONAL (<=>)
4. Expression Handling
    - Comprehensive logical expression parser
    - Support for nested parentheses
    - Clean handling of whitespace and formatting variations

Additional Features

1. Error Handling
    - Robust error detection for malformed expressions
    - Graceful handling of unbalanced parentheses
    - Detection of unknown symbols
2. Input Processing
    - Support for TELL/ASK format
    - Flexible input parsing with support for various formats
    - Clean handling of empty lines and whitespace

6.2 Known limitations

1. Expression Parsing Edge Cases
    - Complex nested expressions with multiple biconditionals (<=>) might not parse correctly in some rare cases
    - Some deeply nested parentheses combinations might cause incorrect operator precedence
2. Symbol Extraction
    - The symbol extraction might include some operators if they contain alphabetic characters in certain edge cases

Missing Features

1. No optimization for truth table enumeration (implements basic exhaustive search)
2. No visualization of the inference process
3. No support for custom operator precedence rules
4. No support for quantifiers (∀, ∃) as this is focused on propositional logic only

7. Research Initiatives

7.1 Potential Improvements

- Implementing a generic theorem prover
- Supporting more complex knowledge bases
- Optimization techniques for inference algorithms

7.2 Performance Analysis

Algorithm	Strengths	Weaknesses	Time Complexity
Truth Table (TT)	- Works with general knowledge bases - Comprehensive and exact	- Computationally expensive - Exponential growth with the number of symbols	O(2^n), where n is the number of symbols
Forward Chaining (FC)	- Efficient for Horn-form KB - Incremental and straightforward derivation of facts	- Limited to Horn-form KB - Fails on non-Horn clauses	O(n·m), where n is the number of rules, mmm is the number of known facts
Backward Chaining (BC)	- Efficient for proving specific queries - Avoids generating unnecessary facts	- Susceptible to infinite loops - Recursive approach can be memory-intensive	O(n·d), where d is the recursion depth

8. Team Collaboration

8.1 Individual Contributions

Vu Minh Dung:

- Testing and validation
- Algorithm research
- Implementation of inference methods
- Performance analysis
- Report writing

Trinh Khoi Nguyen:

- Testing and validation
- Algorithm research
- Implementation of inference methods
- Performance analysis
- Report documentation

8.2 Collaboration Process

- Regular meetings
- Shared code repository
- Peer code reviews
- Collaborative problem-solving

Percentage Contribution:

- Vu Minh Dung: 50%
- Trinh Khoi Nguyen: 50%

9. Conclusion

The propositional logic inference engine demonstrates the power of logical reasoning in artificial intelligence. While each inference method has its strengths, the choice depends on the specific knowledge base and computational constraints.

Future Recommendations:

- Implement more advanced theorem-proving techniques
- Optimize algorithmic efficiency
- Extend support for first-order logic

10. References

Russell, S. J., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd Edition). Pearson.

Nilsson, N. J. (1998). Artificial Intelligence: A New Synthesis. Morgan Kaufmann.

Stanford Encyclopedia of Philosophy. (n.d.). Propositional Logic. Retrieved from https://plato.stanford.edu/entries/logic-propositional/

11. Acknowledgements

Special thanks to our course instructors and teaching assistants for guidance throughout this project.