Project report on

Belief Revision

for 02180 Introduction to Artificial Intelligence

Contributors: Jannis Haberhausen (s186398)
Jack Reinhardt (s186182)
Kilian Speiser (s181993)
Jacob Miller (s186093)

Contents

1 Introduction

2 Design and Implementation

The design of the belief base takes advantage of the object-oriented nature of the Python programming language. At the highest level, the contents of the belief base are stored in an instance of a class called BeliefBase. The BeliefBase class simply contains a list of beliefs along with methods for adding or removing beliefs, checking for logical entailment, performing partial meet contraction, and showing the current belief base. One level lower, the beliefs are represented as an instance of the class Belief. Each belief contains a list of clauses, which is initialized when the object is created, taking an input string in CNF with an option for negating the entire belief. The Belief class also contains methods to print the belief or to convert the belief to a string. At the lowest level is the Clause class, which stores a list of the positive propositional symbols and a list of the negative propositional symbols. These two lists are initialized when each Clause is created by parsing the string that is input. The methods in Clause include deleting symbols, checking if the clause is empty and printing or converting the clause to a string. The Clause class also implements static methods to combine clauses, create a copy of a clause, check if two clauses are equal, negate a clause, and resolve two clauses.

2.1 Implementation - Converting String into Beliefs and Clauses

2.2 Assumptions

The design and implementation of the belief base makes two important assumptions: (1) that the string representing a belief is already in CNF and (2) that each propositional logic symbol is a single character. Assumption one makes it possible to directly apply the resolution algorithm to the clauses in the belief. Assumption two was a design choice for robustly implementing the parsing of string and converting the strings into beliefs and clauses.

3 Logical Entailment

A resolution-based approach was used to check for logical entailment in the belief base. Once a belief base has been initialized and beliefs have been added to it, logical entailment can be checked of an input belief in string form. First, the input string is converted into its negation as a belief object as described in section 3.2. Then, all of the clauses in the belief base and the clauses of the input belief are stored in a single list to iterate over. The resolution algorithm loops over pairs of clauses, resolves these clauses, and then checks if the resolvent is either empty or already in the list of clauses. The implementation of the resolution algorithms is further explained in section 3.1. If two clauses resolve to an empty clause, then the method returns true, indicating that the input belief is entailed by the belief base. Otherwise, if the list of clauses has been exhausted and no further clauses can be added through resolution, then the loop terminates and the method returns false, indicating that the input belief is not entailed by the belief base.

A list containing all pairs of clauses that have already been resolved was added to improve the efficiency of the resolution algorithm. Each time the algorithm iterates over a pair of clauses, it checks that the clauses are not equal (not the same clause) and that the pair of clauses is not in the list of previously resolved clauses.

3.1 Resolution

3.2 Belief Negation

4 Revision

4.1 Partial Meet Contraction