Minimization

Grao en Intelixencia Artificial, Lógica

April 2025

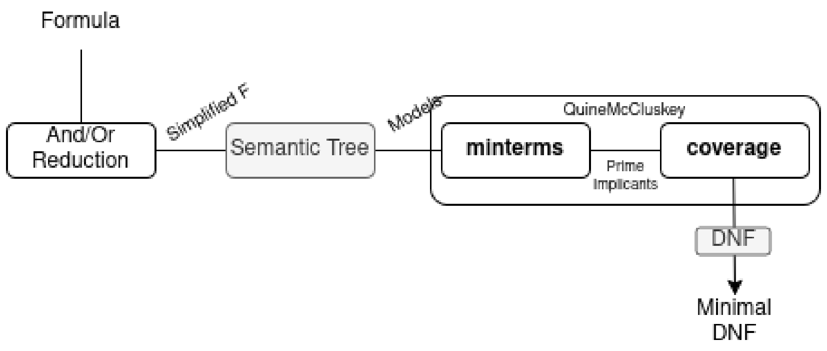
1 Introduction

In this assignment, we will implement the predicate

`minimize/2`

which takes as input an arbitrary Boolean formula and **returns all its minimal Disjunctive Normal Forms (DNFs)**. The predicate internally constructs the truth table of the formula, selects the models (valuations that satisfy the formula), and **applies the Quine–McCluskey algorithm to minimize the resulting set of terms**. The output is a list of minimal equivalent formulas in DNF, each representing a different way of expressing the original formula as a disjunction of prime implicants.

The complete process is illustrated step by step in the diagram below:



The components highlighted in gray will be provided as part of the implementation.

Some example executions are shown below:

```
?- F = (a -> b), minimize(F, Min).
F = (a->b),
Min = ~a v b.
```

```
?- F = (a -> b) v (c <-> d), minimize(F, Min).
F = (a->b)v(c<->d),
Min = ~d & ~c v (d & c v (~a v b)).


?- F = ((a -> b) v (c <-> d) -> d), minimize(F, Min).
F = ((a->b)v(c<->d)->d),
Min = a & (~b & c) v d.
```

2 And/Or Reduction

We will implement a predicate

`unfold/2`

that receives an arbitrary Boolean formula and rewrites it using only the basic connectives: conjunction, disjunction, and negation. It applies standard logical equivalences to eliminate implications and biconditionals. The resulting formula is not necessarily normalized, but it is suitable for semantic table construction.

```
?- F = (a -> b), unfold(F, Unfold, Vars).
F = (a->b),
Unfold = ~a v b.

?- F = (a -> b) v (c <-> d), unfold(F, Unfold).
F = (a->b)v(c<->d),
Unfold = ~a v b v (~c v d)&(~d v c).

?- F = ((a -> b) v (c <-> d) -> d), unfold(F, Unfold).
F = ((a->b)v(c<->d)->d),
Unfold = ~ (~a v b v (~c v d)&(~d v c))v d.
```

Note: The choice of symbols for the logical operators may differ from those shown in the examples, depending on your preferred notation or implementation.

3 Semantic Tree

You will be provided with the predicate

`tab_/2`

which takes a Boolean formula and **returns each of the open branches of its semantic tableau**. Each open branch is expressed as a **term**, that is, a conjunction of literals.

The input formula **must be expressed using only basic connectives** (conjunction, disjunction, and negation). If the original formula contains implications or biconditionals, they should be eliminated beforehand using the `unfold/2` predicate.

4 Quine-McCluskey

The final step of the pipeline involves minimizing the set of models obtained from the semantic tableau, using the Quine–McCluskey algorithm. This process is divided into two main stages: identifying prime implicants and selecting minimal covers.

Prime Implicants

To compute the prime implicants, we suggest the implementation of a predicate

`minterms/?`

which should start from the terms obtained from the semantic tableau and construct a table of implicants using `assertz/1` internally. It iteratively combines compatible terms until no further combinations are possible, leaving only the prime implicants.

For storing the implicants table we suggest to define a dynamic predicate of at least 3 arguments.

minterm/3.

This predicate should store: the id of the model (or models) represented by the minterm, th e implicant binary* representation and the size of the model (the number of models covered by it). Additionally, we will need a sort of mechanism mark implicants and therefore know which implicants are prime.

Minimal Coverage

After, we suggest the implementation of a predicate

`min_set_cover/1`

to select all minimal subsets of prime implicants that together cover all the original models of the formula.

This predicate should consult the internal implicant table to retrieve the prime implicants. Subsets of prime implicants that cover the original minterms must be identified but only those with minimal cardinality should be returned by the predicate.

5 Converting to DNF

As a result of the Quine-McCluskey step, we will have one (or more) set of prime implicants. Now this set must be converted back to a formula. To that aim, we will provide you with a predicate

todnf/2.

which receives a list of terms and converts them back to a DNF formula.

Having this into account, **you must convert the output from your coverage step to terms** before using this predicate. Some examples are shown below:

```
?- Terms = [[a, ~b, c], [d]], todnf(Terms, F).
Terms = [[a, ~b, c], [d]],
F = a&(~b&c)v d.

?- Terms = [[a, c], [d]], todnf(Terms, F).
Terms = [[a, c], [d]],
F = a&c v d.

?- Terms = [[a, c], [d, ~c]], todnf(Terms, F).
Terms = [[a, c], [d, ~c]],
F = a&c v d& ~c.

?- Terms = [[a, c], [d, ~c, a]], todnf(Terms, F).
Terms = [[a, c], [d, ~c, a]],
F = a&c v d&(~c&a).
```

Important Notes

- The use of SWIPL's `findall` predicate can simplify the implementation of this assignment a lot.
- Input formulas must admit the `->`, and `<->` operators on top of the basic conjuction, disyunction and negation.
- **A task will be enabled in moodle for the delivery of the practice, dated 16 May at 23:59 hours. Submissions after this deadline will not be evaluated.**