Foundations of Artificial Intelligence

Lorenzo Bozzoni

December 6, 2023

Contents

0.1 Model checking

Two algorithms:

1. Reasoning with truth tables
2. Exploting satisfiability (DPLL)

0.1.1 Reasoning with truth tables

Reasoning with truth tables is a form of semantic reasoning, in the sense that it directly exploits the definition of entailment: $\alpha \vDash \beta$ holds when β is true in every model that makes α true In PL, a model is an assignment of truth values (1 or 0, true or false, T or $\bot$) to every propositional symbol that appears in α or β (or both) Therefore, with n symbols we have $2n$ different models, which correspond to the rows of the truth table. For every model (row), we compute the truth values of α and β (by recursively computing the truth values of all the subsentences of α and β). Then we have that $\alpha \vDash \beta$ if, and only if, every model (row) that assigns 1 to α also assigns 1 to β.

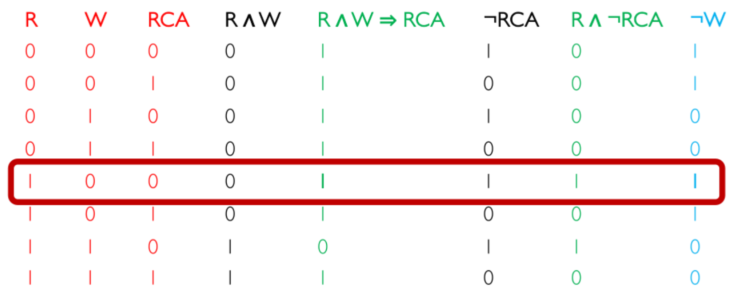
Example:
Let us consider:

- When it rains and it is windy Alice, wears a raincoat
- It rains, but Alice does not wear a raincoat
- Therefore, it is not windy

Propositional representation:

- $R \wedge W \rightarrow RCA$
- $R \wedge \neg RCA$
- $\neg W$

We want to prove $a, b \vDash c$. Here is the truth table:



R	W	RCA	R ∧ W	R ∧ W ⇒ RCA	¬RCA	R ∧ ¬RCA	¬W
0	0	0	0	1	1	0	1
0	0	1	0	1	0	0	1
0	1	0	0	1	1	0	0
0	1	1	0	1	0	0	0
1	0	0	0	1	1	1	1
1	0	1	0	1	0	0	1
1	1	0	1	0	1	1	0
1	1	1	1	1	0	0	0

This reasoning procedure is sound and complete. It always terminates, making reasoning in PL decidable. However, it is inefficient when many propositional symbols are involved, because it has to compute a table of size $2^n \times M$, where n is the number of propositional symbols and M is the number of subsentences that appear in the premises and the conclusion. Another drawback of reasoning with truth tables is that it is very unnatural, in the sense that the reasoning process is very far from the most usual forms of human reasoning. This may be problematic in those applications in which an artificial agent must be able to justify the conclusions of its reasoning processes in a way that is easily understandable for a human being.

Certain applications of PL require an agent to establish whether a set of sentences α is or is not satisfiable (i.e., whether there is an assignment of truth values to the symbols of α that makes α true).

Note: in general, set α may be finite or infinite; if it is finite, i.e., $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$, then α set is logically equivalent to the conjunction of its elements, $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$. In standard AI applications, α is always finite (although it may be very large): therefore, when it is convenient we can regard α as a single sentence with no loss of generality. The problem of establishing the satisfiability of a set of propositional sentences is known as SAT. Many interesting problems, including establishing propositional entailment, can be reduced to SAT. A (rather inefficient) solution of SAT is given by truth tables: α is satisfiable if, and only if, it has truth value 1 in at least one row of its truth table. A more efficient method is provided by the DPLL algorithm.

0.1.2 DPLL

From Davis-Putnam-Logemann-Loveland. Establish whether a set of sentences α is or is not satisfiable. Preprocessing: convert every sentence in **CNF (Conjunctive Normal Form)**. Body of the procedure: from an empty assignment, incrementally try to build a model of α (i.e., an assignment of truth values to the propositional symbols)

- if a model is built, α is satisfiable
- if the algorithm terminates without being able to build a model, α is unsatisfiable

CNF (Conjunctive Normal Form) represents a sentence as a conjunction of clauses, where a clause is a disjunction of literals and a literal is either a propositional symbol or the negation of a symbol. Every sentence of PL can be transformed in an equivalent sentence in CNF:

- $A \rightarrow B \vee C$ becomes $\neg A \vee B \vee C$
- $C \rightarrow \neg D$ becomes $\neg C \vee \neg D$

A CNF sentence is often considered as a set of clauses (in logical conjunction), which are in turn considered as sets of literals (in logical disjunction): $(\neg A \vee B \vee C) \wedge (\neg C \vee \neg D)$ becomes $\{\{\neg A, B, C\}, \{\neg C, \neg D\}\}$.

Unit clause: clause with only one literal; e.g. $\neg C$. Two literals are **complementary** if they refer to the same propositional symbol but have different "signs", like $\neg C$ and C.

Conversion to CNF, consider $A \leftrightarrow (B \vee C)$:

1. Eliminate $\leftrightarrow$, replacing $\alpha \leftrightarrow \beta$ with $(\alpha \rightarrow \beta) \vee (\beta \rightarrow \alpha)$: $(A \rightarrow (B \vee C)) \wedge ((B \vee C) \rightarrow A)$
2. Eliminate $\rightarrow$, replacing $\alpha \rightarrow \beta$ with $\neg\alpha \vee \beta$: $(\neg A \vee B \vee C) \wedge (\neg(B \vee C) \vee A)$
3. Move $\neg$ inwards using deMorgan's rule $(\neg(\alpha \wedge \beta))$ is equivalent to $(\neg\alpha \vee \neg\beta)$ and $(\neg(\alpha \vee \beta))$ is equivalent to $(\neg\alpha \wedge \neg\beta)$: $(\neg A \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee A)$
4. Apply distributivity law and flatten: $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$