Tautology verifier

A tautology verifier is a program that takes a "*propositional statement*" and verifies the statement is a "*tautology*" or not.
A Tautology is a "*propositional statement*" which is true no matter what valuation is applied for each of the variables in the "*vocabulary"* of the statement. A propositional statement is formed by combining "*propositional variables*" with arbitrary logical operators. Vocabulary of the propositional statement is the set of all the variables in the statement.
A propositional variable is one that can take truth values(true/false). Logical operators that we consider for this exercise are "*AND*" represented by "&", "*OR*" represented by "|" and "*NOT*" by "!".

Examples of valid propositional statements:
1. a
2. a & b
3. a & (b | c)
4. !a & !b
5. a | !a
6. (a & (!b | b)) | (!a & (!b | b))

Out of the above set of valid propositional statements, only 5 and 6 are tautologies. We'll leave it to you to actually work it out and internalize why they are tautologies.

Given this context, we want you to parse a given set of statements and for each statement return whether its a tautology or not.

Assumptions:
1. All statements that are given as input will be syntactically valid. (You dont need to handle invalid sentences).
2. All propositional variables will be single letter alphabets and no more than 10 variables will be there in any statement. (ie., cardinality of the vocabulary <= 10).
3. The propositional statement can be arbitrarily nested.
4. You can expected sentences to be well bracketed in case of binary operations. for ex: a | b will always be (a | b) even if it is the only operation in the statement. (You dont need to worry about associativity)
5. Whitespace can appear anywhere in the statement.
6. Only &, |, ! will be used as logical operators.
7. ! operator will always associate with the immediate right proposition.

Input:

(!a | (a & a))

(!a | (b & !a))
(!a | a)
((a & (!b | b)) | (!a & (!b | b)))

Output:
True
False
True
True

Expectation:
- Clean, simple and elegant code
- Correctness for basic set of cases that's listed above