Truth-tables
CSC/MAT 208: Discrete Structures
Department of Computer Science
Department of Mathematics and Statistics
Grinnell College

There is another approach to studying the logical relationships among Boolean expressions. Because there are only two possible values that a Boolean variable can have, it is often feasible to consider every possible assignment of values to the Boolean variables that occur in an expression or in a small group of logically related expressions.

Corresponding to the syntactic rules that define the class of Boolean expressions (presented in “The Propositional Calculus,” we can supply semantic rules that specify a Boolean value for every Boolean expression, relative to an assignment \( A \) of values to the variables occurring in that expression. If the value of a Boolean expression, relative to a given assignment, is the true Boolean value, the assignment is said to satisfy the Boolean expression.

1. The value of the literal constant ‘\( \top \)’, relative to \( A \), is the true Boolean value. The value of the literal constant ‘\( \bot \)’, relative to \( A \), is the false Boolean value. (Thus every assignment satisfies ‘\( \top \)’ and no assignment satisfies ‘\( \bot \)’.)

2. The value of a variable, relative to \( A \), is the value that \( A \) assigns to it.

3. \( A \) satisfies the negation \( \neg \phi \) of a Boolean expression \( \phi \) if, and only if, it does not satisfy \( \phi \).

4. \( A \) satisfies the conjunction \( \phi \land \psi \) of two Boolean expressions \( \phi \) and \( \psi \) if, and only if, it satisfies both \( \phi \) and \( \psi \).

5. \( A \) satisfies the disjunction \( \phi \lor \psi \) of two Boolean expressions \( \phi \) and \( \psi \) if, and only if, it satisfies either \( \phi \) or \( \psi \) or both.

6. \( A \) satisfies the implication \( \phi \rightarrow \psi \) of two Boolean expressions \( \phi \) and \( \psi \) if, and only if, it either satisfies the apodosis \( \psi \), or does not satisfy the protasis \( \phi \), or both.

7. \( A \) satisfies the equivalence \( \phi \equiv \psi \) of two Boolean expressions \( \phi \) and \( \psi \) if, and only if, if the value of \( \phi \) relative to \( A \) is the same as the value of \( \psi \) relative to \( A \).
The semantic rules for the logical connectives are often summarized in “truth-tables,” in which each row corresponds to one possible combination of values for the component expressions:

<table>
<thead>
<tr>
<th>φ</th>
<th>¬φ</th>
<th>ψ</th>
<th>(φ ∧ ψ)</th>
<th>φ</th>
<th>ψ</th>
<th>(φ ∨ ψ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>φ</th>
<th>ψ</th>
<th>(φ → ψ)</th>
<th>φ</th>
<th>ψ</th>
<th>(φ ≡ ψ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Using these semantic rules, we can work out, step by step, the value of any Boolean expression, however complicated. For instance, consider the expression \(((p → q) ∧ (p ≡ ⊥)) ≡ (¬p ∧ q))\', relative to an assignment \(A\) that assigns ‘\(p\)’ the true Boolean value and ‘\(q\)’ the false one. From the semantic rule for ‘→’, it follows that \(A\) does not satisfy ‘\((p → q)\)’. The value of ‘\(⊥\)’ is always the false Boolean value, so (by the semantic rule for ‘≡’), \(A\) does not satisfy ‘\((p ≡ ⊥)\)’, since the values of ‘\(p\)’ and ‘\(⊥\)’ are distinct. Then, by the semantic rule for ‘∧’, \(A\) does not satisfy ‘\(((p → q) ∧ (p ≡ ⊥))\)’, since it does not satisfy both of the conjuncts. On the other side of the equivalence, \(A\) does not satisfy ‘¬\(p\)’, by the semantic rule for ‘¬’, and so does not satisfy ‘\((¬p ∧ q)\)’ either (since it does not satisfy either conjunct). Finally, then, we can see \(A\) satisfies the entire expression ‘\(((p → q) ∧ (p ≡ ⊥)) ≡ (¬p ∧ q))\’), since each of the subexpressions of this equivalence has the same value relative to \(A\) (namely, the false Boolean value).

Since the Boolean expression that we have just been considering contains two Boolean variables, there are four different ways to assign values to those variables. We can build a truth-table in which each row represents a different assignment, and efficiently summarize the calculations by which we evaluate the expression relative to each of these assignments:

<table>
<thead>
<tr>
<th>(p)</th>
<th>(q)</th>
<th>((p → q))</th>
<th>⊥</th>
<th>((p ≡ ⊥))</th>
<th>(((p → q) ∧ (p ≡ ⊥)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
A Boolean expression is *satisfiable* if there is at least one assignment that satisfies it, and it is a *tautology* if every assignment satisfies it. The truth-table given above shows that ‘(((\(p \to q\)) \land (p \equiv \bot)) \equiv (\neg p \land q))’ is satisfiable but is not a tautology. The assertion that some Boolean expression \(\Gamma \phi\) is a tautology is sometimes written ‘\(\models \phi\)’.

An inference is *valid* if every assignment that satisfies all of its premisses also satisfies its conclusion. (This includes cases in which no assignment satisfies all of the premisses; such inferences are said to be valid “vacuously.”) The notation ‘\(\Gamma \models \phi\)’ is sometimes used to indicate that the inference from the premisses in the set \(\Gamma\) to the conclusion \(\phi\) is valid.

For instance, we could confirm that the *modus tollens* inference from the premisses ‘\(\neg q\)’ and ‘\((p \to q)\)’ to the conclusion ‘\(\neg p\)’ is valid by building the following truth-table:

<table>
<thead>
<tr>
<th>(p)</th>
<th>(q)</th>
<th>(\neg q)</th>
<th>((p \to q))</th>
<th>(\neg p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Only the fourth row of this truth-table represents an assignment that satisfies both premisses, and this assignment also satisfies the conclusion of the inference, so the inference is valid.

**Soundness and completeness**

Building truth-tables by hand is a somewhat tedious process. Fortunately, it can be automated. It is not difficult to write a computer program that tests any given inference for validity by generating every possible assignment of values to the variables that occur in its premisses and conclusion, evaluating the premisses and conclusion relative to every assignment, and determining whether any of these assignments satisfies all of the premisses without also satisfying the conclusion.

Using truth-tables, it is easy to confirm that the rules of inference adopted
in “The Propositional Calculus” justify only valid inferences, though in some cases (such as the rule of contradiction) the validity is vacuous (because it is impossible for both of the premisses to be true relative to the same assignment). Similarly, the axioms that we adopted, as well as the theorems that we derived, are all tautologies. (If you want some practice with truth-tables, you can look over the list of well-known theorems, pick out any that look interesting, and apply the truth-table method to them. You should wind up in each case with a column underneath each of the theorems consisting entirely of copies of the word ‘true’.

A system of axioms and rules of inference is said to be sound if the inferences made using those axioms and rules are all valid, and specifically if all of the theorems that can be proven within that system are tautologies. Checking all of the rules of inference with the truth-table method is one way to prove that our rules of inference for the propositional calculus are sound.

Ideally, we’d also like them to be complete, meaning that every tautology can be proven as a theorem (and, more generally, that every valid inference can be justified using the available rules of inference). Completeness is more difficult to establish, but it also holds for our system of axioms and rules of inference for the propositional calculus.

We can actually use the truth-table analysis of any tautology as a guide or hint to the structure of an appropriate proof of its theoremhood. Each entry in the truth-table corresponds to a useful lemma. Suppose that we have shown, by truth-tables, that $\models \phi$ (for some Boolean expression $\phi$). Consider any row of this truth-table, and let $A$ be the assignment represented by that row. Collect, as a set of premisses for this row, all of the variables to which $A$ assigns the true Boolean value and the negations of all of the variables to which $A$ assigns the false Boolean value. Now work across the row, proving, for each “true” entry, that the collected premisses entail the formula at the head of the column, and, for each “false” entry, that the collected premisses entail the negation of that formula. (Each of these proofs is quite short, and it is usually obvious which rules to use in each case.)

The last entry in each row provides a lemma showing that the collected premisses for that row entail the given tautology. The main result can then be derived from the lemmas by citing the principle of excluded middle for each of the variables and using disjunctive syllogism, contradiction, and reductio ad absurdum to remove the extra premisses one by one, leaving $\vdash \phi$ as the final result.
Exercises

1. Construct a truth-table to show that every instance of the principle of
the excluded middle is a tautology.

2. Construct a truth-table to show that \( \overline{p \equiv q}, \overline{p} \models q \).

3. If a Boolean expression contains \( n \) different variables, how many dif-
ferent assignments to those variables are there?

4. If a tautologous Boolean expression contains 64 different variables, and
our computer program can evaluate that expression relative to a given
assignment in one nanosecond, approximately how long will it take the
program to confirm that it is a tautology?

5. Suppose that we have a Scheme predicate \texttt{satisfiable?} that takes
one argument, a Boolean expression, and returns \#t if it is satisfiable
and \#f if it is not. Could we use this predicate to help us define a
similar predicate \texttt{tautology?}, which tests whether its argument is a
tautology? What else would we need in order to be able to define such
a predicate?

6. In Scheme, we can model an assignment as a list of pairs, in which
each pair consists of a character (the variable) and a Boolean value
(the value assigned to it). For instance, for an assignment that gives ‘p’
the true Boolean value and ‘q’ the false Boolean value, we could use the
Scheme value ((\texttt{\#p} . \texttt{\#t}) (\texttt{\#q} . \texttt{\#f})). Define a Scheme procedure
that takes a list of variables (characters) as its argument and returns
a list containing every possible assignment to those variables.
I am indebted to Sun Han 2015 for calling my attention to a significant typographical error in an earlier version of this document.

Copyright © 2013, 2014, 2016  John David Stone
This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by-sa/4.0/deed.en_US

or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.