truthtable: LaTeX Package for automatically generated Truth Tables

Dominic Flück <K-Trout>

0.1.0 2023/09/16

Abstract

`truthtable` is a LaTeX package for creating automatically generating truth tables given a table header. It supports a number of logical operations which can be combined as needed. It's built upon the package `luacode` and therefore has to be used with the LuaLaTeX compiler.

Contents

1 Introduction

Tables in LaTeX have the reputation of being a bit tedious. When creating a table with many cells, such as a truth table, they are not only tedious to build, but also not very readable.

To help this situation when creating a truth table for a document, this package provides a macro, which allows simply for the variables and the columns of a truth table to be defined. The package then takes care of the rest.

2 Dependencies

`truthtable` uses the package `luacode` to run, as the heavy lifting of the processing is done in *Lua*. The package checks if `luacode` is already loaded, and if not, does so. LuaLaTeX is required to compile the resulting documents.

3 Usage

The `truthtable` package provides two commands: `\truthtable` and `\truthtableasc`. Both commands take the same arguments, but generate the table in different order.

They position in the normal table boilerplate. This leads to the redundant practice of defining the column count twice, once for the table environment as the column layout and once in the command by defining the variables and statements.[1]

This is intentional to allow for more flexibility in customising the column layout as well as pre- and appending of further rows to the table.

3.1 `\truthtable`

```
\truthtable{comma separated variables}{comma separated display variables}
{comma separated statements}{comma separated display statements}
{display true value}{display false value}
```

This command generates the table in descending value of the variables, from all variables being *true* to being *false*.

3.2 `\truthtableasc`

```
\truthtableasc{comma separated variables}{comma separated display variables}
{comma separated statements}{comma separated display statements}
{display true value}{display false value}
```

This command generates the table in ascending value of the variables, from all variables being *false* to being *true*.

3.3 Comma separated variables

The basic variables, for which every combination of *true* and *false* a row of table will be generated. The variables should be relatively simple, as they are not used for the formatting the table but simply to calculate the answers. The variables should be separated using commas. Don't use variables, which contain another variable, i.e., don't do this: `{n,An}`.

3.4 Comma separated display variables

These are the display values corresponding to the *Comma separated variables*. Fancy variable formatting can be applied. At least normal text and "math" mode seem to work.[2] The same number of display variables as variables is required. The comma cannot be used as a display character, as it is used as delimiter.

[1] See Listing 1 for example

[2] More testing needs to be done

3.5 Comma separated statements

The statements using the *Comma separated variables* which are used to evaluate the statements for any given combination of variables. Parentheses can be used in the normal fashion to indicate the order of combined statements. The notation for the different operations is as follows:

3.5.1 `NOT` / Negation

To negate a variable or statement, the exclamation point `!` is used.

- $\neg A$: `!A`
- $\neg(\neg A)$: `!(!A)`

3.5.2 `AND` / Conjunction

For the conjunction of two variables or statements the and symbol `&` is used. **The & must not be escaped for the comma separated statements!**

- $A \wedge B$: `A & B`
- $A \wedge (A \wedge B)$: `A & (A & B)`

3.5.3 `OR` / Disjunction

For the Disjunction of two variables or statements the vertical line character `|` is used.

- $A \vee B$: `A | B`
- $A \vee (A \vee B)$: `A | (A | B)`

3.5.4 `XOR` / Exclusive disjunction

The exclusive disjunction (`XOR`) is written in parentheses preceded by the hat operator. **Note that the delimiter used is the semicolon ; and not the comma , ! This is because the statements are separated using the comma.**

- $A \veebar B$: `^(A; B)`
- $A \veebar (A \vee B)$: `^(A; (A | B))`

3.5.5 `NAND` / Negated conjunction

The `NAND` operation is written in parentheses preceded by the the `NOT` and the `AND` operator (`!&`). **Note that the delimiter used is the semicolon ; and not the comma , ! This is because the statements are separated using the comma.**

- $A|B$: `!&(A; B)`
- $A|(A \vee B)$: `!&(A; (A | B))`

3.5.6 $\rightarrow$ / Implication

The implication can also be expressed. **Note that the delimiter used is the semicolon ; and not the comma , ! This is because the statements are separated using the comma.**

- $A \rightarrow B$: `>>(A; B)`
- $A \rightarrow (A \vee B)$: `>>(A; (A | B))`
- $A \wedge (A \rightarrow B)$: `A & >>(A; B)`

3.5.7 $\leftrightarrow$ / Equality

The equality can also be expressed. Since version 0.0.2 this command can also be expressed as `<>(A; B)`. The previous definition[3] of `__(A; B)` also works. **Note that the delimiter used is the semicolon ; and not the comma , ! This is because the statements are separated using the comma. The `__` must not be escaped for the comma separated statements!**

- $A \leftrightarrow B$: `__(A; B)` = `<>(A; B)`
- $A \leftrightarrow (A \vee B)$: `__(A; (A | B))` = `<>(A; (A | B))`
- $A \wedge (A \leftrightarrow B)$: `A & __(A; B)` = `A & <>(A; B)`

3.6 Comma separated display statements

Display statements are defined the same way as the *comma separated display variables*. The comma cannot be used as a display character, as it is used as delimiter.

3.7 Display true value

The displaying string which will be used in the table body for *true*. Normal text and "math" mode can be used.

3.8 Display false value

The displaying string which will be used in the table body for *false*. Normal text and "math" mode can be used.

4 Example of use

The code snippet seen in Listing 1 is the entirety of code required to produce the truth table seen in Table 1.[4] The code for the ascending truth table as seen in Table 2 is identical, except for the command used being `\truthtableasc` instead of `\truthtable`.

The command generates the code seen in Listing 2.

Listing 1: Code for an sample truth table

```
\begin{table}[h]
 \centering
 \begin{tabular}{c|c||c|c|c|c|c|c|c}

  % Content of table is generated using this single command.
  \truthtable{A,B}{$A$,$B$}
  {!A, A & B, A | B, ^(A; B), !&(A; B), >>(A; B), <>(A; B)}{$\lnot A$, $A \land B$, $A \lor B
      $,$A \veebar B$,$A | B$, $A \rightarrow B$, $A \leftrightarrow B$}
  {$T$}{$F$}

 \end{tabular}
\end{table}
```

Listing 2: Code generated by \truthtable

```
$A$ & $B$ & $\lnot A$ &  $A \land B$ &  $A \lor B$ & $A \veebar B$ & $A | B$ &  $A \
    rightarrow B$ &  $A \leftrightarrow B$ \\ \hline
$T$ & $T$ & $F$ & $T$ & $T$ & $F$ & $F$ & $T$ & $T$ \\
$T$ & $F$ & $F$ & $F$ & $T$ & $T$ & $T$ & $F$ & $F$ \\
$F$ & $T$ & $T$ & $F$ & $T$ & $T$ & $T$ & $T$ & $F$ \\
$F$ & $F$ & $T$ & $F$ & $F$ & $F$ & $T$ & $T$ & $T$ \\
```

[3] The equality operation was defined this way in v0.0.1

[4] The captioning setup was omitted in the listing.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \veebar B$	$A \mid B$	$A \rightarrow B$	$A \leftrightarrow B$
T	T	F	T	T	F	F	T	T
T	F	F	F	T	T	T	F	F
F	T	T	F	T	T	T	T	F
F	F	T	F	F	F	T	T	T

Table 1: Sample truth table

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \veebar B$	$A \mid B$	$A \rightarrow B$	$A \leftrightarrow B$
F	F	T	F	F	F	T	T	T
F	T	T	F	T	T	T	T	F
T	F	F	F	T	T	T	F	F
T	T	F	T	T	F	F	T	T

Table 2: Sample truth table in ascending order

5 Development

5.1 Repository

This package is on *CTAN* (ctan.org/pkg/truthtable). The repository of the package is github.com/K-Trout/truthtable. For bug reports and feature requests create an issue on GitHub: github.com/K-Trout/truthtable/issues.

5.2 Changes

v0.1.0 (2023/09/16)

- Added the `\truthtableasc` command to generate the table in ascending order. Thanks to *slnkahveci* on GitHub for the feature request.
- Fixed alignment issues in the first column of the table. Thanks to *cpierquet* on GitHub for the bug report.

v0.0.2 (2021/10/08)

- Added support for *XOR* and *NAND*.
- Added definition for equivalence operation to be written as `<>(A; B)`. `__(A; B)` is still supported
- Added some error messages when the number of arguments and display arguments don't correspond.

v0.0.1 (2021/10/01)

- Initial release

5.3 Known issues and bugs

Stability The Lua code of the macro is not very error resistant. The package only checks if the same amount of working and display variables, as well as working and display statements are provided. If a mismatch is detected, an error message is output and the package code halts. Further improvements may be undertaken in the future.

Display formatting Whilst normal text and "math" mode work for both headers and truth values, other text formatting such as `\textbf` does not. It is not yet clear if this will be addressed in future versions.

Operations For the moment seven operations are defined. Further operations may be added in future versions.

6 Implementation

Listing 3: Source code of the truthtable package

```
% truthtable.sty
%% Copyright 2021 D. Flück
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status "author-"maintained.
%
% The Current Maintainer of this work is D. Flück.
%
% This work consists of the file truthtable.sty.
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesPackage{truthtable}[2023/09/16 0.1.0 Package for generating truth tables
    automatically using LuaTeX]

\ProcessOptions\relax
\@ifpackageloaded{luacode}{
 \PackageWarningNoLine{truthtable}{Package luacode was already loaded}
}{
 \RequirePackage{luacode}
}

\begin{luacode*}

function Impl(a,b)
 return (not a or b);
end

function Equiv(a,b)
 return ((a and b) or ((not a) and (not b)));
end

function Xor(a,b)
 return ((a or b) and (not (a and b)));
end

function Nand(a,b)
 return (not (a and b));
end

function ComputeRows(header)
 return 2^header
end

function Split(s, delimiter)
    local result = {};
    for match in (s..delimiter):gmatch("(.-)"..delimiter) do
        table.insert(result, match);
    end
    return result;
end

function EvaluateFormula(formula)

 local parsedFormula = "function res() return( " .. string.gsub(string.gsub(string.gsub(
     string.gsub(string.gsub(string.gsub(string.gsub(string.gsub(string.gsub(string.gsub(
     formula, " ", ""),">>","Impl"),"__","Equiv"),"<>","Equiv"),"%^","Xor"),"!&","Nand")
     ,"!","not "),"&" ," and "),"|"," or "),";",",") .. " ) end";

 chunk = load(parsedFormula);
```

```
 chunk();
 local result =  res();
 return result;
end

function toBits(num)
    local t = "" -- will contain the bits
    while num>0 do
        local rest = math.fmod(num,2)
        if (rest == 1) then
   t =  "1" .. t
  else
   t = "0" .. t
  end

        num=(num-rest)/2
    end
    return t;
end

function printTruthValue(expr, dTrue, dFalse)

 local returnVal = ""

 if (expr) then
  returnVal = dTrue;
 else
  returnVal = dFalse;
 end

 return returnVal;
end

function parse(commaSepVariables, commaSepDisplayVariables, commaSepResultRows,
    commaSepResultDisplayRows, displayTrue, displayFalse, order)

 print("\n\ntruthtable v0.1.0\n")

 local vrbls = Split(commaSepVariables, ",");
 local numberOfColumns = #(vrbls);
 local rows = ComputeRows(numberOfColumns);
 local dVrbls = Split(commaSepDisplayVariables, ",");
 local resRows = Split(commaSepResultRows, ",");
 local dResRows = Split(commaSepResultDisplayRows, ",");

 local dHeader = string.gsub(commaSepDisplayVariables, ",", " & ") .. " & " .. string.gsub(
     commaSepResultDisplayRows, ",", " & ") ..  [[ \\ \hline]];

 if (#(dVrbls) ~= #(vrbls)) then
  print("Error: The number of variables does not match the number of display variables.");
  return
 end

 if (#(dResRows) ~= #(resRows)) then
  print("Error: The number of statements does not match the number of display statements.")
      ;
  return
 end

 tex.print(dHeader);


 local startVal;
 local endVal;
 local stepVal;

 if order == "asc" then
  startVal = 0;
  endVal = rows - 1;
  stepVal = 1;
```

```
 else
  startVal = rows - 1;
  endVal = 0;
  stepVal = -1;
 end

 for i = startVal,endVal,stepVal
 do
  local bitString = toBits(i);

  while #bitString < numberOfColumns do
   bitString = "0" .. bitString
  end

  local wVrbls = commaSepVariables;
  local wCommaSepRows = commaSepResultRows
  for ii = 1,numberOfColumns
  do
   wVrbls = string.gsub(wVrbls, vrbls[ii], (string.sub(bitString,ii,ii) == "1" ) and "+" or
        "-" )
   wCommaSepRows = string.gsub(wCommaSepRows, vrbls[ii], (string.sub(bitString,ii,ii) ==
       "1" ) and "+" or "-" )
  end

  local aWVrbls = Split(string.gsub(string.gsub(wVrbls, "+", "true"),"-", "false"), ",");

  local aWCommaSepRows = Split(string.gsub(string.gsub(wCommaSepRows, "+", "true"),"-", "
      false"), ",");

  local row = "";

  for c = 1,#(aWVrbls)
  do
   row = row .. printTruthValue(EvaluateFormula(aWVrbls[c]), displayTrue, displayFalse) ..
       " & ";
  end

  for c = 1,#(aWCommaSepRows)
  do
   row = row .. printTruthValue(EvaluateFormula(aWCommaSepRows[c]), displayTrue,
       displayFalse) .. " & ";
  end

  row = string.sub(row, 1, #row - 2) .. [[\\]];

  tex.print(row);
 end
end

\end{luacode*}

\newcommand{\truthtable}[6]{
 \luadirect{parse("#1", "\luaescapestring{#2}", "\luaescapestring{#3}", "\luaescapestring
     {#4}", "\luaescapestring{#5}","\luaescapestring{#6}", "\luaescapestring{desc}")}
}

\newcommand{\truthtableasc}[6]{
 \luadirect{parse("#1", "\luaescapestring{#2}", "\luaescapestring{#3}", "\luaescapestring
     {#4}", "\luaescapestring{#5}","\luaescapestring{#6}", "\luaescapestring{asc}")}
}

\endinput
```