ConTEXt

title : ConTEXt User Module

subtitle : The Ratiocinator – A Propositional Logic Toolkit

author : Philipp Gesang

date : February 25, 2011

1 Preliminary remarks

1.1 License

Licensed under the BSD-License with 2 clauses.

```
Copyright 2011 Philipp Gesang. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
    1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

1.2 Wishful thinking

TODO list:

A manual.
`\defineexpression`
`\definenotation`
More configurability.
`\defineinterpretation`
`\setupnotation`

Something like

```
\definenotation [set] [tex]
\setupnotation [set] [
   NOT=\neg,
   OR=\cup,
   AND=\cap,
   IFTHEN=\supset,
]
```

would be great.

2 Module and namespace initialization

```
\writestatus{loading}{ConTeXt User Module / Ratiocinator}
```

```
\unprotect
```

```
\startinterface all
  \setinterfacevariable {rat} {rat}
\stopinterface
```

Module namespace definition.

```
\definenamespace [rat] [
  type=module,
  comment=Ratiocinator module,
  version=hg-r35+,
  name=rat,
  style=no,
  command=yes,
  setup=list,
  parent=rat,
]
```

\setuprat Expected module parameters:

compress	boolean: compress truth tables
highlightcolor	color of your choice
notation	pm, pm_tex, tex, unicode
truthstyle	boole, frege, unicode, tf
treefont	font for use with GraphViz
labelfont	font to pass to MP as “defaultfont”

The *treefont* needs to be a valid fontconfig matching pattern.

```
\setuprat[
  compress=false,
  notation=tex,
  truthstyle=unicode,
  highlightcolor=gray:2,
  treefont={CMU Serif},
  labelfont={latin-modern},
  \c!width=,      % locally, for GraphViz inclusion
  \c!height=,     % item
]
```

Loading the Lua internals.

```
\startluacode
environment.loadluafile("ratiocinator")
local rat = thirddata.ratiocinator
\stopluacode
```

Verbose logging can now be switched on via:

```
\enabletrackers[ratiocinator.*]
\enabletrackers[ratiocinator.module]
\enabletrackers[ratiocinator.notations]
```

3 Main User Interface

\ratio

The macro \ratio takes an expression in the input notation and converts it into the specified notation.

- First argument: local setups;
- second argument: input expression.

```
\def\do_ratio[#1]#2{%
  \bgroup \ifsecondargument
    \setuprat[#1]%
  \fi
  \startluacode
    local rat = thirddata.ratiocinator
    rat.options.notation.output = rat.notations["\ratparameter{notation}"]
    local parsed = rat.tree ("\luaescapestring{#2}")
    context(rat.group(parsed))
  \stopluacode%
  \egroup%
}
```

```
\def\ratio{\dosingleempty\do_ratio}
```

macros definetruthvalues,showtruthvalues

The macro \definetruthvalues converts a comma-separated list into an internal hashtable of truth value assignments to atoms.

```
\def\do_define_truthvalues[#1][#2]{%
  \ifsecondargument
    \iffirstargument
      \ctxlua{thirddata.ratiocinator.define_truthvalues("#1", "\luaescapestring{#2}")}
    \fi
  \fi
}
```

```
\def\definetruthvalues{\dodoubleempty\do_define_truthvalues}
```

The macro \showtruthvalues pretty prints a previously defined truth value assignment as a table. Via the optional first argument the name of a truth style in which to represent the values may be specified.

```
\def\do_show_truthvalues[#1]#2{%
  \iffirstargument
    \ctxlua{thirddata.ratiocinator.options.notation.truth
              = thirddata.ratiocinator.notations.truth["#1"]}%
  \else
    \ctxlua{thirddata.ratiocinator.options.notation.truth
              = thirddata.ratiocinator.notations.truth["\ratparameter{truthstyle}"]}%
  \fi
  \ctxlua{thirddata.ratiocinator.show_truthvalues("#2")}%
}
```

```
\def\showtruthvalues{\dosingleempty\do_show_truthvalues}
```

\evaluate The macro `\evaluate` evaluates an expression against a previously defined truth value assignment.

```
\definetruthvalues[zeroone] [p=0,q=1]
\definetruthvalues[zerozero][p=0,q=0]
\evaluate[zeroone] {p*q}
\evaluate[zeroone] {p+q}
\evaluate[zerozero]{p*q}
\evaluate[zerozero]{p+q}
```

```
\def\do_evaluate[#1][#2]#3{%
  \bgroup \ifsecondargument
    \setuprat[#2]%
  \fi
  \startluacode
    local rat = thirddata.ratiocinator
    local on  = rat.options.notation
    on.output = rat.notations["\ratparameter{notation}"]
    on.truth  = rat.notations.truth["\ratparameter{truthstyle}"]
    local parsed = rat.tree("\luaescapestring{#3}")
    local _, _, value = rat.evaluate(parsed, rat.truthvalues["#1"])
    context(on.truth[value])
  \stopluacode%
  \egroup%
}
```

```
\def\evaluate{\dodoubleempty\do_evaluate}
```

\truthtable The macro `\truthtable` receives an ascii expression and evaluates for each combination of propositional variables and truth values.

- First argument: local setups;
- second argument: input expression.

```
\def\do_truth_table[#1]#2{%
  \bgroup \iffirstargument
    \setuprat[#1]%
  \fi
  \startluacode
    local rat = thirddata.ratiocinator
    rat.options.notation.output = rat.notations["\ratparameter{notation}"]
    rat.ctx.complete_tt("\luaescapestring{#2}", \ratparameter{compress})
  \stopluacode
  \egroup%
}
```

```
\def\truthtable{\dosingleempty\do_truth_table}
```

\syntaxtree The macro `\syntaxtree` generates a dotfile from the syntactical structure of an expression. The resulting code is then passed on to *GraphViz* in order to obtain an includable `.pdf`-graphic.

- First argument: local setups + `treefont`;
- second argument: input expression.

```
\def\do_syntax_tree[#1]#2{%
  \bgroup \ifsecondargument
    \setuprat[#1]%
  \fi
  \startluacode
    local rat = thirddata.ratiocinator
    rat.options.notation.output = rat.notations["\ratparameter{notation}"]
    rat.options.treefont = "\luaescapestring{\ratparameter{treefont}}"
    rat.ctx.syntaxtree("\luaescapestring{#2}")
  \stopluacode
  \externalfigure
    [\ctxlua{context(thirddata.ratiocinator.current_filename)}]
    [\c!width=\ratparameter{width},\c!height=\ratparameter{height}]%
  \egroup%
}
```

```
\def\syntaxtree{\dosingleempty\do_syntax_tree}
```

\venndiagram

The macro `\venndiagram` evaluates an expression against the complete truth table; the values obtained are, then, converted into colors of a Venn diagram.

NB: although the algorithm generates output for expressions with $3 < \#\text{atoms}$, the resulting diagrams are not valid (some intersections not visible).

Got some spare time? Read this nice website about Venn diagrams, or Venn's book.

- First argument: local setups, + `labelfont`;
- second argument: input expression.

```
\def\do_venn_diagram[#1]#2{%
  \bgroup \ifsecondargument
    \setuprat[#1]%
  \fi
  \startluacode
    local rat = thirddata.ratiocinator
    rat.options.notation.output = rat.notations["\ratparameter{notation}"]
    rat.options.labelfont = "\luaescapestring{\ratparameter{labelfont}}"
    thirddata.ratiocinator.venn("\luaescapestring{#2}")
  \stopluacode%
}
```

```
\def\venndiagram{\dosingleempty\do_venn_diagram}
```

```
\protect \endinput
```