KRR Lab 2

Introduction

Before you begin

If you do not know what a Backus-Naur Form (BNF) is, please read this Wikipedia article before going any further.

0.1 Propositional logic

The language of propositional logic consists of constants ($\top$ = true, $\bot$ = false), propositions (p, q, r, ...) and operators ($\neg$ = negation, $\wedge$ = conjunction / and, $\vee$ = disjunction / or, $\rightarrow$ = implication / if ... then, $\leftrightarrow$ = equivalence / if and only if). The formulas (or sentences) of propositional logic (using a reduced set of propositions) are defined by the following BNF:

```
ϕ ::= ⊤ | ⊥ | π | ¬ϕ | ϕ ∧ ϕ | ϕ ∨ ϕ | ϕ → ϕ | ϕ ↔ ϕ | (ϕ)
π ::= p | q | r | s | t | u | v
```

For example, $\phi = p \vee q \wedge r \wedge s \leftrightarrow \neg t \rightarrow u$ is a formula. In order to properly read this formula, we rely on the following operator precedence (highest priority first):

- negation
- conjunction
- disjunction
- implication
- equivalence

We will also assume right associativity for binary operators, in the sense that whenever an operand lies between two operators with the same precedence, it will associate with the operator on its right: $p \wedge q \wedge r$ is the same as $p \wedge (q \wedge r)$.

With the previous considerations, note that the formula ϕ is equivalent to the following fully parenthesized formula:

$$(p \vee (q \wedge (r \wedge s))) \leftrightarrow ((\neg t) \rightarrow u)$$

An interpretation of a propositional formula consists in the assignment of a truth value to each proposition from the language and the computation of a truth value for the sentence based on truth tables for the operators. For example, for the sentence $p \wedge \neg p \rightarrow q$ we can build the following truth table:

The following cases are possible:

p	q	$\neg p$	$p \wedge \neg p$	$p \wedge \neg p \rightarrow q$
$\top$	$\top$	$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\bot$	$\top$
$\bot$	$\bot$	$\top$	$\bot$	$\top$

- the formula is true for all possible interpretations $\Rightarrow$ valid formula / tautology
- the formula is false for all possible interpretations $\Rightarrow$ unsatisfiable / inconsistent formula
- the formula is true for some interpretations and false for other interpretations $\Rightarrow$ contingent formula

The example formula is, thus, a tautology.

The following equivalences are valid for any formulas ϕ and ψ:

$$\begin{aligned}
\neg\top &\leftrightarrow \bot \\
\neg\bot &\leftrightarrow \top \\
\phi \vee \neg\phi &\leftrightarrow \top \\
\phi \wedge \neg\phi &\leftrightarrow \bot \\
\neg\neg\phi &\leftrightarrow \phi \\
\neg(\phi \wedge \psi) &\leftrightarrow \neg\phi \vee \neg\psi \\
\neg(\phi \vee \psi) &\leftrightarrow \neg\phi \wedge \neg\psi \\
\phi \rightarrow \psi &\leftrightarrow \neg\phi \vee \psi \\
(\phi \leftrightarrow \psi) &\leftrightarrow (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)
\end{aligned}$$

Based on such equivalences we can use substitution in order to convert formulas to particular forms. For example, we can write

$$\begin{aligned}
(\phi \leftrightarrow \psi) &\leftrightarrow (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \\
&\leftrightarrow (\neg\phi \vee \psi) \wedge (\neg\psi \vee \phi) \\
&\leftrightarrow \neg(\phi \wedge \neg\psi) \wedge \neg(\psi \wedge \neg\phi)
\end{aligned}$$

Note that the last formula has the remarkable property that it only contains the $\neg$ and $\wedge$ operators. It can be proved that any propositional formula is equivalent to a formula that only uses propositions, negation and conjunction (i.e. the constants are not needed either).

The following sections describe the tasks that you should implement during the lab.

1 Read - 4p

Using a programming language of your choice, you should read several propositional formulas from an input file. The syntax for the propositional formulas in the input file is given by the following BNF:

```
<formula> ::= "T" | "F" | <prop> | "!"<formula>
    | <formula> <op> <formula> | "(" <formula> ")"
<prop> ::= "p" | "q" | "r" | "s" | "t" | "u" | "v"
<op> ::= "&" | "|" | "->" | "<->"
```

Note that there may be any number of spaces between consecutive tokens in the formula. The data structure that you use internally for the representation of the read formulas is your own choice, but a tree-like structure is hignly recommended. For each formula read from the input file you should print its fully parenthesized version to an output file "read.txt". You may assume that the provided formulas are well formed (parentheses are correctly paired etc).

A partial implementation of the read function is provided for the Scheme language. The function reads the tokens that compose a formula into a list. Sublists are generated whenever parentheses are encountered. Note that you still have to implement the function that computes the full parenthesized version of a formula. You are also advised to change the order of the symbols in the list so that the operator comes first (this will help you write less when you solve the other tasks).

Bonus 1p: Write a function that takes a propositional formula and removes all unnecessary parentheses. It is easier if you start from the fully parenthesized version of the formula and transform it recursively.

2 Evaluate - 3p

For each of the propositional formulas read from the input file, determine the distinct subformulas, then compute the corresponding truth table containing the truth value of the formula for each possible interpretation. The subformulas will be the column headers for the table, just as in the example provided in the introduction. Note that the truth constants need not be in the table, as their value does not change with the interpretation of the propositions. The syntax for the table content will be adjusted to fit the BNF of the input file. The formula $p \wedge \neg p \to q$ will be read from the input file as `p & !p -> q` and will produce the following truth table:

p	q	!p	p & !p	p & !p -> q
T	T	F	F	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

Based on the obtained truth values for all possible interpretations, you should also say whether the formula is valid, contingent or inconsistent. Write the output in a file named "evaluate.txt".

Bonus - 1p: The actual layout of the table in text mode is your own choice, but one bonus point will be given for well aligned representations.

3 Convert - 3p

Using the equivalences provided in the introductory section, convert each propositional formula read from the input file to an equivalent formula that only uses propositions, negation and conjunction. Test your conversion by evaluating the equivalence between the original formula and the converted one (you should get that this equivalence is valid). Write the output in a file “convert.txt”.

Bonus - 1p: Convert the read propositional formulas to an equivalent form that uses only propositions, negation and implication.