AIT Semantic and Declarative Technologies Course

Practice L1: Propositional Logic

In tasks requiring that you show that some formula is true, try not to use "brute force" techniques such as writing down the truth table for the formula. Instead, please use case based reasoning, i.e. considering the two cases of $X = 0$ and $X = 1$, for some variable(s) X in the formula. You can also transform the formula using equalities/equivalences shown in earlier tasks.

Boolean operations

In the examples on this page we assume that A denotes "Adam is guilty", and B denotes "Ben is guilty".

- Negation

  Negation of A is denoted by $\neg A$ or `not` A.
  Read $\neg A$ as "It is not the case that A holds".
  Example: $\neg A$ can be read as "Adam is not guilty".

A	$\neg A$
0	1
1	0

- Conjunction

  Conjunction of A and B is denoted by $A \wedge B$ or A `and` B.
  Read $A \wedge B$ as "Both A and B hold".
  Conjunction is commutative and associative.
  Example: $A \wedge B$ can be read as "Both Adam and Ben are guilty".

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

- Disjunction

  Disjunction of A and B is denoted by $A \vee B$ or A `or` B.
  Read $A \vee B$ as "At least one of A and B hold".
  Disjunction is commutative and associative.
  Example: $A \vee B$ can be read as "At least one of Adam and Ben is guilty".

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

- Implication

  Implication "A implies B" is denoted by $A \rightarrow B$ or A `implies` B.
  Read $A \rightarrow B$ as "If A holds then B hold".
  Implication is **not** commutative and **not** associative.
  Example: $A \rightarrow B$ can be read as "If Adam is guilty, so is Ben".

A	B	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

- Equivalence

  Equivalence of A and B is denoted by $A \leftrightarrow B$, $A \equiv B$, or A `equiv` B.
  Read $A \leftrightarrow B$ as "A holds if, and only if, B holds".
  Equivalence is commutative and associative.
  Example: $A \leftrightarrow B$ can be read as "Adam is guilty if, and only if, Ben is guilty".

A	B	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

- Exclusive Or

  Exclusive or of A and B is denoted by $A \oplus B$, or A `xor` B.
  Read $A \oplus B$ as "Exactly one of A and B hold", or "A or B hold, but not both".
  Exclusive or is commutative and associative.
  Example: $A \oplus B$ can be read as "Exactly one of Adam and Ben is guilty" or "One of Adam and Ben is guilty, but not both".

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

1. In this practice we examine the behavior of the five basic binary Boolean operations: `and`, `or`, `implies`, `xor` and `equiv`. We will first consider operations having both 0 and 1 operands, then operations on two 0's, and finally those on two 1's. Look out for the most "stable" pair of operands, in the following sense. Count the number of zeros and ones returned by the five Boolean operations on a given pair, and determine when do you get the largest count.
   a) `0 and 1 =`
   b) `0 or 1 =`
   c) `0 implies 1 =`
   d) `1 implies 0 =`
   e) `0 xor 1 =`
   f) `0 equiv 1 =`

   g) `0 and 0 =`
   h) `0 or 0 =`
   i) `0 implies 0 =`
   j) `0 xor 0 =`
   k) `0 equiv 0 =`

   l) `1 and 1 =`
   m) `1 or 1 =`
   n) `1 implies 1 =`
   o) `1 xor 1 =`
   p) `1 equiv 1 =`
2. Determine the value of the Boolean expression below.
   a) `0 implies (0 implies 0) =`
   b) `(0 implies 0) implies 0 =`
   c) Based on the results of the previous two tasks, can you decide if the operation `implies` is associative?
   d) `0 xor 0 =`
   e) `(0 xor 0) xor 0 =`
   f) `1 xor 0 =`
   g) `(1 xor 0) xor 0 =`
   h) Consider the operation U `xor 0`. Will the result be the same as U, or will it be its negation?
   i) `0 xor 1 =`
   j) `(0 xor 1) xor 1 =`
   k) `1 xor 1 =`
   l) `(1 xor 1) xor 1 =`
   m) Consider the operation U `xor 1`. Will the result be the same as U, or will it be its negation?
   n) Assume that a Boolean expression contains `xor` operations only, and there are n 1's and m 0's appearing as the operands. Example: `(1 xor 0) xor 1`. Here $n = 2$ and $m = 1$.

      Write an arithmetic test involving n and/or m, which expresses the condition that this given Boolean expression evaluates to 1. You can use the operators `+`, `-`, `*`, `/`, `mod`, `=`, `<` and `>`.

      (As a related example, consider a Boolean expression containing `or` operations only. In this case a solution could be the following: $n > 0$.)
   o) `0 equiv 0 =`
   p) `(0 equiv 0) equiv 0 =`
   q) `1 equiv 0 =`
   r) `(1 equiv 0) equiv 0 =`
   s) Consider the operation U `equiv 0`. Will the result be the same as U, or will it be its negation?
   t) `0 equiv 1 =`
   u) `(0 equiv 1) equiv 1 =`
   v) `1 equiv 1 =`
   w) `(1 equiv 1) equiv 1 =`
   x) Consider the operation U `equiv 1`. Will the result be the same as U, or will it be its negation?
   y) Assume that a Boolean expression contains `equiv` operations only, and there are n 1's and m 0's appearing as the operands. Write an arithmetic test involving n and/or m, which expresses the condition that this given Boolean expression evaluates to 1.

3. In this practice we explore Boolean expressions with a single variable, say x. Any such expression is equivalent to one of the following four functions: constant zero (`0`), identity (x, i.e. the function returns its input unchanged), negation (`not` x) and constant one (`1`),

   Determine which of the four basic functions is equivalent to the Boolean expressions below.

   a) `x and x`
   b) `x or x`
   c) `x implies x`
   d) `x xor x`
   e) `x equiv x`
   f) `x and 0`
   g) `x or 0`
   h) `x implies 0`
   i) `0 implies x`
   j) `x xor 0`
   k) `x equiv 0`
   l) `x and 1`
   m) `x or 1`
   n) `x implies 1`
   o) `1 implies x`
   p) `x xor 1`
   q) `x equiv 1`
   r) `not (not x)`
   s) `(not x) and x`
   t) `(not x) or x`
   u) `(not x) implies x`
   v) `x implies (not x)`
   w) `(not x) xor x`
   x) `(not x) equiv x`
   y) `(x xor x) xor x`
   z) `(x equiv x) equiv x`

4. Prove the equalities below using case-based reasoning.

   a) `u and (v or w) = (u and v) or (u and w)`
   b) `u or (v and w) = (u or v) and (u or w)`
   c) `u or (u and w) = u`
   d) `u and (u or w) = u`
   e) `u implies v = not u or v`
   f) `u implies (u or w) = 1`
   g) `not u implies (u implies v) = 1`
   h) `v implies (u implies v) = 1`
   i) `u implies v = not v implies not u`
   j) `(u and (u implies v)) implies v = 1`
   k) `(u or v) and (w or not v) implies (u or w) = 1`
   l) `not (u and v) = (not u) or (not v)`
   m) `not (u or v) = (not u) and (not v)`
   n) `not (u implies v) = u and (not v)`
   o) `not (u xor v) = (not u) xor v`
   p) `not (u xor v) = u equiv v`
   q) `not (u equiv v) = (not u) equiv v`
   r) `u xor (v xor 1) = v equiv u`
   s) `(u or v) and (u or not v) = u`