DISCRETE STRUCTURES
Lab 2
Fundamentals of Logic

Trần Hồng Tài

Abstract

In this Laboratory, we will practice using python on Logic expression and statement.

1 Fundamentals of Logic in Python

The connection between logic, proofs and programming is a very rich and interesting field that goes far byond the scope of an introductory course in discrete mathematics. But instead of illustrating this point here by introducing loads of material not covered in the text book, we will limit our ambitions to simply emphasizing the obvious: the similarities between logical truth values and Pythons boolean type and its operations.
In Python terms, a logical statement is an expression of boolean type, such as 3 $<$ 7. Variables can be used in such expressions, although it should be noted that every variable will stand for some concrete value at run-time. Computing with open statements is also possible in Python, but the techniques required will not be described in this note.
Of the logical connectives, Python directly supports conjunction and disjunction in terms of the built-in primitives and and or, as well as negation via the primitive not. Other connectives can easily be encoded as functions by use of the if statement. The implication arrow, for example, may be defined as follows:

```
def implies(a,b):
  if a:
    return b
  else:
    return True
```

To express the logical statement $x \geq 0 \wedge y \geq 0 \rightarrow x * y \geq 0$ in Python one simply writes:

```
  implies(x>=0 and y>=0, v*y>=0)
```

Notice, though, that the Python interpreter will merely check whether the implication statement holds for the particular values assigned to v and y at run-time. To check the validity of the statement in general, one has to run it for every possible combination of values for x and y — a daunting task even considering the limited range of Python's integers! Alternatively, one may attempt to prove the statement by logical means, but it should be noted that this is an activity that is quite different from repeated evaluation of boolean expressions. Whether computers can be helpful in constructing proofs as well is a question best deferred to a later course. For more information, read following lecture notes of Discrete Structures on site elit.tdtu.edu.vn

2 Exercises

1. Write python functions to calculate:

   (a) logical implies r=lImplies(p,q)

   (b) logical and r=lAnd(p,q)

   (c) logical or r=lOr(p,q)

   (d) logical xor r=lXor(p,q)

   (e) logical not r=lNot(p)

   (f) logical equivalent r=lEquipvalent(p,q)

   where p,q and r are 3 logical variables.
   Hint: to define a function in python we use:

```
def functionname(inputvariables):
  #calculate
  return outputvariables
```

2. Write python functions to calculate:

   (a) logical implies R=lLImplies(P,Q)

   (b) logical and R=lLAnd(P,Q)

   (c) logical or R=lLOr(P,Q)

   (d) logical xor R=lLXor(P,Q)

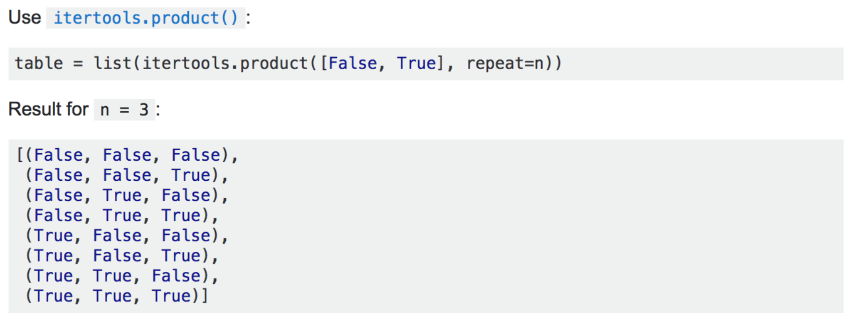
   (e) logical not R=lLNot(P)

   (f) logical equivalent R=lLEquivalent(P,Q)

   where P,Q and R are 3 lists of logical variables
   for example with:
   $P = [True, True, False, False];$
   $Q = [True, False, True, False]:$
   Then we have:

P	Q	R=					
		lLImplies(P,Q)	lLAnd(P,Q)	lLOr(P,Q)	lLXor(P,Q)	lLNot(P)	lLEquivalent(P,Q)
T	T	T	T	T	F	F	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

3. Write python program to calculate and print truth table for following expressions:
   $p \wedge q \rightarrow r$
   $r \rightarrow p \wedge\ q$
   The table should contain a row for each possible combination of values for variables p, r and q and columns for the final result as well as each relevant Boolean subexpression.
   Hint:import function product from library **itertools**.



Use `itertools.product()`:

```
table = list(itertools.product([False, True], repeat=n))
```

Result for `n = 3`:

```
[(False, False, False),
 (False, False, True),
 (False, True, False),
 (False, True, True),
 (True, False, False),
 (True, False, True),
 (True, True, False),
 (True, True, True)]
```

4. Write python programs to calculate and print truth table for following expressions:
   $p \vee q \rightarrow p \wedge q \rightarrow \neg p \vee \neg q$
   $\neg p \vee (q \wedge r) \rightarrow r$
   $(p \rightarrow q) \wedge (q \rightarrow r)$
   $(p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (p \vee r))$
   $p \vee q \rightarrow \neg r \vee t$
   $p \vee t \rightarrow q \rightarrow (r \rightarrow t)$
   $(p \vee (q \wedge r)) \leftrightarrow (((p \vee q) \wedge (p \vee r)) \wedge (t \vee \neg t))$

5. Write python programs to tell if the following formulas are equivalent: (The programs should print “equivalent” or “Inequivalent” as result)
   $p \equiv \neg(\neg p)$
   $\neg(\neg q \wedge p) \wedge (q \vee p) \equiv q$
   $\neg(p \vee q) \equiv (\neg p \vee \neg q)$
   $(p \vee q) \rightarrow r \equiv (p \rightarrow r) \wedge (q \rightarrow r)$
   $\neg(p \wedge q) \equiv (\neg p \wedge \neg q)$

$(p \lor \neg q) \to \neg p \equiv (p \lor (\neg q)) \to \neg p$
$\neg(p \lor q) \equiv (\neg p \land \neg q)$

6. Write python programs to show the validity of the following argument: (the programs should simply print "INVALID" or "VALID" as result)

   (a) $p \to r$
   $\neg p \to q$
   $q \to s$
   $\therefore \neg r \to s$

   (b) $p \to (q \to r)$
   $p \lor s$
   $t \to q$
   $\neg s$
   $\therefore \neg r \to \neg t$

   (c) $p \to q$
   $\neg r \lor s$
   $p \lor r$
   $\therefore \neg q \to s$

   (d) p
   $p \to r$
   $p \to (q \lor \neg r)$
   $\neg q \lor \neg s$
   $\therefore s$