Monash University
Faculty of Information Technology
Semester 2, 2022

FIT2014
Tutorial 2
Quantifiers, Games, Proofs

ASSESSED PREPARATION: Question 1.
You must provide a serious attempt at this entire question at the start of your tutorial (for on-campus classes), or submit it online before your tutorial begins (for online classes).

1. *(mostly from FIT2014 Final Exam, 2018)*
The predicate `supervised` has two arguments, both of which are people. The meaning of `supervised`(X, Y) is that person X supervised the PhD of person Y.
Express each of the following three sentences in predicate logic.

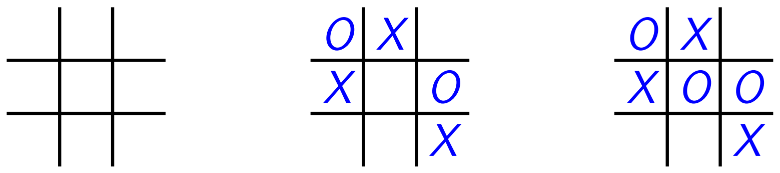
(a) Alonzo Church was the PhD supervisor of Alan Turing.

(b) No person supervised their own PhD.

(c) Not every PhD supervisor had a PhD supervisor.
Note. This one must be written as an *existential statement*.

(d) Alan Turing supervised only one person's PhD.

2. This question is about the game *Noughts-and-Crosses* (known as *Tic-Tac-Toe* in the US). This game is played using a 3×3 grid, usually drawn in the manner of Figure 1(a). Two players, *Crosses* and *Noughts*, each take turns to place X and O, respectively, in one cell of the grid. Once a cell is occupied by one player, its entry cannot be changed, and neither player can play there again. A player wins when they have three of their symbols in a line, horizontally, vertically, or diagonally, there being eight possible lines altogether; when that happens, the game stops. If all cells are occupied (five by Crosses and four by Noughts) and none of the eight three-cell lines have three identical symbols, then the game stops and is a Draw. (For simplicity, we forbid resignations and agreed draws.)



(a) (b) (c)

Figure 1: Some positions in Noughts-and-Crosses

The variable P always stands for a position in Noughts-and-Crosses, which represents the state of the array after the players have played some number of moves. A position thus corresponds to a 3×3 array with some subset of the cells occupied, in which the number of Crosses either equals, or exceeds by one, the number of Noughts. In the former case, then the next player to move must be Crosses; in the latter case, the next player to move must be Noughts. The diagram in Figure 1(a) shows the position before anyone has moved; the next player to move is Crosses. The diagram in Figure 1(b) shows a possible position after Crosses has had three turns and Noughts has had two turns; the next player to move is Noughts.

A move can be specified by naming the player whose turn it is and specifying the cell into which they place their symbol. For example, from the position of Figure 1(b), the move "Noughts: centre cell" gives the position in Figure 1(c).

A *winning move* is a move by a player that wins immediately, i.e., that completes the first line of three identical symbols seen in the game so far.

We use the following variables, predicates and function:

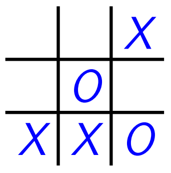
- The predicate CrossesWins(P) is True if, in position P, there is a line of three Crosses and no line of three Noughts.
- The predicate NoughtsWins(P) is True if, in position P, there is a line of three Noughts and no line of three Crosses.
- The predicate CrossesToMove(P) is True if, in position P, it is the turn of Crosses to move (in other words, the numbers of Crosses and Noughts are equal, and no-one has won yet).
- The predicate NoughtsToMove(P) is True if, in position P, it is the turn of Noughts to move (in other words, the number of Crosses is one greater than the number of Noughts, and no-one has won yet).
- The function ResultingPosition($P, X_1, X_2, \ldots, X_k$) returns the position produced by starting with position P and then playing moves $X_1, X_2, \ldots, X_k$ (where $1 \le k \le 9$). For example, if P denotes the position in Figure 1(b), then ResultingPosition(P,"Noughts: centre cell") gives the position in Figure 1(c). If a move X_i is illegal — because it is out of turn, or in a cell that is already occupied — then it has no effect and leaves the position unchanged.

Using quantifiers and the variables, predicates and function described above, write statements in predicate logic with each of the following meanings.

(a) Crosses has a winning move in position P.

(b) Noughts has a winning move in position P.

(c) For each of the statements you wrote for (a) and (b), determine whether the statement is True or False when P is the following position.



By analogy with the definition of "winning move" given above, we could define a *losing move* to be a move that gives a position that is a win for the opponent, i.e., where there is a line of three of the opponent's symbols that did not exist in the game before. In Noughts-and-Crosses, this never happens. (Why?)

(d) Write a predicate logic statement to say that losing moves are impossible.

Now write predicate logic statements with each of the following meanings, where again P can be any Noughts-and-Crosses position, and P_0 is the initial position (when the 3×3 grid is empty: see Figure 1(a)).

(e) Crosses has a strategy for winning within three moves from position P (where the three moves are one by Crosses, one by Noughts, and another by Crosses).

(f) Crosses does not have a strategy for winning within three moves from position P.
For this question, you must ensure that no logical negation occurs immediately in front of any quantifier.

(g) Crosses has a winning strategy from the initial position P_0.

(h) Noughts has a winning strategy from the initial position P_0.

(i) With best possible play from both sides from the initial position P_0, the game ends in a Draw.

(j) It is possible for Noughts to win from the initial position P_0.

3. A language L is called **hereditary** if it has the following property:

For every nonempty string x in L, there is a character in x which can be deleted from x to give another string in L.

Prove by contradiction that every nonempty hereditary language contains the empty string.

4. Prove the following statement, by mathematical induction:

(*) The sum of the first k odd numbers equals k^2.

(a) First, give a simple expression for the k-th odd number.
(b) Inductive basis: now prove the statement $(*)$ for $k = 1$.

Assume the statement $(*)$ true for a specific value k. This is our **Inductive Hypothesis**.

(c) Express the sum of the first $k + 1$ odd numbers ...

$$1 + 3 + \cdots + ((k + 1)\text{-th odd number})$$

... in terms of the sum of the first k odd numbers, plus something else.
(d) Use the inductive hypothesis to replace the sum of the first k odd numbers by something else.

(e) Now simplify your expression. What do you notice?

(f) When drawing your final conclusion, don't forget to briefly state that you are using the Principle of Mathematical Induction!

5. Prove, by induction on n, that for all $n \geq 1$, every tree on n vertices has $n-1$ edges.

Use the fact that every tree has a leaf, except for the trivial tree with one vertex and no edge. But it's also interesting to try to *prove* this fact.

6. (a) Prove, by induction on n, that for all $n \geq 3$,

$$n! \leq (n-1)^n.$$

(b) [Challenge]
Can you use a similar proof to show that $n! \leq (n-2)^n$? What assumptions do you need to make about n? How far can you push this: what if 2 is replaced by a larger number? What is the best upper bound of the form $f(n)^n$ that you can find, where $f(n)$ is some function of n?

Supplementary exercises

7. Let P_n be the proposition $x_1 \wedge x_2 \wedge \cdots \wedge x_n$. Note that P_1 consists just of x_1, and P_n is equivalent to $P_{n-1} \wedge x_n$.

Prove by induction on n that, if $x_1 = \mathrm{F}$, then P_n is False.

8. This question is based on Lab 0, Section 8, Exercise 1.

Let *program* be any program that can be run in Linux and produces standard output. Suppose we do *program* | `wc` as above, followed by a sequence of further applications of | `wc`.

```
$ program | wc
    ...
$ program | wc | wc
    ...
$ program | wc | wc | wc
    ...
⋮
```

(a) Determine how many pipes are required before the output ceases to change, and what that output will be.

(b) Prove by induction that, whatever *program* is (as long as it produces some standard output), continued application of | `wc` eventually produces this same fixed output.

This is probably best done by proving, by induction on n, that if | `wc` is applied repeatedly to a file of $\leq n$ characters, it will eventually produce the fixed output you found in part (a).