

Unit D1 Supplement, 4/8/05

Sample Response to Exercise D1 Part 1: Big- O Notation

This exercise is intended to serve not only as a review of algorithm analysis but also as a review of logic. The use of game-theoretic interpretation often reveals whether one understands the semantic aspect of logic (which is more important than the syntactic aspect, in my opinion). If one can associate quantifiers with the operations of the graphing tool, s/he clearly demonstrates how to deal with quantifiers. I realized that most of you need to see how to do it. So, here is how I would do.

Definition: ‘ O ’ (asymptotic upper bound) is defined as follows: $f(n) \in O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c g(n)$ for every integer $n \geq n_0$.

The definition can be translated into the following first-order logic (FOL) statement (use of ‘ \wedge ’ for ‘and’):

$$\exists c \exists n_0 \forall n [(c > 0) \wedge (n_0 \geq 1) \wedge (n \geq n_0) \wedge (f(n) \leq c g(n))]$$

which can also be written as: $\exists c > 0 \exists n_0 \geq 1 \forall n \geq n_0 [f(n) \leq c g(n)]$. As for the ordering of the quantifier, $\exists n_0$ and $\forall n$ must be placed in that order because the choice of n depends on that of n_0 . The placement of $\exists c$ is flexible because only the main relation, $f(n) \leq c g(n)$, depends on it.

Advanced notes (can be skipped): Although it would also be possible to quantify the functions as follows:

$$\forall f \forall g \exists c \exists n_0 \forall n [(c > 0) \wedge (n_0 \geq 1) \wedge (n \geq n_0) \wedge (f(n) \leq c g(n))]$$

such a statement is beyond the ability of FOL because functions are “higher-order” and not individuals.

The general procedure would be as follows. First, assume that the two functions are given. Then, define a strategy for each quantifier.

1. Verifier: Choose $c > 0$ by adjusting the slide bar so that the statement would hold.
2. Verifier: Choose $n_0 \geq 1$ by finding the value by adjusting the scale(s) so that the statement would hold.
3. Falsifier: Choose $n \geq n_0$ by finding the value by adjusting the scale(s) so that the statement would not hold.

If the falsifier can find such a value, the falsifier wins, i.e., the statement does not hold. Otherwise, the verifier wins, i.e., the statement holds.

Example 1: $n^4 \in O(2^n)$

- Set $f(n) = n^4$, $g(n) = 2^n$. Draw the graph with x -axis: 10^1 , y -axis: 10^3 (default); $f(n) = n^4$ looks like growing faster.

- The verifier keeps the default $c = 1$, but searches for a n_0 by adjusting x -axis to 10^2 . Since the values go beyond the current range, s/he increases the y -axis to 10^6 . Since $g(n) = 2^n$ appears to exceed $f(n) = n^4$ after $n = 16$ (or so), the verifier sets the n_0 to 20 (to be safe).
- The falsifier tries to find an $n \geq n_0$ at which the situation reverses by extending the x -axis (and y -axis to increase the range). However, s/he fails to find such an n . The falsifier lost. Thus, $n^4 \in O(2^n)$.

Example 1: $2^n \notin O(n^2)$

- Set $f(n) = 2^n$, $g(n) = n^2$. Draw the graph with x -axis: 10^1 , y -axis: 10^3 (default); $f(n) = 2^n$ looks like growing faster.
- The verifier tries to raise $c = 10$. Then, $g(n) = n^2$ appears to be no less than $f(n) = 2^n$, within the window. Just to be make $c g(n)$ larger, s/he raises c to 10,000. S/he stays with $n_0 = 1$.
- But then, the falsifier can adjust the scale to observe that $f(n) = 2^n$ exceed $g(n) = n^2$ at some point. Although this step depends on the verifier's choice of c and n_0 , the falsifier seems to be able to find a point where $f(n) = 2^n$ exceed $g(n) = n^2$, at least within the operations of the graphing tools. The falsifier won. Thus, $2^n \notin O(n^2)$.

// End