The Logic in Computer Science Column

by

Yuri Gurevich

Microsoft Research
One Microsoft Way, Redmond WA 98052, USA
gurevich@microsoft.com

A General Definition of the O-notation for Algorithm Analysis

Kalle Rutanen[1,3], Germán Gómez-Herrero[2], Sirkka-Liisa Eriksson[1], and Karen Egiazarian[3]

[1]Department of Mathematics, Tampere University of Technology
[2]Innovative Travel Ltd
[3]Department of Signal Processing, Tampere University of Technology

Abstract

We provide an extensive list of desirable properties for an O-notation — as used in algorithm analysis — and reduce them to 8 primitive properties. We prove that the primitive properties are equivalent to the definition of the O-notation as linear dominance.

Contents

1 Introduction

Algorithm analysis is concerned with the correctness and complexity of algorithms. *Correctness analysis* is about verifying that an algorithm solves the problem it is claimed to solve, and *complexity analysis* is about counting the amount of resources an algorithm takes. The resource can be time, memory, operations, or comparisons — or anything else that can be quantified with a non-negative real number. This paper is concerned with algorithm complexity.

Here is an intuitive description of this paper. The resource-consumption of an algorithm is captured by a function which maps each input of the algorithm to a non-negative real number. The resource-consumptions are ordered by an order-relation $\preceq$, which tells for resource-consumptions f and g whether $f \preceq g$, $g \preceq f$, both — then f and g are equivalent — or neither. The relation $\preceq$ makes it possible to *simplify*, and to *compare* resource-consumptions. A resource-consumption can be simplified by replacing it with an equivalent, simpler resource-consumption. The "smaller" the resource-consumption is according to $\preceq$, the "better" the algorithm — at least for the measured resource.

The O-notation $O(f)$ is the set of functions g which satisfy $g \preceq f$. It contains the same simplification and comparison tools in a slightly different, but equivalent form.

How should the O-notation be defined? By the above, we may ask an equivalent question: how should the order relation $\preceq$ be defined? We provide 8 intuitive properties for $\preceq$, and then show that there is exactly one definition of $\preceq$ which satisfies these properties.

The rest of this paper formalizes these intuitive ideas. We begin by defining the notation.

1.1 Notation

We will assume the Zermelo-Fraenkel set-theory with the axiom of choice, abbreviated ZFC. The set of natural numbers, integers, and real numbers are denoted by $\mathbb{N} = \{0, 1, 2, ...\}$, $\mathbb{Z}$, and $\mathbb{R}$, respectively. Let X be a set. The set of subsets of X is denoted by $\mathcal{P}(X)$. A set $C \subseteq \mathcal{P}(X)$ is a *cover* of X, if $\bigcup C = X$. Let $\sim \subseteq X^2$ be a relation. We define $x \sim y :\iff (x, y) \in \sim$, for all $x, y \in X$, and

$$X^{\sim y} := \{x \in X : x \sim y\}, \tag{1}$$

for all $y \in X$.

Example 1.1. $\mathbb{R}^{\geq 0}$ is the set of non-negative real numbers.

Let Y be a set. The set of functions from X to Y is denoted by $X \to Y$, or alternatively by Y^X. We are specifically interested in non-negative real-valued functions; we define

$$R_X := \left(X \to \mathbb{R}^{\geq 0}\right), \tag{2}$$

where X is a set. A relation $\sim \subseteq Y^2$ is extended to functions $f, g : X \to Y$ by

$$f \sim g :\iff \forall x \in X : f(x) \sim g(x). \tag{3}$$

Example 1.2. Let $f \in R_X$. Then $f \geq 0$.

Example 1.3. Let $x, y \in \mathbb{R}^d$, where $d \in \mathbb{N}^{>0}$. Then

$$x \geq y \iff \forall i \in d : x_i \geq y_i. \tag{4}$$

We extend a binary function $\oplus : R_X \times R_X \to R_Y$ to $\mathcal{P}(R_X) \times \mathcal{P}(R_X) \to \mathcal{P}(R_Y)$ by $U \oplus V = \{u \oplus v\}_{(u,v) \in U \times V}$. Similarly, we extend a unary function $\overline{\cdot} : R_X \to R_Y$ to $\mathcal{P}(R_X) \to \mathcal{P}(R_Y)$ by $\overline{U} = \{\overline{u}\}_{u \in U}$.

If f is a function, we use f^{-1} to denote both the inverse $Y \to X$ and the preimage $\mathcal{P}(Y) \to \mathcal{P}(X)$ under f; context should make the intent clear. The *composition* of $g : Y \to Z$ and $f : X \to Y$ is $(g \circ f) : X \to Z$ such that $(g \circ f)(x) = g(f(x))$. The *restriction* of $f : X \to Y$ to $D \subseteq X$ is $(f|D) : D \to Y$ such that $(f|D)(x) := f(x)$.

A *class of sets* is an unambiguous collection of sets, which may or may not be a set itself. A *proper class* is a class of sets which is not a set.

Example 1.4 (Classes of sets). Every set is a class of sets. The collection of all sets in ZFC is a proper class.

Remark 1.5 (Proper classes). The proper classes are not formalizable in the ZFC set-theory. This is not a problem for two reasons. First, everything in this paper can be carried through without forming proper classes, by only referring to the class elements and their relationships. Alternatively, we may adopt the von Neumann-Bernays-Gödel set theory [1] — a conservative extension of ZFC which formalizes proper classes. △

A *universe* is a class U of sets such that $\mathbb{N} \in U$, $\forall X \in U : \mathcal{P}(X) \subseteq U$, and $\forall X, Y \in U : X \times Y \in U$. If U is a universe, then a *sub-universe of* U is a subclass $V \subseteq U$ which is also a universe. If X is a set such that $\mathbb{N} \subseteq X$, then the *universe generated by* X is the set

$$U_X = \bigcup_{d \in \mathbb{N}^{>0}} \mathcal{P}\left(X^d\right). \tag{5}$$

Example 1.6 (Examples of universes). The smallest universe is given by $U_{\mathbb{N}}$; every universe contains this set as a sub-universe. The class of all sets is a universe which is a proper class.

An *O-notation over the universe U* is a class of functions

$$O := \{O_X : R_X \to \mathcal{P}(R_X)\}_{X \in U}. \tag{6}$$

When the underlying universe is not relevant, we will use the term O-notation. The O is used as a generic symbol for studying how the different desirable properties of O interact together. We will use accents for specific versions of the O-notation, such as $\widehat{O}$ for asymptotic linear dominance.

A *computational problem* is a function $P : X \to Y$, where $X, Y \in U$. The set of algorithms[1] which solve P, under a given model of computation, is denoted by $\mathcal{A}(P)$. The set of all algorithms from X to Y is

$$X \twoheadrightarrow Y = \bigcup_{P \in (X \to Y)} \mathcal{A}(P). \tag{7}$$

The *composition* of algorithms $G : Y \twoheadrightarrow Z$ and $F : X \twoheadrightarrow Y$ is the algorithm $G \circ F : X \twoheadrightarrow Z$, which is obtained by using the output of F as the input of G. We will sometimes use an algorithm $F : X \twoheadrightarrow Y$ as if it were its induced function instead. The resource-consumption of an algorithm $F \in \mathcal{A}(P)$ is denoted by $\mathcal{R}(F) \in R_X$.

1.2 Algorithms

What is an algorithm? We adopt an extremely liberal, but completely formalized view: an *algorithm* is an abstract state machine [2] [3] [4].

A variable in an abstract state machine M is identified with a string, called a *(function) symbol*. Each symbol has an *arity* $n \in \mathbb{N}$, which gives the number of arguments the symbol accepts as input. A *(ground) term* is defined recursively as follows:

- a 0-ary symbol is a term, and
- if f is an n-ary symbol, and $t_1, \dots, t_n$ are terms, then the string $f(t_1, \dots, t_n)$ is a term, and
- there are no other terms.

The set of user-defined symbols, together with a small set of predefined symbols — such as true, false, or, and, not, =, undef — is called the *vocabulary* of the abstract state machine M.

Each n-ary symbol f is associated with a function[2] $\overline{f} : S^n \to S$. The $\overline{f}$ is an *interpretation* of f. The set S is the *base-set*, which is common to all interpretations. The *value* of a term t, denoted by $[t]$, is defined recursively as follows:

[1]We will define the term *algorithm* formally in Section 1.2.

[2]$S^0 = \{\emptyset\}$.

- if x is a 0-ary symbol, then $[x] = \bar{x}(\emptyset)$,
- if f is an n-ary symbol, for $n \in \mathbb{N}^{>0}$, and $t_1, \dots, t_n$ are terms, then

$$[f(t_1, \dots, t_n)] = \bar{f}([t_1], \dots, [t_n]).$$

As the abstract machine steps through an algorithm, the interpretation of a symbol changes with each write to its memory. In the following, by $f : X \to Y$ — where f is a symbol, $X \subseteq S^n$, and $Y \subseteq S$ — we mean that $\bar{f}(X) \subseteq Y$. In addition, if x is a 0-ary symbol, then by $x \in X$ we mean that $x : \{\emptyset\} \to X$ in the previous sense.

Compared to an ordinary programming language, a 0-ary symbol corresponds to a variable, while an n-ary symbol, for $n \in \mathbb{N}^{>0}$, corresponds to an n-dimensional array — however, here the index can be an arbitrary set element.

The abstract machine specifies how the interpretations of symbols are to be modified at each step. The program driving the abstract machine is a finite sequence of conditional assignments of the form

```
**if** condition **then**
    $t_1 := s_1$
    $\vdots$
    $t_n := s_n$
**end if**
```

where condition, t_1, ..., t_n, and s_1, ..., s_n are terms. The formula $t_i := s_i$ can be thought of as copying an element from an array to another, $[t_i] := [s_i]$, provided $[\text{condition}] = [\mathit{true}]$. This sequence of assignments is repeated until (possible) termination.

All of the assignments in a single step are carried out in parallel — not in sequence. For example, $t_1 := s_1$ followed by $s_1 := t_1$ causes t_1 and s_1 to swap values in the next step.

The basic definition of abstract state machines is both simple, and extremely general. The generality derives from the virtue of making the whole of set-theory available for modeling variables. Further abstraction-tools — such as sequential composition, sub-machine calls, local variables, and return values — are constructed over this basic definition. For example, the Turbo-ASMs in [4] provide such features. From now on, we will assume that such abstraction tools have already been defined.

Consider Algorithm 1, which is Newton's method for finding local zeros of differentiable functions[3]. The input-symbols to this algorithm are a 1-ary continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$, a 0-ary initial guess $x^* \in \mathbb{R}$, and a 0-ary

[3]This example generalizes an example from [5], where Newton's method is formalized as an algorithm for real rational functions under the real-RAM model .

error threshold $\varepsilon \in \mathbb{R}^{\geq 0}$; the input-set is $C_1(\mathbb{R} \to \mathbb{R}) \times \mathbb{R} \times \mathbb{R}^{\geq 0}$. The output — provided the algorithm terminates — is a 0-ary point $x \in \mathbb{R}$ such that $|f(x)| \leq \varepsilon$; the output-set is $\mathbb{R}$. Other symbols are a 1-ary symbol $' : C_1(\mathbb{R} \to \mathbb{R}) \to C_1(\mathbb{R} \to \mathbb{R})$ — differentation —, a 1-ary symbol $|\cdot| : \mathbb{R} \to \mathbb{R}^{\geq 0}$ — absolute value — a 2-ary symbol $> : \mathbb{R} \times \mathbb{R} \to \{0, 1\}$ — greater-than — and 2-ary symbols $- : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and $/ : \mathbb{R} \times \mathbb{R}^{\geq 0} \to \mathbb{R}$ — subtraction and division. We have used infix notation for subtraction, division, and greater-than; postfix notation for differentiation; and midfix notation for the absolute value.

Algorithm 1 Newton's method for finding an element $x \in X$, such that $|f(x)| \leq \varepsilon$, for a continuously differentiable function $f \in C_1(\mathbb{R} \to \mathbb{R})$.

```
1: procedure findZeroOrHang(f, $x^*$, $\varepsilon$)
2:     $x := x^*$
3:     while $|f(x)| > \varepsilon$ do
4:         $x := x - f(x)/f'(x)$
5:     end while
6:     return $x$
7: end procedure
```

Algorithm 1 reads like pseudo-code, but is a completely formalized abstract state machine. With abstract state machines, the programmer is free to use the most fitting abstraction for the problem at hand. It should be clear how using such an abstract programming language enhances communication between software engineers and domain experts (e.g. physicists).

Termination is not required for an algorithm; consider for example an operating system. Depending on the input, Algorithm 1 may terminate, or not. Suppose we require Algorithm 1 to always terminate. To satisfy this requirement, the programmer can either restrict the input-set to terminating inputs, or to modify the algorithm — perhaps by limiting the number of iterations. From now on, we assume every analyzed algorithm to terminate.

A software development project utilizing abstract state machines starts by creating the most abstract description of the software as an abstract state machine, called the *ground model* [4]. This model captures the requirements, but does not provide any additional details on how the goals are to be attained. The project then proceeds to refine the model until it can be implemented in a concrete programming language. The model — in all stages — is used for verification and validation, and may even be used to generate code automatically.

This brief introduction to abstract state machines is to encourage the reader to look beyond the Church-Turing thesis, and to realize the usefulness of even the most abstract algorithms — not just those computable algorithms which work with natural numbers. These are algorithms which take arbitrary sets as input, and produce arbitrary sets as output. To analyze such abstract algorithms, we need correspondingly abstract tools[4].

1.3 Computational model and cost-model

Before an algorithm can be written, the writer must decide on the model of computation. A model of computation is a mathematical structure, and a set of atomic operations which manipulate that structure. Some models of computation are the Turing machine, the random-access machine (RAM) [7], the real-RAM [5], and the abstract state machine.

The result of complexity analysis — for a given model of computation, a given algorithm, and a given resource — is a function $f : X \to \mathbb{R}^{\geq 0}$, which provides for each *input* of the algorithm a non-negative real number. This number tells how much of that resource the algorithm consumes with the given input. As discussed in Section 1.2, the input-set X can be arbitrary.

Before a complexity analysis can be carried out, the analyst must decide on the cost-model. The cost-model specifies the amount of resources that running an atomic operation takes on a given input. A given computational model can assume different cost-models. When the cost-model is unspecified — as it often is — the cost of each atomic operation is assumed to be one unit irrespective of input.

Example 1.7 (Constant cost-models). The most common cost-model is the unit-cost model, which counts the number of performed atomic operations. Zero costs can be used to concentrate the interest to specific resources, such as order-comparisons.

Example 1.8 (Non-constant cost-models). An example of a non-constant cost-model is to assign the addition of natural numbers[5] a cost which is proportional to the logarithms of the arguments, so as to be proportional to the number of bits in their binary representations.

Example 1.9 (Cost-models for abstract state machines). An abstract state machine specifies costs for reading or writing a memory location through a given symbol. Reading an addition symbol $+$ at $(x, y) \in \mathbb{N}^2$ — $x + y$ — could be assigned a logarithmic cost as described above.

1.4 Primitive properties

Complexity analysis aims to classify and compare algorithms based on their resource consumptions; the less an algorithm uses resources to solve a given problem, the better it is compared to other algorithms which solve the same problem.

[4]We also note that algorithms in computational geometry are defined over subsets of $\mathbb{R}^d$ [6].

[5]Assuming the computational model supports such an operation.

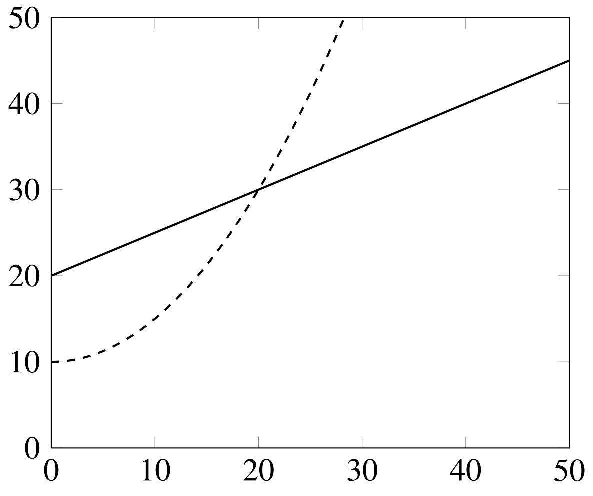

Figure 1: Here $f \in R_{\mathbb{N}}$ is such that $f(n) = n/2 + 20$ (solid line), and $g \in R_{\mathbb{N}}$ is such that $g(n) = n^2/20 + 10$ (dashed line). If interpreted as resource-consumptions, should $f \preceq_X g$, $g \preceq_X f$, $f \approx_X g$, or should f and g be incomparable?

The resource-consumptions of the algorithms to solve a problem $P : X \to Y$ are elements of R_X. The most general way to compare them is to define a preorder $\preceq_X \subseteq R_X \times R_X$. This relation should capture the intuitive concept of a resource-consumption $f \in R_X$ being either better or equivalent to the resource-consumption $g \in R_X$ — in some sense. For brevity, we use the term dominated by.

Remark 1.10 (Not worse). It is tempting to use the phrase *not worse than* instead of *better or equivalent*. However, the former means better, equivalent, *or incomparable*, which is not what we want. △

The most obvious fundamental *dominance* property is

- **order-consistency**; if $f \leq g$, then $f \preceq_X g$; if f does not use more resources than g for any input, then f is dominated by g,

for all $f, g \in R_X$. In particular, this implies that $\preceq_X$ is reflexive: $f \preceq_X f$; f is dominated by itself. Since $\preceq_X$ is a preorder, we have, by definition, the second fundamental *dominance* property:

- **transitivity**; $(f \preceq_X g$ and $g \preceq_X h) \implies f \preceq_X h$; if f is dominated by g, and g is dominated by h, then f is dominated by h,

for all $f, g, h \in R_X$. How should we define the other properties? If neither $f \leq g$, or $g \leq f$, can there still be cases where the other is obviously a better choice?

Consider Figure 1, where $f(n) > g(n)$ for $n \in [0, 20)_{\mathbb{N}}$, and $f(n) \leq g(n)$, for $n \in [20, \infty)_{\mathbb{N}}$. We would then be inclined to say that f has a better resource-consumption, since $f(n)$ is small anyway on the finite interval $n \in [0, 20)_{\mathbb{N}}$; f is "essentially" better than g. How can this intuition be formalized? There are many possibilities, some of which are:

- **local dominance**; there exists (an infinite interval / an infinite set / a cofinite set / a cobounded set / ...) $A \subseteq X$, such that $(f|A) \leq (g|A)$,
- **linear dominance**; there exists $c \in \mathbb{R}^{>0}$, such that $f \leq cg$,
- **translation dominance**; there exists $c \in \mathbb{R}^{>0}$, such that $f \leq g + c$,
- **combinations of the above**; local linear dominance $(f|A) \leq c(g|A)$, affine dominance $f \leq cg + c$, and local affine dominance $(f|A) \leq c(g|A) + c$.

All of these choices of $\preceq_X$ have order-consistency, and transitivity. But there is an infinity of other formalizations with these properties. For this reason, pondering about a plot such as Figure 1 does not help much.

A better approach is to reflect on the fundamental properties that the analyst needs to complete his/her complexity analysis. Suppose $\overline{f} \in R_X$, and $F : X \twoheadrightarrow Z$ has resource-consumption $f \in R_X$. The fundamental *structural* properties are:

- **locality**; if the complexity analysis of F is divided into a finite number of cases $A_1, \dots, A_n \subseteq X$ such that $\bigcup_{i=1}^{n} A_i = X$, and the analyses show that $(f|A_i) \preceq_{A_i} \left(\overline{f}|A_i\right)$, for all $i \in [1, n]_{\mathbb{N}}$, then $f \preceq_X \overline{f}$.
- **sub-composability**; if it has been shown that $f \preceq_X \overline{f} \in R_X$, and $s : Y \to X$, then $F \circ s$ has resource-consumption $f \circ s \preceq_Y \overline{f} \circ s$; the mapped upper-bound of resource-consumption is an upper-bound of mapped resource-consumption.
- $\mathbb{N}$**-sub-homogenuity**; if it has been shown that $f \preceq_X \overline{f} \in R_X$, then repeatedly running F in a loop $u \in R_X$ times, where $u(X) \subset \mathbb{N}$, has resource-consumption $uf \preceq_X u\overline{f}$; the repeated upper-bound of resource-consumption is an upper-bound of repeated resource-consumption.
- $\mathbb{N}^{>0}$**-cancellation**; if it has been shown that $uf \preceq_X u\overline{f} \in R_X$, where $u \in R_X$ is such that $u(X) \subset \mathbb{N}^{>0}$, then F has resource-consumption $f \preceq_X \overline{f}$.

These properties reveal that the preorders in different sets cannot be defined independently of each other; they are tightly connected by locality and sub-composability. Rather, the problem is to find a consistent class of preorders $\{\preceq_X\}_{X \in U}$, where U is a given universe.

Remark 1.11 (**$\mathbb{N}^{>0}$-cancellation**)**.** The $\mathbb{N}^{>0}$-cancellation reverts $\mathbb{N}$-sub-homogenuity, so that a resource-consumption of a repeated algorithm can be used to deduce the resource-consumption of the algorithm that was repeated. Under a given computational model, and a cost-model, all of the possible cases provided by $\mathbb{N}^{>0}$-cancellation may not occur in this way. However, it does not seem plausible to restrict $\mathbb{N}^{>0}$-cancellation individually based on the computational model and the cost-model; we prefer something that works without exceptions. $\triangle$

A problem with the listed properties thus far is that the trivial preorder $\preceq_X = R_X \times R_X$, for all $X \in U$, is able to fulfill them all; then the functions in R_X are all equivalent. Therefore, for the comparison to be useful, we need to require $\preceq_X$ to distinguish at least some functions in R_X, at least for some $X \in U$. The fundamental *non-triviality* property is:

- **one-separation**; $n \npreceq_{\mathbb{N}^{>0}} 1$.

Note that $1 \preceq_{\mathbb{N}^{>0}} n$ already holds by order-consistency; one-separation prevents the order from collapsing due to equivalence.

Finally, there is the question of robustness. Writing an algorithm is an iterative process, which causes the resource-consumption of an algorithm to change constantly. If every change is reflected in the ordering, then each change invalidates an existing complexity analysis of the algorithm. Worse, the change invalidates all the analyses of the algorithms which use the changed algorithm as a sub-algorithm. Since an algorithm may face hundreds or thousands of changes before stabilising into something usable in practice, such an approach is infeasible. Therefore, the ordering needs to introduce identification, to make it robust against small changes in the algorithms. But what is a small change?

The key realization is that for a given algorithm $F \in \mathcal{A}(P)$ it is often easy, with small changes, to produce an algorithm $G \in \mathcal{A}(P)$, for which the improvement-ratio $\mathcal{R}(F)/\mathcal{R}(G)$ stays bounded[6] [7]. In contrast, obtaining an unbounded improvement-ratio often requires considerable insight and fundamental changes to the way an algorithm works — a new way of structuring data and making use of it. This definition of interesting provides the desired robustness against small changes in algorithms. Correspondingly, the fundamental *abstraction* property is:

- **scale-invariance**; if $f \preceq_X \bar{f}$, then $f \preceq_X \alpha\bar{f}$, for all $\alpha \in \mathbb{R}^{>0}$.

We will show that the listed 8 fundamental properties, which we call *primitive*, determine a class of preorders $\{\preceq_X\}_{X \in U}$ uniquely.

A preorder $\preceq_X$ induces an equivalence relation $\approx_X \subseteq R_X \times R_X$, defined by $f \approx_X g \iff f \preceq_X g$ and $g \preceq_X f$. We may then define a strict comparison

[6]Assuming $\mathcal{R}(F) > 0$.

[7]For example, change to use Single Instruction Multiple Data (SIMD) instructions

$\prec_X \subseteq R_X \times R_X$ such that $f \prec_X g \iff f \preceq_X g$ and $f \not\approx_X g$. While we are at it, let $\succ_X \subseteq R_X \times R_X$ be such that $f \succ_X g \iff g \prec_X f$, and $\succeq_X \subseteq R_X \times R_X$ be such that $f \succeq_X g \iff g \preceq_X f$.

1.5 Problem complexity

Apart from analyzing the resource-consumption of a specific algorithm, complexity analysts also have a greater underlying goal: that of analyzing the resource-consumption of the underlying problem $P : X \to Y$ itself — for a given computational model, a given cost-model, and a given resource. This activity divides into two sub-activities.

First, one wishes to find a lower-bound for the resource-consumption of P. A resource-consumption $f \in R_X$ is a *lower-bound* for P, if $f \preceq_X \mathcal{R}(A)$, for all $A \in \mathcal{A}(P)$. A resource-consumption $f \in R_X$ is *optimal* for P, if f is a lower-bound for P, and $g \preceq_X f$ for each $g \in R_X$ a lower-bound for P. The ultimate goal of lower-bound analysis is to find the optimal resource-consumption for P.

Second, one wishes to find an upper-bound for the resource-consumption of P. A resource-consumption $f \in R_X$ is an *upper-bound* for P, if $\mathcal{R}(A) \preceq_X f$, for some $A \in \mathcal{A}(P)$. This is done by finding an actual algorithm for solving P. While there are computational problems which cannot be solved at all[8], establishing at least one upper-bound for a solvable problem is often easy. These are the *brute-force* algorithms, which compute or check everything without making any use of the underlying structure in the problem.

An algorithm $A \in \mathcal{A}(P)$ is *optimal*, if $\mathcal{R}(A)$ is optimal for P. The ultimate goal of upper-bound analysis is to find an optimal algorithm $A \in \mathcal{A}(P)$ for solving P.

An optimal resource-consumption may not exist for P, as shown in Theorem 4.24. In this case the lower- and upper-bound analyses never meet their goals.

When it exists, the optimal resource-consumption for P, $\inf_{\preceq_X} \mathcal{R}(\mathcal{A}(P))$, is unique up to equivalence. Even then there may not exist an optimal algorithm for P. In this case the lower-bound analysis may meet its goal, but the upper-bound analysis does not. However, optimality may not be that important, if the upper-bound can be brought close to the optimal lower-bound.

Remark 1.12 (Optimality). Why is it that an optimal algorithm for a practical problem often seems to exist? Are there existing problems for which there is no optimal algorithm, or not even an optimal resource-consumption? Could matrix multiplication under the unit-cost real-RAM be an example of one or both? These questions are open. △

[8]e.g. the halting problem under Turing machines.

1.6 O-notation

An *O-notation in a set X* is a function $O_X : R_X \to \mathcal{P}(R_X)$, such that

$$O_X(f) = \{g \in R_X : g \preceq_X f\}. \tag{8}$$

An *O-notation* is the class $\{O_X\}_{X \in U}$, where U is a given universe.

It is equivalent to define either the functions $\{O_X\}_{X \in U}$, or the preorders $\{\preceq_X\}_{X \in U}$; one can be recovered from the other. We shall give the theorems in terms of $\{O_X\}_{X \in U}$, since this is more familiar to computer scientists. However, we find that intuition works better when working with $\{\preceq_X\}_{X \in U}$.

Apart from the primitive properties, there are various other desirable properties for an O-notation, which are summarized in Table 1. It can be shown that they are all implied by the primitive properties. We skip their proofs due to space-constraints, save for those needed to prove the necessity of linear dominance. Each property is given for O_X, where $X \in U$. A given property holds for O if it holds for O_X for all $X \in U$.

We define the related notations $\Omega_X, \omega_X, o_X, \Theta_X : R_X \to \mathcal{P}(R_X)$ as follows:

$$\begin{aligned}
\Omega_X(f) &= \{g \in R_X : g \succeq_X f\}, \\
\omega_X(f) &= \{g \in R_X : g \succ_X f\}, \\
o_X(f) &= \{g \in R_X : g \prec_X f\}, \\
\Theta_X(f) &= \{g \in R_X : g \approx_X f\}.
\end{aligned} \tag{9}$$

Remark 1.13 (Definition of o_X). The way o_X is used, it is probably agreed that

$$g \in o_X(f) \implies g \in O_X(f) \text{ and } g \not\approx_X f, \tag{10}$$

for all $f, g \in R_X$. Here we also affirm the converse:

$$g \in o_X(f) \impliedby g \in O_X(f) \text{ and } g \not\approx_X f, \tag{11}$$

for all $f, g \in R_X$. Missing this latter property — because of using some other definition — would make it impossible to transfer the analyses done in terms of O and Θ to the analyses done in terms of o. The related notations must reflect different viewpoints of the same invariant concept — namely of the underlying preorder. △

Remark 1.14 (Study of the O-notation suffices). It suffices to study the O-notation, since the related notations are completely determined by the O-notation. △

In the following we will sometimes abbreviate a function in the parameter of O_X with an expression, such as in $O_{\mathbb{N}}(n)$, where we actually mean $O_{\mathbb{N}}(f)$, with $f \in$

Name	Property
order-consistency	$f \leq g \implies f \in O_X(g)$
reflexivity	$f \in O_X(f)$
transitivity	$(f \in O_X(g) \text{ and } g \in O_X(h)) \implies f \in O_X(h)$
the membership rule	$f \in O_X(g) \iff O_X(f) \subseteq O_X(g)$
zero-separation	$1 \notin O_{\mathbb{N}^{>0}}(0)$
one-separation	$n \notin O_{\mathbb{N}^{>0}}(1)$
zero-triviality	$O_X(0) = \{0\}$
scale-invariance	$O_X(\alpha f) = O_X(f)$
bounded translation-invariance	$(\exists \beta \in \mathbb{R}^{>0} : f \geq \beta) \implies O_X(f + \alpha) = O_X(f)$
power-homogenuity	$O_X(f)^\alpha = O_X(f^\alpha)$
additive consistency	$uO_X(f) + vO_X(f) = (u + v)O_X(f)$
multiplicative consistency	$O_X(f)^u O_X(f)^v = O_X(f)^{u+v}$
maximum consistency	$\max(O_X(f), O_X(f)) = O_X(f)$
locality	$(\forall D \in C : (f \mid D) \in O_D(g \mid D)) \implies f \in O_X(g)$
scalar homogenuity	$\alpha O_X(f) = O_X(\alpha f)$
$\mathbb{R}^{\geq 0}$-sub-homogenuity	$uO_X(f) \subseteq O_X(uf)$
$\mathbb{Q}^{\geq 0}$-sub-homogenuity	$uO_X(f) \subseteq O_X(uf) \quad (u(X) \subset \mathbb{Q}^{\geq 0})$
$\mathbb{N}$-sub-homogenuity	$uO_X(f) \subseteq O_X(uf) \quad (u(X) \subset \mathbb{N})$
$\mathbb{N}^{>0}$-cancellation	$uO_X(f) \subseteq O_X(uf) \quad (u(X) \subset 1/\mathbb{N}^{>0})$
super-homogenuity	$uO_X(f) \supset O_X(uf)$
sub-multiplicativity	$O_X(f)O_X(g) \subseteq O_X(fg)$
super-multiplicativity	$O_X(f)O_X(g) \supset O_X(fg)$
sub-restrictability	$(O_X(f) \mid D) \subseteq O_D(f \mid D)$
super-restrictability	$(O_X(f) \mid D) \supset O_D(f \mid D)$
additivity	$O_X(f) + O_X(g) = O_X(f + g)$
the summation rule	$O_X(f + g) = O_X(\max(f, g))$
the maximum rule	$\max(O_X(f), O_X(g)) = O_X(\max(f, g))$
the maximum-sum rule	$\max(O_X(f), O_X(g)) = O_X(f) + O_X(g)$
sub-composability	$O_X(f) \circ s \subseteq O_Y(f \circ s)$
injective super-composability	$O_X(f) \circ s \supset O_Y(f \circ s) \quad (s \text{ injective})$
the subset-sum rule	$\sum_{i \in [0, n(y))} a(y, i)\widehat{h}(S(y, i), i) \in O_Y\left(\sum_{i \in [0, n(y))} a(y, i)h(S(y, i), i)\right)$

Table 1: Desirable properties for an O-notation. Here $X, Y \in U$, $f, g, u, v \in R_X$, $\alpha \in \mathbb{R}^{>0}$, $D \subseteq X$, $s : Y \to X$, $n : Y \to \mathbb{N}$, $S : Y \times \mathbb{N} \to X$, $a : Y \times \mathbb{N} \to \mathbb{R}^{\geq 0}$, $B = \{(y, i) \in Y \times \mathbb{N} : i \in [0, n(y))\}$, $h \in R_B$, $\widehat{h} \in O_B(h)$, and $C \subseteq \mathcal{P}(X)$ is a finite cover of X. Primitive properties marked with a bold face.

$R_{\mathbb{N}} : f(n) = n$. When the expression contains multiple symbols, as in $O_{\mathbb{N}^2}\big(n^2 m\big)$, we interpret the symbols to be assigned to the input-tuple in alphabetical order, as in $O_{\mathbb{N}^2}\big((m, n) \mapsto n^2 m\big)$. This is to acknowledge that $(m, n) \mapsto n^2 m$ and $(n, m) \mapsto n^2 m$ are different functions.

1.7 Example analyses

In this section we provide some example analyses of algorithms, using several different definitions. As for our main result — the characterization of the O-notation — this section can be skipped.

Asymptotic linear dominance is defined by

$$g \in \widehat{O}_X(f) \iff \exists c \in \mathbb{R}^{>0}, y \in \mathbb{N}^d : \big(g|X^{\geq y}\big) \leq c\big(f|X^{\geq y}\big), \tag{12}$$

for all $f, g \in R_X$, and all $X \in U$, where $U = \bigcup_{d \in \mathbb{N}^{>0}} \mathcal{P}\big(\mathbb{N}^d\big)$. This definition is from [8] (second edition), from an exercise on page 50.

Coasymptotic linear dominance is defined by

$$g \in \widecheck{O}_X(f) \iff \exists c \in \mathbb{R}^{>0}, y \in \mathbb{N}^d : (g|(X \setminus X^{<y})) \leq c(f|(X \setminus X^{<y})). \tag{13}$$

This definition is from the third edition of the same book [9], from the same exercise on page 53. The books [8] and [9] mention that these definitions can be extended to $\mathbb{R}^d$ — by replacing $y \in \mathbb{N}^d$ with $y \in \mathbb{R}^d$ — but refer to doing this as an abuse.

Cofinite linear dominance is defined by

$$g \in \acute{O}_X(f) \iff \exists c \in \mathbb{R}^{>0}, A \in \overline{\mathcal{P}}(X) : (g|A) \leq c(f|A), \tag{14}$$

for all $f, g \in R_X$, and all sets X, where $\overline{\mathcal{P}}(X)$ is the set of subsets of X with finite complement. We are not aware of any book using this definition.

Linear dominance is defined by

$$g \in \bar{O}_X(f) \iff \exists c \in \mathbb{R}^{>0} : g \leq cf, \tag{15}$$

for all $f, g \in R_X$, and all sets X. This is the definition we prove to be equivalent to the primitive properties.

Remark 1.15 (A computational model for examples). The examples are to be interpreted as abstract state machines — appropriately enriched to support structured programming. The symbol $+$ is interpreted as addition in the set of its operands. Similarly for the other operators. When analyzing the time-complexity of the example-algorithms, we will assume that each algorithm takes one unit of time for initialization, and that calling an algorithm takes no time. A simple step — such as an increment, an assignment, or a comparison — takes one unit of time. A for-loop which iterates $n \in \mathbb{N}$ times checks its condition $n + 1$ times. $\triangle$

Algorithm 2 An algorithm which takes as input $(m, n) \in \mathbb{N}^2$, and returns $n(1 - \text{sgn}(m))$.

```
procedure computeOnPlane($m, n$)
    $j := 0$
    if $m = 0$ then
        for $i \in [0, n)$ do
            $j := j + 1$
        end for
    end if
    return $j$
end procedure
```

Algorithm 3 An algorithm which takes as input $n \in \mathbb{N}$, and returns n.

```
procedure mapNaturalsToPlane($n$)
    return computeOnPlane($0, n$)
end procedure
```

Example 1.16 (Using a strip in $\mathbb{N}^2$). Consider Algorithm 2, which takes two input parameters $m, n \in \mathbb{N}$. When $m \neq 0$, this algorithm takes $1 + 1 + 1 + 1 = 4$ operations. When $m = 0$, this algorithm takes $1+1+1+(3n+1)+1 = (3n+1)+4$ operations. That is, $(3n + 1)(1 - \text{sgn}(m)) + 4$ operations. By the definitions, Algorithm 2 has time complexity $\widehat{O}_{\mathbb{N}^2}(1)$, and $O_{\mathbb{N}^2}(n(1 - \text{sgn}(m)) + 1)$ for the other definitions.

Consider Algorithm 3, which takes a single input parameter $n \in \mathbb{N}$, and then makes a single call to Algorithm 2 after mapping the input n through $s : \mathbb{N} \to \mathbb{N}^2$ such that $s(n) = (0, n)$. By injective sub-composability, the resource-consumption of the called sub-algorithm is an element of $\widehat{O}_{\mathbb{N}}(1)$, and $O_{\mathbb{N}}(n + 1)$ for the other definitions.

If the call to Algorithm 2 in Algorithm 3 is replaced with the algorithm body itself, then the algorithm takes $(3n + 1) + 2$ operations[9]. For all definitions, this is $O_{\mathbb{N}}(n + 1) \setminus O_{\mathbb{N}}(1)$.

We notice that for $\widehat{O}$ these two results are contradictory. Tracing back, we see that our assumption of injective sub-composability does not hold for $\widehat{O}$ under this specific s.

From now on we consider $\widehat{O}$ and $\widecheck{O}$ to be defined on $\mathbb{R}^d$, as discussed, to study them in their strongest forms.

[9]Initialization and return are elided; 2 operations are saved.

Algorithm 4 An algorithm which takes as input $z \in \mathbb{Z}$, and returns $\max(-z, 1)$.

```
procedure computeOnIntegers($z$)
    $j := 0$
    if $z < 0$ then
        for $i \in [0, -z)$ do
            $j := j + 1$
        end for
    end if
    return $j$
end procedure
```

Algorithm 5 An algorithm which takes as input $n \in \mathbb{N}$, and returns $\max(n, 1)$.

```
procedure mapNaturalsToIntegers($n$)
    return computeOnIntegers($-n$)
end procedure
```

Example 1.17 (Using negative numbers in $\mathbb{Z}$). Consider Algorithm 4, which takes one input parameter $z \in \mathbb{Z}$. When $z \geq 0$, this algorithm takes $1+1+1+1 = 4$ operations. When $z < 0$, this algorithm takes

$$1 + 1 + 1 + (3z + 1) + 1 = (3z + 1) + 4 \tag{16}$$

operations. By the definitions, Algorithm 4 has time-complexity $\widehat{O}_{\mathbb{Z}}(1)$, $\widetilde{O}_{\mathbb{Z}}(1)$, and $O_{\mathbb{Z}}(\max(-z, 1))$ for the other definitions.

Consider Algorithm 5, which takes a single input parameter $n \in \mathbb{N}$, and then makes a single call to Algorithm 4 after mapping the input n through $s : \mathbb{N} \to \mathbb{Z}$ such that $s(n) = -n$. By injective sub-composability, the resource-consumption of Algorithm 5 is an element of $\widehat{O}_{\mathbb{N}}(1)$, $\widetilde{O}_{\mathbb{N}}(1)$, and $O_{\mathbb{N}}(\max(n, 1))$ for the other definitions.

If the call to Algorithm 4 in Algorithm 5 is replaced with the algorithm body itself, then for all definitions, the resource-consumption of Algorithm 5 is an element of $O_{\mathbb{N}}(\max(n, 1)) \setminus O_{\mathbb{N}}(1)$.

We notice that for $\widehat{O}$ and $\widetilde{O}$ these two results are contradictory. Tracing back, we see that we our assumption of injective sub-composability does not hold for $\widehat{O}$ and $\widetilde{O}$ under this specific s.

In both examples, we wish to analyze the complexity of an algorithm, while treating a sub-algorithm as a black-box, whose complexity is given by an O-notation. This complexity is “imported” into the caller’s context by sub-composability. However, if the mapping function s is such that the O-notation does not satisfy sub-composability for s, then either the sub-algorithm has to be expanded into the call-site, or the analysis has to be stopped. Since neither of these is a satisfying solution, we instead require sub-composability to hold for all mapping functions.

What about cofinite linear dominance $\acute{O}$, which seems to work well? The following example provides a tricky case for sub-composability.

Example 1.18 ($\widehat{O}$, $\widetilde{O}$, and $\acute{O}$ fail sub-composability in $\mathbb{N}$). Let O be one of $\widehat{O}$, $\widetilde{O}$, or $\acute{O}$, and $s : \mathbb{N} \to \mathbb{N}$ be such that

$$s(n) := \begin{cases} 0, & n \in 2\mathbb{N}, \\ n, & n \in 2\mathbb{N} + 1. \end{cases} \tag{17}$$

Let $\widehat{f} : \mathbb{N} \to \mathbb{N}$ be such that $\widehat{f}(0) = 1$ and $f : \mathbb{N} \to \mathbb{N}$ be such that $f(0) = 0$ and $\widehat{f} \in O_{\mathbb{N}}(f)$. Then $\left(\widehat{f} \circ s\right) \notin O_{\mathbb{N}}(f \circ s)$. The problem is that $(f \circ s)(2k) = 0$ cannot be multiplied to linearly dominate $(\widehat{f} \circ s)(2k)$, where $k \in \mathbb{N}$, and that the number of such points is not finite. Linear dominance $\overline{O}$ avoids this problem: from $\widehat{f}(0) = 1$ and $f(0) = 0$, it follows that $\widehat{f} \notin \overline{O}_{\mathbb{N}}(f)$.

It can be shown that $\widetilde{O}_X(f) = \acute{O}_X(f)$, for all $f \in R_X$ and $X \subset \mathbb{N}^d$. It can also be shown that injective sub-composability holds for cofinite linear dominance $\acute{O}$ in any set, and that sub-composability holds for *positive functions*. That is, $\acute{O}_X(f) \circ s \subset \acute{O}_Y(f \circ s)$, for all $f \in R_X$ such that $f > 0$, where X and Y are arbitrary sets. Therefore, the following complexity analyses are guaranteed to be correct:

- Those which assume coasymptotic linear dominance $\widetilde{O}$, remain in subsets of $\mathbb{N}^d$, and assume positive resource-consumptions.

- Those which assume cofinite linear dominance $\acute{O}$, and assume positive resource-consumptions.

To our knowledge, the first item covers the analyses in [9]. An analysis of a graph algorithm uses the domain $\mathbb{N}^2$ — corresponding to a worst-case / best-case / average-case analysis under the number of vertices and the number of edges.

Since zeros occur naturally in resource-consumptions (e.g. by concentrating on the number of comparisons), we would like the O-notation to also work in these cases. Therefore, the last step is to require sub-composability to hold for all non-negative resource-consumptions, which leads to linear dominance $\overline{O}$.

1.8 Implicit conventions

An *implicit convention* is an overload of notation adopted by people working in a given field. Since it is an overload, the reader is required to deduce the correct meaning of such notation from the context. Here are some implicit conventions related to the O-notation — as commonly used in computer science.

The *anonymous function convention* is to use $O_X(f)$ as a placeholder for an anonymous function $\widehat{f} \in O_X(f)$. It then must be guessed from the context whether the author means by $O_X(f)$ the whole set, or just an element of this set. For example, to say that the time-complexity of a given algorithm is $O_{\mathbb{N}}\left(n^2\right)$ means that the time-consumption function is an element of this set.

The *domain convention* is to leave off the domain of the O-notation, say $O\left(n^2\right)$, and then let the reader guess, for each use, the domain from the context. Here is an example where the domain convention leads to a difficult interpretation. Consider an algorithm [10] under the unit-cost w-bit RAM model, where $w \in \mathbb{N}$, which for $I \subseteq [0, 2^w)_{\mathbb{N}} = U$ finds a nearest neighbor of $i \in U$ in I in time $O(\log_2(\log_2(\Delta + 4)))$, where $\Delta \in \mathbb{N}$ is the distance between i and its nearest neighbor in I. Intuitively, this sounds reasonable, but what is the domain?[10] Our thinking process went as follows.

Since the expression contains only a single symbol Δ, we assumed it to be a univariate O-notation. Our first guess was $O_U(\log_2(\log_2(\Delta + 4)))$ — with w fixed. However, since U is bounded, this is equal to $O_U(1)$. The guess had to be wrong; otherwise the authors would have reported the complexity as $O(1)$.

Our second guess was $O_{\mathbb{N}}(\log_2(\log_2(\Delta + 4)))$ — again with w fixed. However, this arbitrarily extends the complexity analysis to elements outside the input-set, since $\Delta < 2^w$. In addition, it is not always possible to do such an extension, such as when the function is $\log_2(\log_2(2^w - \Delta + 4))$ instead.

Finally, we observed that the complexity depends both on w and Δ — although $O(\log_2(\log_2(\Delta + 4)))$ never mentions w. The correct formalization is given by $O_D(\log_2(\log_2(\Delta + 4)))$, where $D = \left\{(w, \Delta) \in \mathbb{N}^2 : \Delta \in [0, 2^w)\right\}$. The corresponding algorithm would then take as input $w \in \mathbb{N}^{>0}$, $I \in \cup_{k \in \mathbb{N}^{>0}} \mathcal{P}\left([0, 2^k)_{\mathbb{N}}\right)$, and $i \in \cup_{k \in \mathbb{N}^{>0}} [0, 2^k)$, subject to $I \subset [0, 2^w)_{\mathbb{N}}$ and $i \in [0, 2^w)$.

2 Previous work

In [11], Bachmann gave a rather brief and informal definition of the O-notation on page 401:

> ... wenn wir durch das Zeichen $O(n)$ eine Grösse ausdrücken, deren Ordnung in Bezug auf n die Ordnung von n nicht überschreitet; ...

[10]The inability to answer this question was the motivation for us to start studying the O-notation formally.

which we translate as: when we present some term with $O(n)$, its order does not surpass the order of n. In [12], Landau put this definition on a formal grounding on page 31 by

$$f \in O_{\mathbb{N}}(g) :\iff \exists c \in \mathbb{R}^{>0}, y \in \mathbb{N} : \forall x \geq y : f(x) \leq cg(x), \tag{18}$$

for all $f, g \in (\mathbb{N} \to \mathbb{R})$. On page 883, Landau credits this definition to [11].

In [13], Knuth defined O-notation as $f \in O_{\mathbb{N}}(g) :\iff \exists c \in \mathbb{R}^{>0} : f \leq cg$, for all $f, g \in (\mathbb{N} \to \mathbb{R})$. This seems to have been by accident; in [14], the definition was replaced with Equation 18. Knuth credited the definition to Bachmann [11].

In [12], Landau defined the o-notation on page 61 as

$$f \in o_{\mathbb{R}}(g) :\iff \lim_{x\to\infty} \frac{f(x)}{g(x)} = 0, \tag{19}$$

for all $f, g \in (\mathbb{R} \to \mathbb{R})$. On page 883, Landau states that the o-notation is his own.

In [15], Knuth defined the Ω-notation as $f \in \Omega_{\mathbb{N}}(g) :\iff g \in O_{\mathbb{N}}(f)$, the Θ-notation as $f \in \Theta_{\mathbb{N}}(g) :\iff f \in O_{\mathbb{N}}(g)$ and $g \in O_{\mathbb{N}}(f)$, and the ω-notation as $f \in \omega_{\mathbb{N}}(g) :\iff g \in o_{\mathbb{N}}(f)$, for all $f, g \in (\mathbb{N} \to \mathbb{R})$.

We are unaware of who first generalized the O-notation (and related notations) to the multi-variate case. The definition as asymptotic linear dominance, Equation (12), can be found from an exercise of [8], page 50. The definition as co-asymptotic linear dominance, Equation (13), can be found from an exercise of [9], page 53.

2.1 Howell's counterexample

In [16], Howell showed that asymptotic linear dominance $\widehat{O}$ does not satisfy the subset-sum rule. To be precise, Howell required the following properties from an O-notation:

- **asymptotic refinement**;

$$O_X(f) \subseteq \widehat{O}_X(f) \tag{20}$$

- **reflexivity**;

$$f \in O_X(f) \tag{21}$$

- **asymptotic order-consistency**;

$$\left(\exists y \in \mathbb{N}^d : \left(\widehat{f}|X^{\geq y}\right) \leq \left(f|X^{\geq y}\right)\right) \implies O_X\left(\widehat{f}\right) \subseteq O_X(f) \tag{22}$$

- **simplified subset-sum**;

$$\begin{aligned}\sum_{i=0}^{n_k} \widehat{g}(n_1, \dots, n_{k-1}, i, n_{k+1}, \dots, n_d) \in \\ O_X\left(\sum_{i=0}^{n_k} g(n_1, \dots, n_{k-1}, i, n_{k+1}, \dots, n_d)\right),\end{aligned} \tag{23}$$

where $X = \mathbb{N}^d$, $\widehat{f}, f, g \in R_X$, $\widehat{g} \in O_X(g)$, and $k \in [1, d] \subseteq \mathbb{N}$.

Let us also assume that O has scale-invariance[11]. While Howell did not explicitly assume this property, we claim that any sensible definition must satisfy it.

Theorem 2.1 (Howell's definition is asymptotic dominance). *O has asymptotic order-consistency, scale-invariance, reflexivity, and asymptotic refinement.* $\implies$ *$O = \widehat{O}$.*

Proof. By asymptotic order-consistency, scale-invariance, and reflexivity,

$$\begin{aligned} & \widehat{f} \in \widehat{O}_X(f) \\ \implies\quad & \exists c \in \mathbb{R}^{>0}, y \in \mathbb{N}^d : \left(\widehat{f}|X^{\geq y}\right) \leq c\left(f|X^{\geq y}\right) \\ \implies\quad & \exists c \in \mathbb{R}^{>0} : O_X\left(\widehat{f}\right) \subseteq O_X(cf) \\ \implies\quad & O_X\left(\widehat{f}\right) \subseteq O_X(f) \\ \implies\quad & \widehat{f} \in O_X(f), \end{aligned} \tag{24}$$

for all $f, \widehat{f} \in R_X$. Therefore $\widehat{O}_X(f) \subseteq O_X(f)$. It follows from asymptotic refinement that $O_X(f) = \widehat{O}_X(f)$. □

Therefore, while Howell's argument seems general, it only concerns the $\widehat{O}$-notation. Howell gave the following counterexample[12] to the subset-sum rule under $\widehat{O}$. Let $X = \mathbb{N}^2$, and $\widehat{g} \in R_X$ be such that

$$\widehat{g}(m, n) = \begin{cases} 2^n, & m = 0, \\ mn, & m > 0, \end{cases} \tag{25}$$

[11]Recall that scale-invariance is $O_X(\alpha f) = O_X(f)$, for all $f \in R_X$ and $\alpha \in \mathbb{R}^{>0}$.

[12]We have fixed the error of having the sum-index i run only to $m - 1$.

and $g \in R_X$ be such that $g(m, n) = mn$. Then

$$\begin{aligned}\sum_{i=0}^{m} \widehat{g}(i, n) &= 2^n + m(m+1)n/2 \\ &\notin \widehat{O}_X(m(m+1)n/2) \qquad (26) \\ &= \widehat{O}_X\left(\sum_{i=0}^{m} g(i, n)\right).\end{aligned}$$

This observation was the starting point for our paper.

3 Worst case, best case, average case

In this section we will formalize the concepts of worst-case, best-case, and average-case complexities.

Let $X, Y, Z \in U$. A *Z-grouping of X* is a surjective function $g : X \to Z$. A *case over g* is a function $s : Z \to X$ such that $g \circ s = \mathrm{id}_Z$; a right inverse of g. A case s over g is called *the worst of f*, if

$$(f \circ s)(z) = \sup f\left(g^{-1}(\{z\})\right), \qquad (27)$$

for all $z \in Z$. A *worst-case analysis of f over g* is the process of finding out $O_Z(f \circ s)$, where s is the worst case of f over g. A case s over g is called *the best of f* if

$$(f \circ s)(z) = \inf f\left(g^{-1}(\{z\})\right), \qquad (28)$$

for all $z \in Z$. A *best-case analysis of f over g* is the process of finding out $\Omega_Z(f \circ s)$, where s is the best case of f over g.

Example 3.1 (Analysis of insertion sort). Assume the RAM model, with unit cost for comparison of integers and zero cost for other atomic operations. Let $\mathbb{N}^* = \bigcup_{d \in \mathbb{N}} \mathbb{N}^d$ be the set of all finite sequences over $\mathbb{N}$. Let $F : \mathbb{N}^* \twoheadrightarrow \mathbb{N}^*$ be the insertion sort algorithm [9], which sorts a given input sequence x into increasing order. Let $f \in R_{\mathbb{N}^*}$ be the number of comparisons made by F. Let $g : \mathbb{N}^* \to \mathbb{N}$ be such that $g(x) = |x|$. Let $s : \mathbb{N} \to \mathbb{N}^*$ be the worst case of f over g; each such sequence is decreasing. Then the worst-case complexity of f over g is $f \circ s$, and the worst-case analysis of f provides $O_{\mathbb{N}}(f \circ s) = O_{\mathbb{N}}\left(n^2 + 1\right)$. Let $r : \mathbb{N} \to \mathbb{N}^*$ be the best case of f over g; each such sequence is increasing. Then the best-case complexity of f over g is $f \circ r$, and the best-case analysis of f provides $\Omega_{\mathbb{N}}(f \circ r) = \Omega_{\mathbb{N}}(n + 1)$. For an arbitrary case $p : \mathbb{N} \to \mathbb{N}^*$ of f over g, it holds that $f \circ p \in \Omega_{\mathbb{N}}(n + 1) \cap O_{\mathbb{N}}\left(n^2 + 1\right)$.

Let $(X, \Sigma_X, \mathbb{P})$ be a probability space, and (Z, Σ_Z) be a measurable space. Let $g : X \to Z$ be measurable and surjective (g is a surjective random element). Let $f \in R_X$ be measurable (f is a random variable). An *average-case analysis of f over g* is the process of finding out $O_Z(\mathbb{E}[f \mid g] \circ s)$, where $s : Z \to X$ is any case over g, and $\mathbb{E}$ stands for (conditional) expectation.

Since worst-case, best-case, and average-case analyses are the most common forms of complexity analysis in computer science — with the grouping set almost always $Z \subseteq \mathbb{N}^d$, for some $d \in \mathbb{N}^{>0}$ — this has led to the widespread misconception that the result of complexity analysis is a function which maps an 'input size' to the amount of used resources. For example, [9] writes as follows on page 25 (emphasis theirs):

> The best notion for *input size* depends on the problem being studied. For many problems, such as sorting or computing discrete Fourier transforms, the most natural measure is the *number of items in the input* - for example, the array size n for sorting. For many other problems, such as multiplying two integers, the best measure of input size is the *total number of bits* needed to represent the input in ordinary binary notation. Sometimes, it is more appropriate to describe the size of the input with two numbers rather than one. For instance, if the input to an algorithm is a graph, the input size can be described by the numbers of vertices and edges in the graph. We shall indicate which input size measure is being used with each problem we study.

We have shown above how this input-size-thinking is subsumed by the more general input-set-thinking. In the input-set thinking, a set is used to provide a mathematical model for a data structure, and resource-consumption is a function of this data.

4 Characterization of the O-notation

4.1 Linear dominance is sufficient

In this section we will show the following theorem.

Theorem 4.1 (Linear dominance has primitive properties). *Let $\bar{O}_X : R_X \to \mathcal{P}(R_X)$ be defined by*

$$g \in \bar{O}_X(f) \iff \exists c \in \mathbb{R}^{>0} : g \leq cf, \tag{29}$$

for all $f, g \in R_X$, and all $X \in U$, where the universe U is the class of all sets. Then $\bar{O}$ satisfies the primitive properties.

Proof. The result follows directly from the Lemmas in this section. □

We shall apply the following lemma repeatedly without mentioning it.

Lemma 4.2 (Simplification lemma). *Let $X \in U$, I be a finite set, $X_i \subseteq X$, $f_i \in R_{X_i}$, and $\widehat{f_i} \in \overline{O}_{X_i}(f_i)$, for all $i \in I$. Then there exists $c \in \mathbb{R}^{>0}$, such that $\widehat{f_i} \leq cf_i$, for all $i \in I$.*

Proof. Since $\widehat{f_i} \in \overline{O}_{X_i}(f_i)$, there exists $c_i \in \mathbb{R}^{>0}$, such that $\widehat{f_i} \leq c_i f$, for all $i \in I$. Let $c = \max\{c_i\}_{i \in I}$. Then $\widehat{f_i} \leq cf_i$, for all $i \in I$. □

Lemma 4.3 (Linear dominance has order-consistency). *Let $X \in U$, and $f, g \in R_X$. Then*

$$f \leq g \implies f \in \overline{O}_X(g). \tag{30}$$

Proof. Since $f \leq 1g$, it holds that $f \in \overline{O}_X(g)$. □

Lemma 4.4 (Linear dominance has transitivity). *Let $X \in U$, and $f, g, h \in R_X$. Then*

$$\Big(f \in \overline{O}_X(g) \text{ and } g \in \overline{O}_X(h)\Big) \implies f \in \overline{O}_X(h). \tag{31}$$

Proof. Let $f \in \overline{O}_X(g)$, and $g \in \overline{O}_X(h)$. Then there exists $c \in \mathbb{R}^{>0}$, such that $f \leq cg$ and $g \leq ch$. It follows that $f \leq c^2 h$. Therefore $f \in \overline{O}_X(h)$. □

Lemma 4.5 (Linear dominance has locality). *Let $X \in U$, $f, g \in R_X$, and $C \subseteq \mathcal{P}(X)$ be a finite cover of X. Then*

$$\Big(\forall D \in C : (f|D) \in \overline{O}_D(g|D)\Big) \implies f \in \overline{O}_X(g). \tag{32}$$

Proof. Assume $(f|D) \in \overline{O}_D(g|D)$, for all $D \in C$. Then there exist $c \in \mathbb{R}^{>0}$ such that $(f|D) \leq c(g|D)$, for all $D \in C$. Since C covers X, $f \leq cg$. Therefore $f \in \overline{O}_X(g)$. □

Lemma 4.6 (Linear dominance has one-separation).

$$n \notin \overline{O}_{\mathbb{N}^{>0}}(1). \tag{33}$$

Proof. For all $c \in \mathbb{R}^{>0}$, there exists $n \in \mathbb{N}^{>0}$ — for example $n = \lceil c \rceil + 1$ — such that $n > c1$. Therefore $n \notin \overline{O}_{\mathbb{N}^{>0}}(1)$. □

Lemma 4.7 (Linear dominance has scale-invariance). *Let $X \in U$, $f \in R_X$, and $\alpha \in \mathbb{R}^{>0}$. Then $f \in \overline{O}_X(\alpha f)$.*

Proof. Assume $\widehat{f} \in \overline{O}_X(f)$. Then there exists $c \in \mathbb{R}^{>0}$, such that

$$\begin{aligned} \widehat{f} &\leq cf \\ &= (c/\alpha)(\alpha f). \end{aligned} \tag{34}$$

Therefore $\widehat{f} \in \overline{O}_X(\alpha f)$. □

Lemma 4.8 (Linear dominance has $\mathbb{N}$-sub-homogenuity and $\mathbb{N}^{>0}$-cancellation). *Let $X \in U$, and $f, u \in R_X$. Then*

$$u\overline{O}_X(f) \subseteq \overline{O}_X(uf). \tag{35}$$

Proof. Let $\widehat{f} \in \overline{O}_X(f)$. Then there exists $c \in \mathbb{R}^{>0}$, such that $\widehat{f} \leq cf$. This implies $u\widehat{f} \leq cuf$. Therefore $u\widehat{f} \in \overline{O}_X(uf)$; $\overline{O}_X$ has $\mathbb{R}^{\geq 0}$-sub-homogenuity. Since $\mathbb{N} \subset \mathbb{R}^{\geq 0}$, $\overline{O}_X$ has $\mathbb{N}$-sub-homogenuity. Since $\frac{1}{\mathbb{N}^{>0}} \subset \mathbb{R}^{>0}$, $\overline{O}_X$ has $\mathbb{N}^{>0}$-cancellation. □

Lemma 4.9 (Linear dominance has sub-composability). *Let $X \in U$, $f \in R_X$, and $s : Y \to X$. Then*

$$\overline{O}_X(f) \circ s \subseteq \overline{O}_Y(f \circ s). \tag{36}$$

Proof. Let $\widehat{f} \in \overline{O}_X(f)$. Then there exists $c \in \mathbb{R}^{>0}$, such that $\widehat{f} \leq cf$. This implies $\widehat{f} \circ s \leq c(f \circ s)$. Therefore $\widehat{f} \circ s \in \overline{O}_X(f \circ s)$. □

4.2 Some implied properties

In this section we provide proofs for those implied properties which are needed in the proof of the necessity of linear dominance.

Proposition 4.10 ($\mathbb{Q}^{\geq 0}$-sub-homogenuity is a composite). *O_X has $\mathbb{Q}^{\geq 0}$-sub-homogenuity. $\iff$ O_X has $\mathbb{N}$-sub-homogenuity and $\mathbb{N}^{>0}$-cancellation.*

Proof. Suppose O_X has $\mathbb{Q}^{\geq 0}$-sub-homogenuity. Then O_X has $\mathbb{N}$-sub-homogenuity, since $\mathbb{N} \subset \mathbb{Q}^{\geq 0}$. Let $f, g, u \in R_X$, such that $u(X) \subset \mathbb{N}^{>0}$. Then $\left(\frac{1}{u}\right)(X) \subset \mathbb{Q}^{>0}$, and by $\mathbb{Q}^{\geq 0}$-sub-homogenuity,

$$\begin{aligned} & uf \in O_X(ug) \\ \implies \quad & \frac{1}{u}uf \in O_X\left(\frac{1}{u}ug\right) \\ \implies \quad & f \in O_X(g). \end{aligned} \tag{37}$$

Suppose O_X has $\mathbb{N}$-sub-homogenuity and $\mathbb{N}^{>0}$-cancellation. Let $f, g, u \in R_X$ be such that $u(X) \subset \mathbb{Q}^{\geq 0}$. Then there exists $p, q \in R_X$, such that $p(X) \subset \mathbb{N}^{\geq 0}$,

$q(X) \subset \mathbb{N}^{>0}$, and $u = p/q$. By $\mathbb{N}$-sub-homogenuity and $\mathbb{N}^{>0}$-cancellation,

$$\begin{aligned} & f \in O_X(g) \\ \implies \quad & pf \in O_X(pg) \\ \implies \quad & q\frac{p}{q}f \in O_X\left(q\frac{p}{q}g\right) \\ \implies \quad & \frac{p}{q}f \in O_X\left(\frac{p}{q}g\right) \\ \implies \quad & uf \in O_X(ug). \end{aligned} \tag{38}$$

□

Proposition 4.11 ($\mathbb{R}^{\geq 0}$-sub-homogenuity is implied). *O_X has order-consistency, transitivity, scale-invariance, $\mathbb{N}$-sub-homogenuity, and $\mathbb{N}^{>0}$-cancellation. $\implies$ O_X has $\mathbb{R}^{\geq 0}$-sub-homogenuity.*

Proof. O_X has $\mathbb{Q}^{\geq 0}$-sub-homogenuity by Proposition 4.10. Let $f, g, u \in R_X$, and $h : \mathbb{R}^{\geq 0} \to \mathbb{Q}^{\geq 0}$ be such that

$$x \leq h(x) \leq 2x. \tag{39}$$

By $\mathbb{Q}^{\geq 0}$-sub-homogenuity, order-consistency, transitivity, and scale-invariance,

$$\begin{aligned} & f \in O_X(g) \\ \implies & (h \circ u)f \in O_X((h \circ u)g) \\ \implies & uf \in O_X(2ug) \\ \implies & uf \in O_X(ug). \end{aligned} \tag{40}$$

□

Proposition 4.12 (reflexivity is implied). *O_X has order-consistency $\implies$ O_X has reflexivity.*

Proof. By order-consistency, $f \leq f \implies f \in O_X(f)$, for all $f \in R_X$; O_X has reflexivity. □

Proposition 4.13 (zero-separation is implied). *O has order-consistency, transitivity, one-separation, and $\mathbb{N}$-sub-homogenuity $\implies$ O has zero-separation.*

Proof. Suppose O does not have zero-separation, so that $1 \in O_{\mathbb{N}^{>0}}(0)$. By $\mathbb{N}$-sub-homogenuity, $n \in O_{\mathbb{N}^{>0}}(0)$. By order-consistency, $0 \in O_{\mathbb{N}^{>0}}(1)$. By transitivity, $n \in O_{\mathbb{N}^{>0}}(1)$. This contradicts one-separation. □

Proposition 4.14 (the membership rule is a composite). *O_X has reflexivity and transitivity $\iff$ O_X has the membership rule.*

Proof. $\implies$ Assume $f \in O_X(g)$. Let $\widehat{f} \in O_X(f)$. By transitivity, $\widehat{f} \in O_X(g)$, and so $O_X(f) \subseteq O_X(g)$. Assume $O_X(f) \subseteq O_X(g)$. By reflexivity, $f \in O_X(f)$. Therefore $f \in O_X(g)$, and so O_X has the membership rule.

$\impliedby$ By the membership rule, $f \in O_X(f) \iff O_X(f) \subseteq O_X(f)$. Therefore O_X has reflexivity. Let $f \in O_X(g)$, and $g \in O_X(h)$. By the membership rule, $O_X(g) \subseteq O_X(h)$. Therefore $f \in O_X(h)$, and so O_X has transitivity. □

Proposition 4.15 (the membership rule is implied). *O_X has order-consistency and transitivity. $\implies$ O_X has the membership rule.*

Proof. O_X has reflexivity by Proposition 4.12. O_X has the membership rule by Proposition 4.14. □

Proposition 4.16 (zero-triviality is implied). *O has order-consistency, transitivity, one-separation, scale-invariance, $\mathbb{N}$-sub-homogenuity, $\mathbb{N}^{>0}$-cancellation, and sub-composability. $\implies$ O has zero-triviality.*

Proof. O has zero-separation by Proposition 4.13 and $\mathbb{R}^{\geq 0}$-sub-homogenuity by Proposition 4.11. Suppose O does not have zero-triviality, so that there exists $f \in O_X(0)$ such that $f \neq 0$. Then there exists $y \in X$ such that $f(y) = c$, for some $c \in \mathbb{R}^{>0}$. Let $s : \mathbb{N}^{>0} \to X$ be such that $s(x) = y$. By sub-composability and $\mathbb{R}^{\geq 0}$-sub-homogenuity,

$$\begin{aligned}
& f \in O_X(0) \\
\implies \quad & f \circ s \in O_X(0) \circ s \\
\implies \quad & c \in O_{\mathbb{N}^{>0}}(0 \circ s) \\
\implies \quad & 1 \in O_{\mathbb{N}^{>0}}(0)/c \\
\implies \quad & 1 \in O_{\mathbb{N}^{>0}}(0/c) \\
\implies \quad & 1 \in O_{\mathbb{N}^{>0}}(0).
\end{aligned} \tag{41}$$

This contradicts zero-separation. □

Proposition 4.17 (injective super-composability is implied). *O_X has order-consistency, locality, and injective sub-composability for injective $s : Y \to X$. $\implies$ O_X has injective super-composability for s.*

Proof. Let $f \in R_X$, and $\underline{s} : Y \to s(Y)$ be such that $\underline{s}(y) = s(y)$. Then $\underline{s}$ is bijective. Let $\widehat{g} \in O_Y(f \circ s)$ and $\widehat{f} \in R_X$ be such that

$$\widehat{f}(x) = \begin{cases} \left(\widehat{g} \circ \underline{s}^{-1}\right)(x), & x \in s(Y), \\ 0, & x \notin s(Y). \end{cases} \tag{42}$$

Then $\widehat{g} = \widehat{f} \circ s$; we need to show that $\widehat{f} \in O_X(f)$. By injective sub-composability

$$\begin{aligned} \widehat{f}|s(Y) &= \widehat{g} \circ \underline{s}^{-1} \\ &\in O_Y(f \circ s) \circ \underline{s}^{-1} \\ &\subseteq O_{s(Y)}\left(f \circ s \circ \underline{s}^{-1}\right) \\ &= O_{s(Y)}(f|s(Y)). \end{aligned} \tag{43}$$

By order-consistency,

$$\widehat{f}|(X \setminus s(Y)) = 0 \in O_{X \setminus s(Y)}(f|(X \setminus s(Y))). \tag{44}$$

By locality,

$$\widehat{f} \in O_X(f). \tag{45}$$

Therefore O_X has injective super-composability for s. $\square$

Proposition 4.18 (sub-restrictability is implied). *O_X has injective sub-composability. $\implies$ O_X has sub-restrictability.*

Proof. Let $D \subseteq X$, and $s : D \to X$ be such that $s(x) = x$. Then s is injective. By injective sub-composability

$$\begin{aligned} O_X(f)|D &= O_X(f) \circ s \\ &\subseteq O_X(f \circ s) \\ &= O_X(f|D), \end{aligned} \tag{46}$$

for all $f \in R_X$. $\square$

4.3 Linear dominance is necessary

In this section we will show the following theorem.

Theorem 4.19 (Primitive properties imply linear dominance). *Suppose O has order-consistency, transitivity, one-separation, scale-invariance, locality, $\mathbb{N}$-sub-homogenuity, $\mathbb{N}^{>0}$-cancellation, and sub-composability. Then*

$$f \in O_X(g) \iff \exists c \in \mathbb{R}^{>0} : f \leq cg. \tag{47}$$

Proposition 4.20 ($O_X(1)$ **equals the bounded functions).** *Suppose O has order-consistency, transitivity, one-separation, locality, scale-invariance, and injective sub-composability. Then*

$$O_X(1) = \left\{f \in R_X : \exists c \in \mathbb{R}^{>0} : f \leq c\right\}, \tag{48}$$

provided $X \neq \emptyset$.

Proof. O has injective super-composability by Proposition 4.17, and the membership rule by Proposition 4.15.

$\subseteq$ Assume $f \in O_X(1)$ such that f is unbounded. Then for every $n \in \mathbb{N}^{>0}$ there exists $x_n \in X$ such that $f(x_n) \geq n$. Therefore, let $s : \mathbb{N}^{>0} \to X$ be injective such that $n \leq (f \circ s)(n)$, for all $n \in \mathbb{N}^{>0}$. By the membership rule, $O_X(f) \subseteq O_X(1)$. By order-consistency, order-consistency and transitivity, injective super-composability, and injective sub-composability,

$$\begin{aligned}
(n \mapsto n) &\in O_{\mathbb{N}^{>0}}(n) \\
&\subseteq O_{\mathbb{N}^{>0}}(f \circ s) \\
&\subseteq O_X(f) \circ s \\
&\subseteq O_X(1) \circ s \\
&\subseteq O_{\mathbb{N}^{>0}}(1 \circ s) \\
&= O_{\mathbb{N}^{>0}}(1).
\end{aligned} \tag{49}$$

This contradicts O having one-separation. Therefore f is bounded, which is equivalent to $\exists c \in \mathbb{R}^{>0} : f \leq c$.

$\supset$ Assume $\exists c \in \mathbb{R}^{>0} : f \leq c$. By order-consistency, $f \in O_X(c)$. By scale-invariance, $f \in O_X(1)$.

□

Proposition 4.21 (Big-oh for positive functions). *Suppose O_X has order-consistency, transitivity, scale-invariance, $\mathbb{N}$-sub-homogenuity, and $\mathbb{N}^{>0}$-cancellation. Then*

$$f \in O_X(g) \iff f/g \in O_X(1), \tag{50}$$

for all $f, g \in R_X$ such that $g > 0$.

Proof. O_X has $\mathbb{R}^{\geq 0}$-sub-homogenuity by Proposition 4.11.

$\implies$ By $\mathbb{R}^{\geq 0}$-sub-homogenuity,

$$\begin{aligned}
& f \in O_X(g) \\
\implies \quad & f/g \in O_X(g)/g \\
\implies \quad & f/g \in O_X(g/g) \\
\implies \quad & f/g \in O_X(1).
\end{aligned} \tag{51}$$

$\impliedby$ By $\mathbb{R}^{\geq 0}$-sub-homogenuity,

$$\begin{aligned}
& f/g \in O_X(1) \\
\implies \quad & f \in O_X(1)g \\
\implies \quad & f \in O_X(g).
\end{aligned} \tag{52}$$

□

Theorem 4.22 (Primitive properties imply linear dominance). *Suppose O has order-consistency, transitivity, one-separation, scale-invariance, locality, $\mathbb{N}$-sub-homogenuity, $\mathbb{N}^{>0}$-cancellation, and sub-composability. Then*

$$f \in O_X(g) \iff \exists c \in \mathbb{R}^{>0} : f \leq cg. \tag{53}$$

Proof. O_X has sub-restrictability by Proposition 4.18. O_X has zero-triviality by Proposition 4.16. Let $G := g^{-1}(\mathbb{R}^{>0})$ and $\overline{G} := X \setminus G$. Then

$$\begin{aligned}
& (f|G) \in O_G(g|G) \\
\iff \quad & \frac{(f|G)}{(g|G)} \in O_G(1) \\
\iff \quad & \exists c \in \mathbb{R}^{>0} : \frac{(f|G)}{(g|G)} \leq c \\
\iff \quad & \exists c \in \mathbb{R}^{>0} : (f|G) \leq c(g|G)
\end{aligned} \tag{54}$$

where we used Proposition 4.21 and Proposition 4.20. On the other hand,

$$\begin{aligned}
& \left(f|\overline{G}\right) \in O_{\overline{G}}\left(g|\overline{G}\right) \\
\iff \quad & \left(f|\overline{G}\right) = 0
\end{aligned} \tag{55}$$

by zero-triviality. By locality, sub-restrictability, and zero-triviality,

$$\begin{aligned}
& f \in O_X(g) \\
\iff \quad & (f|G) \in O_G(g|G) \text{ and } \left(f|\overline{G}\right) \in O_{\overline{G}}\left(g|\overline{G}\right) \\
\iff \quad & \left(\exists c \in \mathbb{R}^{>0} : (f|G) \leq c(g|G)\right) \text{ and } \left(f|\overline{G}\right) = 0 \\
\iff \quad & \exists c \in \mathbb{R}^{>0} : f \leq cg.
\end{aligned} \tag{56}$$

□

4.4 Completeness

The $\preceq_X$ is *complete* on $A \subseteq \mathcal{P}(R_X)$, if every $F \in A$ which has a lower-bound (an upper-bound) in R_X has a greatest lower-bound (a least upper-bound) in R_X. The $\preceq_X$ is *complete*, *directed-complete*, *chain-complete*, a *lattice*, and *algorithm-complete*, if it is complete on $\mathcal{P}(R_X)$, complete on directed subsets of R_X, complete on linearly-ordered subsets of R_X, complete on finite subsets of R_X, and complete on $\left\{\{\mathcal{R}(F)\}_{F \in \mathcal{A}(P)}\right\}_{P \in (X \to P(X))}$, respectively.

It follows from the axiom of choice that chain-complete is equivalent to directed-complete. Since $\preceq_X$ is a lattice by Proposition 4.23, every subset of R_X is directed, and therefore directed-complete is equivalent to complete.

Completeness on $\mathcal{P}(R_X)$ is too strict a condition to require, because — under commonly used models — most of these sets can never be realized by algorithms. Instead, the appropriate sets are those which are generated as the resource-consumptions of algorithms to solve computational problems. However, this algorithm-completeness can be harder to examine.

By order-consistency, every subset of R_X has the trivial lower-bound 0, and so the requirement for a lower-bound is redundant.

Proposition 4.23 (lattice structure is implied). *$\preceq_X$ has order-consistency, locality, and injective sub-composability. $\implies$ $\preceq_X$ has lattice structure.*

Proof. Let $f_1, \dots, f_n \in R_X$. Then $f_1, \dots, f_n \leq \max(f_1, \dots, f_n)$, and by order-consistency $f_1, \dots, f_n \preceq_X \max(f_1, \dots, f_n)$. Suppose $h \in R_X$ is such that $f_1, \dots, f_n \preceq_X h \preceq_X \max(f_1, \dots, f_n)$. Let $F_i = \{x \in X : f_i(x) = \max(f_1, \dots, f_n)\}$. By injective sub-composability, $(f_i|F_i) \preceq_{F_i} (h|F_i) \preceq_{F_i} (f_i|F_i)$. Therefore $(f_i|F_i) \approx_{F_i} (h|F_i)$. By locality, $\max(f_1, \dots, f_n) \approx_X h$. Therefore $\sup\{f_1, \dots, f_n\} \approx_X \max(f_1, \dots, f_n)$. Similarly, $\inf\{f_1, \dots, f_n\} \approx_X \min(f_1, \dots, f_n)$. □

Theorem 4.24 (Incompleteness). *$\preceq_{\mathbb{N}^{>0}}$ has order-consistency, transitivity, $\mathbb{N}$-sub-homogenuity, $\mathbb{N}^{>0}$-cancellation, scale-invariance, and one-separation. $\implies$ $\preceq_{\mathbb{N}^{>0}}$ is not complete.*

Proof. $\preceq_{\mathbb{N}^{>0}}$ has $\mathbb{R}^{\geq 0}$-sub-homogenuity by Proposition 4.11.

Let $f_\alpha \in F_{\mathbb{N}^{>0}}$ be such that $f_\alpha(n) = \alpha^n$, for all $\alpha \in \mathbb{R}^{\geq 0}$. By order-consistency, $\alpha \leq \beta \implies f_\alpha \preceq_{\mathbb{N}^{>0}} f_\beta$, for all $\alpha, \beta \in \mathbb{R}^{\geq 0}$.

Suppose there exists $\alpha, \beta \in \mathbb{R}^{\geq 0}$ such that $\alpha > \beta$ and $f_\alpha \preceq_{\mathbb{N}^{>0}} f_\beta$. By $\mathbb{R}^{\geq 0}$-sub-homogenuity, $\left(f_\alpha / f_\beta\right) \preceq_{\mathbb{N}^{>0}} 1$. It can be shown that $n \leq \frac{1}{e \log_e(\alpha/\beta)}\left(f_\alpha / f_\beta\right)$. By order-consistency and scale-invariance, $n \preceq_{\mathbb{N}^{>0}} \left(f_\alpha / f_\beta\right)$. By transitivity, $n \preceq_{\mathbb{N}^{>0}} 1$, which contradicts one-separation. Therefore, $\alpha < \beta \implies f_\alpha \prec_{\mathbb{N}^{>0}} f_\beta$, for all $\alpha, \beta \in \mathbb{R}^{\geq 0}$.

Let $F = \{f_\alpha\}_{\alpha \in \mathbb{R}^{>2}}$. Then f_2 is a lower-bound of F. Let $\underline{f} \in R_{\mathbb{N}^{>0}}$ be a lower-bound of F such that $f_2 \preceq_{\mathbb{N}^{>0}} \underline{f}$.

Suppose $\underline{f} \approx_{\mathbb{N}^{>0}} \beta^n$, for some $\beta \in \mathbb{R}^{>2}$. Then $\left(\frac{2+\beta}{2}\right)^n \prec_{\mathbb{N}^{>0}} \beta^n \approx_{\mathbb{N}^{>0}} \underline{f}$, which contradicts $\underline{f}$ being a lower-bound of F. Therefore $\underline{f} \prec_{\mathbb{N}^{>0}} \alpha^n$, for all $\alpha \in \mathbb{R}^{>2}$.

By order-consistency, $\underline{f} \preceq_{\mathbb{N}^{>0}} n\underline{f}$. Suppose $n\underline{f} \preceq_{\mathbb{N}^{>0}} \underline{f}$. By $\mathbb{R}^{\geq 0}$-sub-homogenuity, $n \preceq_{\mathbb{N}^{>0}} 1$, which contradicts one-separation. Therefore $\underline{f} \prec_{\mathbb{N}^{>0}} n\underline{f}$.

By $\mathbb{R}^{\geq 0}$-sub-homogenuity, $n\underline{f} \prec_{\mathbb{N}^{>0}} n\alpha^n$, for all $\alpha \in \mathbb{R}^{>2}$. It can be shown that $n\alpha^n \leq \frac{\beta/\alpha}{\log_e(\beta/\alpha)e}\beta^n$, for all $\alpha, \beta \in \mathbb{R}^{>2}$ such that $\alpha < \beta$. By order-consistency and scale-invariance, $n\alpha^n \preceq_{\mathbb{N}^{>0}} \beta^n$, for all $\alpha, \beta \in \mathbb{R}^{>2}$ such that $\alpha < \beta$. By transitivity, $n\underline{f} \prec_{\mathbb{N}^{>0}} \alpha^n$, for all $\alpha \in \mathbb{R}^{>2}$; $n\underline{f}$ is a lower-bound for F. Since $\underline{f} \prec_{\mathbb{N}^{>0}} n\underline{f}$, there is no greatest lower-bound for F.

□

Remark 4.25 (Algorithm-completeness). Are there commonly used computational models and cost-models where $\{\leq_X\}_{X \in U}$ are algorithm-complete? This question is open. $\triangle$

5 Conclusion

We showed that the primitive properties for the O-notation are equivalent to its definition as linear dominance.

We had to skip several interesting topics due to space-constraints. For practical use, the most important thing is to prove that the listed desirable properties really are implied by the primitive properties. Then, it must be made sure that the Master theorems in their various forms hold for linear dominance. For intuition, it can be interesting to compare various candidate definitions for the O-notation, and see how they fail the primitive properties.

All this is available in our extended Arxiv-paper with the same name. In particular, we have proved that the Master theorems in their various forms hold without any regularity conditions.

6 Acknowledgements

We would like to thank the anynomous reviewer, whose comments greatly improved the presentation of this paper, as well as led to $\mathbb{R}^{\geq 0}$-sub-homogenuity being split into $\mathbb{N}$-sub-homogenuity and $\mathbb{N}^{>0}$-cancellation.

The work for this paper distributed as follows. KR did the research, and wrote the paper. GGH, SLE, and KE checked and commented on the paper on its various revisions.

The research was done while KR was under funding from the Finnish National Doctoral Programme in Mathematics and its Applications. The review process was carried out while KR was under funding from the science fund of the city of Tampere.

References

[1] Elliott Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall/CRC, 5th edition, 2009.

[2] Yuri Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Trans. Comput. Logic*, 1(1):77–111, July 2000.

[3] Andreas Blass and Yuri Gurevich. Abstract state machines capture parallel algorithms. *ACM Transactions on Computational Logic*, 4:578–651, 2003.

[4] Egon Börger and Robert F. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[5] Lenore Blum, Michael Shub, and Stephen Smale. On a theory of computation over the real numbers; np completeness, recursive functions and universal machines. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 387–397, Washington, DC, USA, 1988. IEEE Computer Society.

[6] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[7] Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pages 73–80, New York, NY, USA, 1972. ACM.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[10] Prosenjit Bose, Karim Douïeb, Vida Dujmović, John Howat, and Pat Morin. Fast local searches and updates in bounded universes. *Computational Geometry*, 46(2):181 – 189, 2013.

[11] Paul Bachmann. *Die Analytische Zahlentheorie*. B. G. Teubner, Leipzig, 1894.

[12] Edmund Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. B. G. Teubner, Leipzig, volume 2 edition, 1909.

[13] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 1st edition, 1968.

[14] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 2nd edition, 1973.

[15] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, April 1976.

[16] Rodney R. Howell. On asymptotic notation with multiple variables, 2008.