# On Computational Complexity for Finite Languages

Caleb Stanford

February 2021

**Abstract**

We define a notion of computational complexity for finite languages using extended regular expressions. This "complexity" is based on program size, i.e. similar to circuit complexity and Kolmogorov complexity.

Turing Machines have long been known to be inadequate as a model of real computation for a number of reasons; in particular, the mismatch between *asymptotic* complexity (i.e. running time/space as $n \to \infty$ ignoring constants) and running time in the real world, where we typically care about concrete time and space bounds for finite inputs. In particular, real computers that are built today only have finite RAM and storage, so an infinite tape is not a correct abstraction. There is the RAM model as a possible alternative, but with well-known problems and being a difficult model that few people understand.n Researchers such as e.g. Moshe Vardi have argued that finite-state transducers are a better model of computation for these reasons, but these are appropriate for modeling regular languages. Here we sketch a definition of complexity for *finite* languages, i.e. the problems that people actually care about and solve in the real world. The below has probably been done for regular expressions, but using extended regular expressions instead seems more appropriate (in particular, treats a problem and its negation similarly).

Define a *finite language* to be $(P, N)$ where $P, N \subseteq \{0, 1\}^*$ are finite and $P \cap N = \varnothing$: a finite set of positive examples and a finite set of negative examples.

Define an *extended regular expression* $R$ to be constructed using the following: 0, 1, $\varnothing$, $^*$ (Kleene star), $\cdot$ (concatenation), $\cup$ (union), $\cap$ (intersection), and $^C$ (complement). For example: $R = (a \wedge ab^*) \vee (cd^*)$. The size is the length of the expression (a nonnegative integer). We let $L(R) \subseteq \{0, 1\}^*$ denote the language of $R$, defined in the obvious way. NB: It would be interesting to add other operators here, for instance, counting loops and back-references.

We say that $R$ *computes* $(P, N)$ if $P \subseteq L(R)$ and $N \subseteq (L(R))^C$. Note that any finite language $(P, N)$ is computable this way for some regex $R$.

*Finally, we define the* complexity *of* $(P, N)$ *to be the smallest size of an extended regular expression* $R$ *such that* $R$ *computes* $(P, N)$.

For example: the complexity of the parity function (evenness of number of bits) would be constant regardless of the choice of bit length. The complexity of an “arbitrary” finite language is likely similar to the size of writing out the members of that language explicitly. It would be interesting to figure out the complexity of languages like $\{0^n 1^n : n \in \mathbb{N}\}$ or the Dyck language (matched parentheses) in this model.