Abstract for the general public

The standard notion of "efficient" or "tractable" computation is programs whose running time is bounded by some polynomial, such as $3n^2 + 7n + 5$, where n is the size of the input. The objective of this proposal is to build a theory of polynomial computation by finite-state devices, such as finite automata. We will study functions that input and output objects such as lists or trees, do so in a "finite-state" way, and play – in the finite-state context – the same role as the class PTIME plays for Turing machines. Apart from reasons of a mathematical nature, the purpose of studying finite-state devices, as opposed to general Turing machines, is to have models for which the halting problem is decidable.

What is a finite-state function? Although we do not know the answer in general, and this proposal is intended to find it, we do know the answer when the inputs are strings, and the outputs are the Boolean values "yes" or "no". In this case, the finite-state functions are the *regular languages*; their theory is firmly established and we do not plan to extend it. Regular languages can be described in many different ways: finite automata (dozens of equivalent models), regular expressions, finite semigroups, or logics (most prominently, monadic second-order logic MSO). Each of these descriptions is useful in different contexts, and the striking fact that they all describe the same class of languages makes for a very appealing theory. This appeal is one of the reasons why the notion of "regular language" has been studied intensively in theoretical computer science, with many results motivated by logic and also the study of other input structures such as trees or graphs. The connections between automata and logic have had a profound impact on (not only theoretical) computer science, including four Turing Awards: in 1976 Rabin and Scott were awarded for introducing nondeterministic machines (finite automata, in their case), in 1996 Pnueli was awarded for introducing temporal logic as a language for program specification (later on, following Vardi and Wolper, automata became the principal means to reason about temporal logics), in 2007 Clarke, Emerson and Sifakis were awarded for model checking (the behavior of a program or protocol is modelled by a finite automaton), and in 2020 Aho and Ullman were awarded for algorithms underlying programming language implementation (including a substantial transducer component in the context of parsing).

The classical model of an automaton inputs a string and returns a Boolean, i.e. a "yes" or "no" value. Outside the classroom, however, a computer program does not simply produce a Boolean. A program can reformat a file, or it can execute another program. From the theoretical point of view, the difference between Boolean outputs and other outputs is frequently irrelevant, since for many computation models, Boolean outputs can be used to represent more complicated outputs. For example, a string-to-string function can be represented as a string-to-Boolean function, i.e. a language, which inputs the string and an indicated bit position in the output, and returns the value of that bit. For Turing machines, say running in polynomial time, there is no difference between a string-to-string function and its language representation, so one can easily restrict attention to languages. However, finite-state devices, such as automata, are not capable of counting bits, and therefore one cannot represent a string-to-string function using an automaton that reads an input string together with an indicated output position. Hence the study of finite-state devices which produce non-trivial outputs (such devices are called *transducers*) is not the same as the study of automata as language recognizers.

Transducers have been around since the beginnings of automata theory; in fact, the early authors understood automata to have string outputs. In their Turing Award paper that introduces nondeterminism, Rabin and Scott advocated moving to the language approach, writing that they were "doing away with a complicated output function and having our machines simply give *yes* or *no* answers". Following this advice, much research on automata has focussed on languages, and not transducers. The present project runs counter to this advice and takes the transducer perspective.

The early transducer models were one-way automata equipped with some simple output mechanism, e.g. each transition could append a letter to an output string. In recent years transducers have grown in sophistication, reaching a point where they resemble programming languages, with constructs such as numerical variables, loops, recursion or higher-order functions. This means that modern models of transducers can now be actually useful programs, as opposed to being idealized and radically simplified models of useful programs (as in model checking). All of this while retaining the good decidability properties of automata (transducers are not supposed to be Turing complete, which is framed here as an advantage: the halting problem is decidable) and their attractive mathematical theory.

The principal goal of this project is to

Identify a notion of finite-state polynomial computation.

Prior work has already produced a satisfactory notion of finite-state linear computation; but the polynomial case is relatively unexplored. We intend to: (a) identify what kind of structures, beyond strings, can be manipulated by finite-state programs; (b) propose polynomial models that manipulate these structures in a finite-state way; (c) give algorithms for reasoning about the proposed model, in particular for deciding equivalence; and (d) prove that the proposed model is unique, i.e. there is no other possible model of finite-state computation.