

What is Computation?

Rephrase the question as: „What different kinds of computation are there?“

Motivation: Different mathematical models with different expressive power
Is there a hierarchy, that is, are some models more general / powerful than others and are some equivalent

Granted: The term computation is not really illuminated; it is, more or less, taken to be well understood in its everyday meaning

Still: The models define certain key concepts, and since each model is a model of computation we gain at least indirect knowledge about the process of computation (and its limits)

Since we have models of computational systems, we might define the term computable with respect to these models → *we have at least a criterion what can be the result of a computation and what can't*

Different models:

Finite-State-Automaton:

Think of it as a set of states with a primitive binary relation defined between those states. This relation can be paraphrased as: If I am in state s_1 and read a symbol S , I am now in state s_2

This model is limited to those symbols that are successively put into it – at each step in the computation you know what to do next solely by knowing the next input
to use a familiar term, a computation can be said to converge if after having processed all its input it is in a valid final state

this model actually is more powerful than one might think – it is used to model the so-called regular languages (morphological systems are usually modeled as FSAs) → Type-3 languages
The classification into Type-0 to Type-3 (whereas Type-0 languages are meant to cover all possible type of expressions) goes back to Chomsky → Chomsky Hierarchy as an example for a hierarchy of computational complexity

To address some questions right now:

Why are we talking about languages?

Because even the whole of math might be thought of as a language, whereas all expressions are the well-formed mathematical expressions. Someone over at Wikipedia found the right words: “This is a somewhat esoteric way of asking this question, so an example is illuminating. We might define our language as the set of all strings of digits which represent a prime number. To ask whether an input string is a member of this language is equivalent to asking whether the number represented by that input string is prime. Similarly, we define a language as the set of all palindromes, or the set of all strings consisting only of the letter 'a'. In these examples, it is easy to see that constructing a computer to solve one problem is easier in some cases than in others.” [[http:// http://en.wikipedia.org/wiki/Computability_theory_\(computer_science\)#Introduction](http://en.wikipedia.org/wiki/Computability_theory_(computer_science)#Introduction)]

If you want to *compute* the result of an expression, you can think of the expression as an input and the rules of the system ‘transform’ the input into the desired output (things actually are much more complicated and I am not under the illusion to really understand them, so I hope this short answer satisfies you; this much be said – the concepts addressed here only *recognize*

whether the input is valid, they do not really produce any output. Therefore, you need concepts with a little more expressive power, such as transducers. The Turing-Machine, however, gives an output).

Where's the meaning?

Glad you asked (no, really) – computation can be thought of as a purely syntactic process (I bet you saw that coming). This is not to say that computations are meaningless – just, that the structures need to be interpreted (so: we compute meanings...what's that supposed to mean? I do not know. Perhaps something like ,We compute a syntactic representation that can be handled by the system responsible for meaning – this system, however, does not produce a new *syntactic structure*, it produces meaning, whatever that might be.“ Perhaps now it's easier for you to understand why persons like Wittgenstein and Quine were so anti-meanings-as-entities).

Another model for computation, one step higher in the hierarchy, are the so-called **push-down automata** that are described by (our familiar) Context-Free-Grammars. Whilst Context-Free-Grammars are quite powerful, certain phenomena (e.g. cross-serial dependencies: there is a prove that Swiss German cannot be described by any Context-Free-Grammar, no matter how complex) cannot be covered by them → Chomsky's introduction of transformation rules. This, however, leads to a much more complex types of computation that can no longer be modeled by a push down automata but require a Turing Machine.

Now, all of you may have heard the term '**Turing Machine**' – but what is it and why is it important?

To put things short, the Turing Machine is THE model for computation (there are other formalism, such as WHILE-program, lambda-calculus etc. which were proven to be equally powerful). Church's theorem states that the intuitive notion of computability (that is, what we think might be computed at all – again, we do have an intuitive understanding) is the same as that of turing-machine-computability: Every function that is computable at all can be thought of as a program (some kind of grammar) for a Turing-Machine.

Now, why are we interested in this kind of notions (why is anybody interested in it)? First of all, these models let us talk about various degrees of complexity of computation. Therefore, we can put questions such as “Can something be computed at all.” more precisely (can it be computed by a FSA...) and can find new ways to answer questions. Once you've shown, for example, that Context-Free-Grammars are decidable (that is: For each string, the question whether this string might have been generated by a given context-free-grammar, can be answered in finite time) you can try to show that a problem can be adequately described in the context-free framework and then know that it is decidable (whether you can decide it or not). On the other hand, certain problems are undecidable in principle (we can not guarantee that we will find an answer, no matter how long we search for one) – to show that something is undecidable, we 'only' have to show that the characteristic function is not-turing-computable (and therefore not computable at all).

I plan to show you some live-action at the blackboard tomorrow for the primitive models such as finite-state-automatas (and perhaps a push-down-automaton) – so don't worry if you do not know by now what this is all about.

Now, what is a computation? At least we know are possible models for computational systems are...

And that there are certain things in the world that just cannot be (the result of) a computation, for whatever we (intuitively) know computation to mean (and what else should it mean) can be proven to be inadequate as the process of its generation.