art.1 Introduction

inc:art:int:sec

In order to connect computability and logic, we need a way to talk about the objects of logic (symbols, terms, formulas, derivations), operations on them, and their properties and relations, in a way amenable to computational treatment. We can do this directly, by considering computable functions and relations on symbols, sequences of symbols, and other objects built from them. Since the objects of logical syntax are all finite and built from an enumerable sets of symbols, this is possible for some models of computation. But other models of computation—such as the recursive functions—-are restricted to numbers, their relations and functions. Moreover, ultimately we also want to be able to deal with syntax within certain theories, specifically, in theories formulated in the language of arithmetic. In these cases it is necessary to *arithmetize* syntax, i.e., to represent syntactic objects, operations on them, and their relations, as numbers, arithmetical functions, and arithmetical relations, respectively. The idea, which goes back to Leibniz, is to assign numbers to syntactic objects.

It is relatively straightforward to assign numbers to symbols as their "codes." Some symbols pose a bit of a challenge, since, e.g., there are infinitely many variables, and even infinitely many function symbols of each arity n. But of course it's possible to assign numbers to symbols systematically in such a way that, say, v_2 and v_3 are assigned different codes. Sequences of symbols (such as terms and formulas) are a bigger challenge. But if can deal with sequences of numbers purely arithmetically (e.g., by the powers-of-primes coding of sequences), we can extend the coding of individual symbols to coding of sequences of symbols, and then further to sequences or other arrangements of formulas, such as derivations. This extended coding is called "Gödel numbering." Every term, formula, and derivation is assigned a Gödel number.

By coding sequences of symbols as sequences of their codes, and by chosing a system of coding sequences that can be dealt with using computable functions, we can then also deal with Gödel numbers usign computable functions. In practice, all the relevant functions will be primitive recursive. For instance, computing the length of a sequence and computing the i-th element of a sequence from the code of the sequence are both primitive recursive. If the number coding the sequence is, e.g., the Gödel number of a formula φ, we immediately see that the length of a formula and the (code of the) i-th symbol in a formula can also be computed from the Gödel number of φ. It is a bit harder to prove that, e.g., the property of being the Gödel number of a correctly formed term, of being the Gödel number of a corret derivation is primitive recursive. It is nevertheless possible, because the sequences of interest (terms, formulas, derivations) are inductively defined.

As an example, consider the operation of substitution. If φ is a formula, x a variable, and t a term, then $\varphi[t/x]$ is the result of replacing every free occurrence of x in φ by t. Now suppose we have assigned Gödel numbers to φ, x, t—say, k, l, and m, respectively. The same scheme assignes a Gödel number to $[t/x]$, say, n. This mapping—of k, l, m to n—is the arithmetical analog of

the substitution operation. When the substitution operation maps φ, x, t to $\varphi[t/x]$, the arithmetized substitution functions maps the Gödel numbers k, l, m to the Gödel number n. We will see that this function is primitive recursive.

Arithmetization of syntax is not just of abstract interest, although it was originally a non-trivial insight that languages like the language of arithmetic, which do not come with mechanisms for "talking about" languages can, after all, formalize complex properties of expressions. It is then just a small step to ask what a theory in this language, such as Peano arithmetic, can *prove* about its own language (including, e.g., whether sentences are provable or true). This leads us to the famous limitative theorems of Gödel (about unprovability) and Tarski (the undefinability of truth). But the trick of arithmetizing syntax is also important in order to prove some important results in computability theory, e.g., about the computational prower of theories or the relationship between different models of computability. The arithmetization of syntax serves as a model for arithmetizing other objects and properties. For instance, it is similarly possible to arithmetize configurations and computations (say, of Turing machines). This makes it possible to simulate computations in one model (e.g., Turing machines) in another (e.g., recursive functions).

Photo Credits

Bibliography