The Language of Arithmetic

The *language of arithmetic* is the language whose only individual constant is "0," whose function signs are "s," "+," "•," and "E," and whose predicates are "<" and "=." In the *standard model*, which we call "ℕ,"[1] "0" denotes the number 0, the four functions signs denote the successor function, addition, multiplication, and exponentiation, respectively, "<" stands for "less than," and "=" stands for identity. More specifically:

> "0" is a term of the language of arithmetic.
>
> The variables "x_0," "x_1," "x_2," "x_3," and so on are terms of the language of arithmetic.[2]
>
> If τ and ρ are symbols of the language of arithmetic, so are $s\tau$, $(\tau + \rho)$, $(\tau \bullet \rho)$, and $(\tau E \rho)$.
>
> Nothing else is a term of the language of arithmetic, unless it is required to be by the three clauses above.

We have *unique readability*: every term is built up in a unique way. More specifically, for each term σ, there is a unique finite labeled tree, called the *structure tree* for σ, with the following features:

> The trunk of the tree is labeled with σ.
>
> If a node of the true is labeled by $s\tau$, then there is directly beneath it one node labeled by τ.

1 We also use "ℕ" to denote the set of natural numbers. I hope that no confusion will result.

2 We'll only use these variables on black-tie occasions. Most of the time, we'll use other letters from the end of the alphabet as variables, to avoid proliferation of subscripts.

If a node is labeled by $(\tau + \rho)$, by $(\tau\bullet\rho)$, or by $(\tau E\rho)$, then there are two nodes directly beneath it, the left labeled by τ and the right by ρ.

Each leaf of the tree is labeled either by “0” or by a variable.

A *closed term* is a term that contains no variables. Within the standard model, each closed term denotes a natural number, uniquely determined as follows:

$\text{Den}(\text{“0”}) = 0.$

$\text{Den}(s\tau) = \text{Den}(\tau) + 1.$

$\text{Den}((\tau + \rho)) = \text{Den}(\tau) + \text{Den}(\rho).$

$\text{Den}((\tau\bullet\rho)) = \text{Den}(\tau)\bullet\text{Den}(\rho)$

$\text{Den}((\tau E\rho)) = \text{Den}(\tau)^{\text{Den}(\rho)}.$

We introduce a *standard numeral* for each number by stipulating that [n] is the symbol obtained by writing n “s”s in front of “0,” so that, for example [7] = “sssssss0.”

The function that takes the symbol τ to the number $\text{Den}(\tau)$ is calculable. One algorithm you might use for this purpose is, first, to produce a labeled tree with τ at its trunk, representing the structure of τ, as described above; and then to associate a number with the label for each node, working from the leaves toward the trunk. “0,” which labels the leaves, is associated with 0. If If the nodes associated with ρ and μ are associated with n and m, respectively, associate n+m with $(\rho + \mu)$. And so on.

If you were to try to write this algorithm as a computer program, you would find it to be a bit of a nuisance. If you are confronted with a term, it’s easy to tell, by visual inspection, what its main function sign is, and so it’s easy to write down the labeled tree that represents its structure. When, however, you try to write down an explicit algorithm, you have to fret more than you’d like about the mating habits of parentheses. We can avoid this by using *Polish notation*, in which the function sign is written before the arguments and there are no parentheses. For example, the Polish notation for "((sss0•ss0) + s0)" is "+•sss0ss0s0"; for "(sss0•(ss0 + s0))," it's "•sss0+ss0s0." Annoying for humans, but convenient for machines. Here we'll try to have it both ways. The language of arithmetic will use ordinary notation, with parentheses, but when we set about trying to represent the expressions of the language by numerical codes, we'll use Polish notation. But that can wait a while.

The *atomic formulas* of the language of the arithmetic are expressions of one of the forms $\tau < \rho$ or $\tau = \rho$, for τ and ρ terms. The *formulas* are characterized by the following stipulation:

Every atomic formula is a formula.

If ϕ and ψ are formulas, so are $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \rightarrow \psi)$, $(\phi \leftrightarrow \psi)$, $\neg\phi$, $(\exists x_n)\phi$, and $(\forall x_n)\phi$, for each n.

Nothing is a formula, unless it's required to be by the clauses above.

We have unique readability for formulas, just as for terms.

An occurrence of the variable x_n within a formula is *bound* iff it occurs within some subformula that begins with $(\forall x_n)$ or $(\exists x_n)$. Occurrences that aren't bound are *free*. A *sentence* is a formula that contains no free variables.

For ϕ a formula and τ a term, let $\phi^{x_n}/_{\tau}$ be the formula obtained from ϕ by substituting τ for each free occurrence of x_n in ϕ. Where there's no threat of confusion about which variable is involved, we'll sometime write $\phi(\tau)$ instead.

Truth in the standard model is defined by first stating the truth conditions for atomic sentences, then seeing how the truth conditions for complex sentences are determined by the truth conditions for simpler sentences. A special feature of the standard model is that every member of the domain of discourse is named by some numeral. This special feature will simplify the definition of truth, since we can define truth directly, rather than having to define truth in terms of satisfaction.

$\tau < \rho$ is true if the standard model iff Den(τ) < Den(ρ).

$\tau = \rho$ is true in the standard model iff Den(τ) = Den(ρ).

$(\phi \vee \psi)$ is true in the standard model iff either or both of ϕ and ψ are true in the standard model.

$(\phi \wedge \psi)$ is true in the standard model iff both ϕ and ψ are true in the standard model.

$(\phi \rightarrow \psi)$ is true in the standard model iff either ϕ isn't true in the standard model or ψ is.

$(\phi \leftrightarrow \psi)$ is true in the standard model iff either ϕ and ψ are both true in the standard model or neither of them is.

$\neg\phi$ is true in the standard model iff ϕ isn't true in the standard model.

$(\exists x_n)\phi$ is true in the standard model iff, for some k, $\phi^{x_n}/_{[k]}$ is true in the standard model.

$(\forall x_n)\phi$ is true in the standard model iff, for every k, $\phi^{x_n}/_{[k]}$ is true in the standard model.

ϕ is *false* in the standard model iff $\neg\phi$ is true in the standard model.

True arithmetic is the set of sentences true in the standard model We shall see that there is no decision procedure for true arithmetic, or even a proof procedure. We do, however, have this much:

> **Proposition.** There is a decision procedure for the set of true quantifier-free sentences.[3]

Proof: First note that there is a decision procedure for the set of true atomic sentences. Namely, $\tau < \rho$ is true iff Den(τ) < Den(ρ) and $\tau = \rho$ is true iff Den(τ) = Den(ρ). The truth of a quantifier-free sentence is determine by the truth or untruth of its atomic components by the laws of the sentential calculus. A way to determine the truth or falsity of a quantifier-free sentence θ is to form the structure true for θ, then associating a truth value, truth or falsity, with the label of each node, starting with the leaves and working toward the trunk. For example, a node labeled $(\phi \vee \psi)$ will be labeled "true" if and only if one or both of the nodes immediately beneath it is labeled "true."⊠

We introduce the *bounded quantifiers*: for ϕ a formula and τ term, $(\exists x_n < \tau)\phi$ will be an abbreviation for $(\exists x_n)(x_n < \tau \wedge \phi)$. $(\forall x_n < \tau)\phi$ abbreviates $(\forall x_n)(x_n < \tau \rightarrow \phi)$. The *bounded formulas* are characterized as follows:

> Every atomic formula is bounded.
>
> If ϕ and ψ are bounded formulas, so are $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \rightarrow \psi)$, $(\phi \leftrightarrow \psi)$ $\neg\phi$, $(\exists x_n < \tau)\phi$, and $(\forall x_n < \tau)\phi$, for each n and τ.
>
> Nothing else is a bounded formula.
>
> A *bounded set* is the extension of a bounded formula, that is, it is a set of the form $\{x: \phi([x])\}$, for some bounded formula ϕ. Similarly for *bounded relations.*

3 Except when there's an indication to the contrary, by "true" arithmetical sentence, I shall mean an arithmetical sentences true in the standard model.

The functions Pair, 1st, and 2nd are bounded. So is the relation that holds between x and y iff x is an element of the set whose code number is y; we'll abuse notation slightly by writing "$x \in y$." The set of code numbers of sequences is a bounded set, as is the partial function that takes x and i to the ith member of the sequence coded by x (which we write "$(x)_i$") if x codes a sequence of length i or greater, and is undefined otherwise.

Proposition. There is a decision procedure for the set of true bounded sentences.

Proof: Working from the outside in, replace each subformula of the form $(\forall x < \tau)\psi$ by the conjunction[4] of all the $\psi([k])$s for $k < \mathrm{Den}(\tau)$. Replace $(\exists x < \tau)\psi$ by the disjunction of all the $\psi([k])$s for $k < \mathrm{Den}(\tau)$. Continue this until you've eliminated all the bounded quantifiers, then test the truth of the quantifier-free sentence that results.⊠

Corollary. There is a decision procedure for each bounded set.

Proof: If S is a bounded set, then it is the extension of a bounded formula $\phi(x)$. To test whether n is in S, check whether $\phi([n])$ is true.⊠

Every bounded set is decidable, but not every decidable set is bounded. To see this, we employ a variant of Cantor's diagonal argument. We can recognize a bounded formula by its syntactic structure, so it is possible to list all the bounded formulas that have "x" as their only free variable. This gives us a list of all the bounded sets. Let C = {n: n is not an element of the

4 To make this work out smoothly, we adopt the convention that the conjunction of $\{\psi\}$ is just ψ, and the conjunction of the empty set of formulas is "$0 = 0$." For the conjunction fo a many-element set of formulas, the order in which the conjuncts are taken and the way they are grouped together can be chosen any way you like. Similarly, the disjunction of $\{\psi\}$ is ψ, and the disjunction of the empty set of formulas is "$\neg\, 0 = 0$."

nth bounded set on the list}. C is decidable. To check whether n is in C, just write out the nth bounded formula on the list and check whether n satisfies it. C is not, however, bounded. For, is C were bounded, then there would be a number k such that C = the kth set of the list. But then we would have:

$k \in$ the kth set on the list

iff $k \in C$ (by the way k was chosen)

iff $k \notin$ the kth set of the list (by the way C was defined)

We can employ the same argument to thwart any attempt to provide a program that generates programs that decide each decidable set. Either some of the programs generated won't be decision procedures (because they will fail to provide answers for some membership questions) or there will be some decidable sets for which no decision procedure is generated.

We can, however, provide a program that list programs that list each effectively enumerable set. For any decidable set S, there will be a program on the list that enumerates S and another program that enumerates the complement of S. Taken together, these two programs provide a program that decides S. Knowing this doesn't provide us with a method for listing decision procedures, because we have no algorithm for matching up the program that enumerates S with the program that enumerates its complement.

In a similar way, we can show that there isn't any program that lists programs for calculating the calculable total functions. Any program that attempts to do this will either list a program for a function that isn't total or it will leave out some calculable total function. To see this, suppose, for *reduction ad absurdum*, that we had such a master program. Define a total function f by:

f(n) = 1 + the output given by the nth program on input n.

Because f is a calculable total function, there will be a program on the list that calculates f; say it's the kth program. Then

the output given by the kth program on input k

= f(k) (by the way k was chosen)

= 1 + the output given by the kth program on input k (by the way f was defined)

Thus the best we can obtain is a program that lists programs that list the calculable partial functions.

The *Σ-formulas* are the formulas obtained by prefixing a block of existential quantifiers to a bounded formula.

Proposition. There is a proof procedure for the set of true Σ sentences.

Proof: Say the sentence is $(\exists v_1)(\exists v_2)...(\exists v_k)\phi$, where ϕ is bounded. The proof procedure is to substitute various k-tuples of numerals for the free variables in ϕ until you get a sentence that's true.⊠

To prove a Σ procedure, all you have to do is to provide a witness. To refute a Σ sentence, you have to shoot down infinitely many potential witnesses, and it's not too surprising that there is no algorithm for doing that. As we shall see later on, the set of true Σ sentences is effectively enumerable but not decidable.

A *Σ set* of natural numbers is the extension of a Σ formula; that is, S is Σ iff there is a Σ formula ϕ such that S = {n: ϕ([n])}. Σ relations are defined similarly.

Corollary. There is proof procedure for each Σ set.

Proof: Say S is the set of numbers that satisfy the Σ formula $\phi(x)$. To enumerate S, start enumerating the true Σ sentences, and add n to the list for S whenever you add $\phi([n])$ to the list of true Σ sentences.⊠

> **Proposition.** If S is a Σ set, then there is a bounded formula $\phi(x,y)$ such that $S = \{x: (\exists y)\phi([x],y)\}$. In other words, to define a Σ set, we don't require a block of existential quantifiers. A single existential quantifier will do.

Proof: If S is a Σ set, then there is a bounded formula $\psi(x,y_1,y_2,...,y_m)$ such that $S = \{x: (\exists y_1)(\exists y_2)...(\exists y_m)\psi([x],y_1,y_2,...,y_m)\}$. Then $S = \{x: (\exists z)(\exists y_1 < z)(\exists y_2 < z)...(\exists y_m < z) \psi([x],y_1,y_2,...,y_m)\}$; the stuff that comes after the "$(\exists z)$" is a bounded formula.⊠

If, instead of starting with a block of existential quantifiers, we start with a block of universal quantifiers,followed by a bounded formula, the result is a Π formula. The block of universal quantifiers can be replaced with a single quantifier. The extension of a Π formula is a Π set. Thus the Π sets are the complements of the Σ sets. A set that is both Σ and Π is said to be Δ.

A confusing terminology has become entrenched. A Δ set or relation is said to be *recursive* iff it's Δ. A set or relation is *recursively enumerable* iff it's Σ. So far so good, but a Σ partial function is referred to as a *partial recursive function.* That's confusing, because it means that a partial recursive function isn't recursive but recursively enumerable. When there's any chance of confusion, I'll try to use "Δ" and "Σ" in place of "recursive" and "recursively enumerable."

Note that the extension of a bounded set is always Σ, since we can tack a vacuous existential quantifier onto the front of the bounded formula. Since the negation of a bounded formula is bounded, it follows that the extension of a bounded formula is always Δ.

The union of two Σ sets is Σ, as is their intersection. If R is a Σ relation, {x: (∃y)Rxy}, {x: (∃y < k)Rxy}, and (x: (∀y < k)Rxy} are all Σ sets, for each k.

Any Σ total function is Δ, as indeed is any Σ partial function with a Δ domain. A set is Δ iff its characteristic function is Δ.

Proposition. The following are equivalent, for any set S of natural numbers:

S is Σ.

S is the domain of a Σ partial function.

There is a Σ partial function f with domain including S such that f(n) = 1 iff n ∈ S.

There is a Δ relation R such that S = {x: (∃y)R([x],y)}.

There is a Σ relation R such that S = {x: (∃y)R([x],y)}.

S is either empty or the range of a Σ total function.

S is the range of a Σ partial function whose domain is an initial segment of the natural numbers.

S is the range of a Σ partial function.

S is either finite or the range of a one-one Σ total function.

Proof: The proofs are analogous to the proofs of the corresponding propositions with "effectively enumerable" in place of "Σ." The only part I want to prove here is that, if S is Σ, then it's either finite or the range of a one-one Σ total function. If S is Σ, it has the form {x: (∃y)ϕ([x],y)}, for some bounded formula ϕ. If S is infinite, we can define a total function with range S as follows:

f(0) = 1st(the least z with ϕ(1st(z),2nd(z)).

f(n+1) = 1st(the least z such that ϕ(1st(z),2nd(z)) and 1st(z) is different from all the f(i)s with $i \leq n$)

In other words,

$f(n) = \text{1st}(\text{the least } z \text{ such that } \phi(\text{1st}(z),\text{2nd}(z)) \wedge (\forall w < z)(\text{either}$

$\neg\phi(\text{1st}(w),\text{2nd}(w)) \text{ or } (\exists i < n)f(i) = \text{1st}(w)))$.

Put a different way, $f(x) = y$ iff there is a finite sequence s of length m that meets the following conditions:

$n < m \rightarrow (\exists z)(\phi(\text{1st}(z),\text{2nd}(z)) \wedge (\forall w < z)(\neg\phi(\text{1st}(w),\text{2nd}(w)) \vee (\exists i<n)(s)_i = \text{1st}(w))$

$\wedge\ \text{1st}(z) = (s)_n)$.

$x < m \wedge (s)_x = y$.

This all gets encoded as a Σ formula.⊠

We have just witnessed a special case of a general technique, devised by Frege and Gödel, for turning recursive definitions into explicit definitions. Let us now describe the technique a little more abstractly. Recursive definitions of functions of one variable take two familiar forms. (For functions of more than one variable, not much changes. The extra variables go along for the ride.)

The more common form of recursive definition occurs when the value of the function only depends on the immediately preceding value of the function. Thus the definition takes the form:

$f(0) = k$

$f(n+1) = g(n+1,f(n))$

where k is a number and g is a (total) function that is already known. We turn this into an explicit definition by stipulating

$f(x) = y =_{\text{Def}}$

$(\exists \text{ finite sequence } s)(\exists m \leq s)(m \text{ is the length of } s \wedge (s)_o = k$

$\wedge$ (for any n with $n+1 < m$, $(s)_{n+1} = g(n+1,(s)_n)$) $\wedge$ $x < m$ and $(s)_x = y$).

If g is Σ, f will be Σ.

The other form, which was the form we witnessed above, is where a value of the function depends on all the earlier values. For f a function, let $f\restriction n$ be the code number for the sequence $<f(0),f(1),...,f(n-1)>$. For s the code number of a sequence of length n or greater, let $s\restriction n$ be the code number for the sequence $<(s)_0,(s)_1,...,(s)_{n-1}>$. I realize I'm using the same symbol for two different purposes, but it should be harmless. Our recursive definition takes the form

$$f(n) = h(f\restriction n),$$

where h is already known. It's explicit version is this:

$f(x) = y =_{Def}$

($\exists$ finite sequence s)($\exists m < s$)(m is the length of s $\wedge$

$(\forall n < m)(s)_n = h(s\restriction n) \wedge x < m \wedge (s)_x = y$).

If h is Σ, so is f.

Proposition (Reduction Principle for Σ sets). For any Σ sets A and B, there are Σ sets C and D such that $C \subseteq A$, $D \subseteq B$, $C \cap D = \varnothing$, and $C \cup D = A \cup B$.

Proposition (Uniformization Priniciple for Σ relations). For any Σ relation R, there is a Σ partial function f with $\text{Dom}(f) = \{x: (\exists y) <x,y> \in R\}$ and $<x,f(x)> \in R$ for each $x \in \text{Dom}(f)$.

I won't give the proofs, which are just like the analogous proofs for effectively enumerable sets and relations.