- link work on logic with work on (recursion) computability: show how we can 'talk about' syntactic notions (e.g. vocabulary, sentence and deduction) in terms of recursive functions
- 15.1-3 are known as the **Arithmetization of Syntax**:
  15.1: The logical operations of negation, disjunction, existential quantification, substitution of a term for free occurrences of a variable, etc. are recursive.
  15.2: the sets of formulas and the set of sentences is recursive.
  15.3: If Γ is a recursive set of sentences, then the relation 'Σ is a deduction of sentence D from Γ' is recursive.
- Corollaries of Arithmetization of Syntax
  15.4 The set of sentences deducible from a given recursive set of sentences is semirecursive.
  15.5: the set of valid sentences is semirecursive (Godel's completeness).
  15.7: Let T be an axiomatizable theory. If T is complete, T is decidable.
- Proofs for 15.2,15.3 provide more inductive evidence for Church thesis.

1 Arithmetization of Syntax (15.1)

- a set/function of symbols/expressions/complicated objects is **recursive** in a transferred or derivative sense iff the set/function of code numbers of elements of the set in question is recursive.
- a language (a set of non-logical symbols) is **recursive** iff the set of code numbers of symbols in the language is recursive.
- a coding of expressions of a formal language by numbers called **Godel numbering**. Further, complicated objects that consist of these expressions can be further coded (e.g. derivations).
- **effectively computable functions**:
  - primitive recursive (identity, constant)
  - non-primitive: obtainable using process for defining new functions from old; exponantiation in terms of multiplication, multiplication in terms of addition, addition in terms of successor, and so on.
- There are many reasonable ways to code finite sequences, and it does not really matter which one we choose. Almost all that matters is that, for any reasonable choice, the following concatenation function will be recursive: $s * t$ = the code number for the sequence consisting of the sequence with code number s followed by the sequence with code number t. This is all that is needed for the proof of the next proposition, in which, as elsewhere in this section, 'recursive' could actually be strengthened to 'primitive recursive'.

Godel Numbers: a unique number ascribed to each logical and non-logical symbols in a language

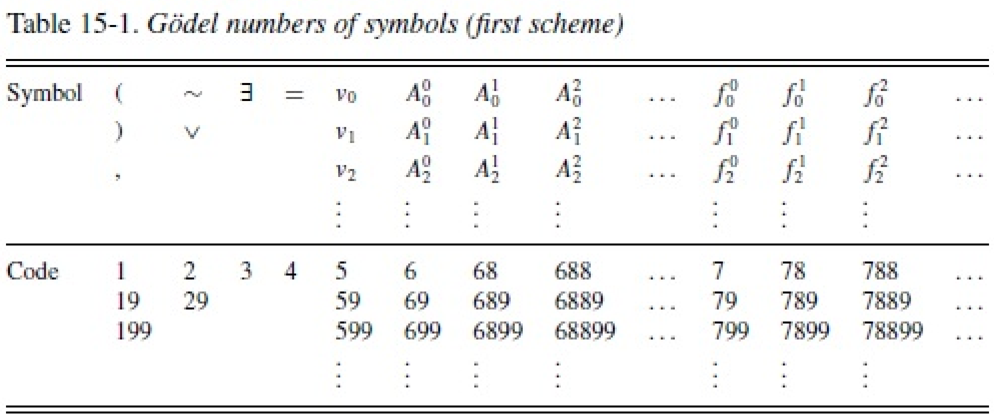

Table 15-1. *Gödel numbers of symbols (first scheme)*

Symbol	(	$\sim$	$\exists$	$=$	v_0	A^0_0	A^1_0	A^2_0	...	f^0_0	f^1_0	f^2_0	...
	)	$\vee$			v_1	A^0_1	A^1_1	A^2_1	...	f^0_1	f^1_1	f^2_1	...
	,				v_2	A^0_2	A^1_2	A^2_2	...	f^0_2	f^1_2	f^2_2	...
					⋮	⋮	⋮	⋮		⋮	⋮	⋮	
Code	1	2	3	4	5	6	68	688	...	7	78	788	...
	19	29			59	69	689	6889	...	79	789	7889	...
	199				599	699	6899	68899	...	799	7899	78899	...
					⋮	⋮	⋮	⋮		⋮	⋮	⋮	

Figure 1: Godel Numbers of Symbols

- Examples for the language of arithmetic
  1. < or A_0^2 has code number 688
  2. 0 or f_0^0 has code number 7
  3. f_0^1 has code number 78
  4. $+$ or f_0^2 has code number 788
  5. $\cdot$ or f_1^2 has code number 7889
- Codes for finite sequences of symbols
  1. $(0 = 0 \vee \neg \mathbf{0} = \mathbf{0})$;
     sequence of symbols with code numbers 1, 7, 4, 7, 29, 2, 7, 4, 7, 19 has code number 174 729 274 719.
- The principle is that if the expression E has code number e and the expression D has code number d, then the expression ED obtained by concatenating them is to have the code number whose decimal numeral is obtained by concatenating the decimal numeral for e and the decimal numeral for d.
- In general the code number for the concatenation of the expressions with code numbers e and d can be obtained from e and d as $e*d = e10lg(d,10)+1+d$, where lg is the logarithm function of Example 7.11.
- For $lg(d,10)+1$ will be the least power z such that $d < 10z$, or in other words, the number of digits in the decimal numeral for d, and thus the decimal numeral for $e10lg(d,10)+1$ will be that for e followed by as many 0s as there are digits in that for d, and the decimal numeral for $e10lg(d,10)+1+d$ will be that for e followed by that for d.

(Prop.1.1) **15.1 Proposition**. The logical operations of negation, disjunction, existential quantification, substitution of a term for free occurrences of a variable, and so on, are recursive.

- *Proof.*
  - Neg. Let n be the code number for the tilde. Let *neg* be the recursive function defined by letting $\text{neg}(x) = n*x$, then if x is the code number for a formula, neg(x) will be the code number for its negation.
    This is what is meant by saying that the operation of negation is recursive.
  - Conj. If l and d and r are the code numbers for the left parenthesis and wedge and right parenthesis, $disj(x,y) = lxdyr$ will be the code number for the disjunction of the formulas coded by x and y.
  - Exst. If e is the code number for the backwards E, then $\text{exquant}(v,x) = e*\vee*x$ will be the code number for the existential quantification with respect to the variable with code number v of the formula with code number x. And similarly for as many other logical operations as one cares to consider. For instance, if officially the conjunction $(X\&Y)$ is an abbreviation for $\neg(\neg X \vee \neg Y)$, the conjunction function is then the composition conj (x, y) = neg(disj (neg(x), neg(y))). The case of substitution is more complicated, but as we have no immediate need for this operation, we defer the proof.

(Prop.1.2) **15.2.** The sets of formulas and of sentences are recursive.

(Prop.1.3) **15.3.** If Γ is a recursive set of sentences, then the relation 'Σ is a deduction of sentence D from Γ' is recursive.

(Cor.1.4) **15.4** the set of sentences deducible from a given recursive set of sentences is semirecursive.

- *Proof*:
  The set of code numbers of sentences deducible from a given recursive set is semirecursive. (Prop 15.3)
  That means that if Γ is recursive, then the relation $Rsd \leftrightarrow d$ is the code number of a sentence and s is the code number of a deduction of it from Γ is recursive.
  And then the set S of code numbers of sentences deducible from Γ, being given by $Sd \leftrightarrow \exists sRsd$, will be semirecursive.

(Cor.1.5) **Godel completeness theorem (abstract)**. The set of valid sentences is semirecursive.

- *Proof*: By the Godel completeness theorem, the set of valid sentences is the same as the set of demonstrable sentences, that is, as the set of sentences deducible from $\Gamma = \emptyset$. Since $\emptyset$ is certainly recursive, it follows from the preceding corollary that the set of valid sentences is semirecursive.

(Cor.1.6) Let Γ be a recursive set of sentences in the language of arithmetic, and D(x) a formula of that language. Then:

(a) The set of natural numbers n such that D(n) is deducible from Γ is semirecursive.
(b) The set of natural numbers n such that $\neg D(n)$ is deducible from Γ is semirecursive.
(c) If for every *n* either $D(n)$ or $\neg D(n)$ is deducible from Γ, then the set of n such that D(n) is deducible from Γ is recursive.

Terminology

- 'Γ **proves**: D is deducible from Γ; written $\Gamma \vdash D$ or $\vdash_\Gamma D$
- **theorems of** Γ: sentences proved by Γ
- **theory**: a set of sentences that contains all sentences of its language that are provable from it
  - Note: theorems of a theory *T* are just the sentences in *T*; written: $\vdash_T B$ or $B \in T$
  - Note: no requirement that any subset of a theory T be singled out as 'axioms'.
- theory *T* is **axiomatizable** if there is a recursive set Γ of sentences s.t. T consists of all and only sentences provable from Γ
- theory *T* is **finitely axiomatizable** if the set Γ is finite
- a set Γ of sentences to be **complete** if for every sentence *B* of its language, either *B* or $\neg B$ is a consequence of Γ, or equivalently, is provable from Γ.
  - Note: theory T is **complete** iff for every sentence *B* of its language, either B or ¬ B is in T
- a set Γ is **consistent** if not every sentence is a consequence of Γ
- a theory *T* is **consistent** if not every sentence of its language is in T.
- A set Γ of sentences is **decidable** if the set of sentences of its language that are consequences of Γ, or equivalently, are proved by Γ, is recursive.
- Note: for a theory *T*, *T* is **decidable** iff T is recursive.

(Cor.1.7) 15.7 Let T be an axiomatizable theory. If T is complete, then T is decidable.

- *Proof.* *T* is decidable.
  'sentence' =abv 'sentence of the language of *T*'.
  Let *T* be an axiomatizable theory i.e. T is the set of sentences provable from some recursive set of sentences Γ. Let $T*$ be the set of code numbers of theorems of *T*. $T*$ is semirecursive. (By Cor 15.4)
  Sub-Proof: $T*$ is recursive. Since, If $T*$ is recursive then *T* is decidable.
  Case 1: $T*$ is the set of all code numbers of sentences, so $T*$ is recursive (By Prop 15.2)
  Case 2: $T*$ is not the set of all code numbers of sentence i.e. not every sentence is a thm of *T*
  Since every sentence would be a theorem of T if for any sentence D it happened that both D and $\neg D$ were theorems of T, for no sentence D can this happen. On the other hand, the hypothesis that T is complete means that for every sentence D, at least one of D and $\neg D$ is a theorem of T. So, D is not a theorem of T iff $\neg D$ is a theorem of T. Hence the complement of $T*$ is the union of the set X of those numbers n that are not code numbers of sentences at all, and the set Y of code numbers of sentences whose negations are theorems of T , or in other words, the set of n such that neg(n) is in $T*$. X is recursive by Proposition 15.2. Y is semirecursive, since it is obtainable by substituting the recursive function neg in the semirecursive set $T*$. So the complement of $T*$ is semirecursive, as was T* itself. That T* is recursive follows by Kleene's theorem (Proposition 7.16) i.e. if a set and its complement are both semirecursive then the set and its complement are recursive.
  So, either way, $T*$ is recursive.

2 Proof of Proposition 15.2 (§15.2 Godel Numbers)

omitted

3 Proof of Proposition 15.3

omitted