

Naming and Diagonalization, from Cantor to Gödel to Kleene

Haim Gaifman

Gödel's incompleteness results apply to formal theories for which syntactic constructs can be given names, in the same language, so that some basic syntactic operations are representable in the theory. It is now customary to derive these results from the fixed point theorem (also known as the reflection theorem), which asserts the existence of sentences that “speak about themselves”. Let \mathbf{T} be the theory and, for each wff ϕ , let $\ulcorner\phi\urcorner$ be the term that serves as its name. Then the theorem says that, for any wff $\alpha(v)$ (with one free variable), there exists a sentence β for which:

$$\mathbf{T} \vdash \beta \leftrightarrow \alpha(\ulcorner\beta\urcorner)$$

β is sometimes called the *fixed point* of $\alpha(v)$. All that is needed for the fixed point theorem is that the diagonal function, the one that maps each $\phi(v)$ to $\phi(\ulcorner\phi(v)\urcorner)$, be representable in \mathbf{T} . The construction of β is more transparent if we assume that the function is represented by a term of the language, $diag(x)$. This means that the following holds for each $\phi(v)$:

$$\mathbf{T} \vdash diag(\ulcorner\phi(v)\urcorner) = \ulcorner\phi(\ulcorner\phi(v)\urcorner)\urcorner$$

(Here ‘=’ is the equality sign of the formal language; we use it also to denote equality in our metalanguage.) In other words, we can prove in \mathbf{T} , for each particular argument, what the function's correct value is; both the argument, $\phi(v)$, and the function's value, $\phi(\ulcorner\phi(v)\urcorner)$, are represented in the formal system via their names. The proof of the fixed point theorem is extremely short:

Let $\alpha^*(v) = \alpha(diag(v))$ and let $\beta = \alpha^*(\ulcorner\alpha^*\urcorner)$. Then $\beta = \alpha(diag(\ulcorner\alpha^*\urcorner))$. By our assumption $\mathbf{T} \vdash diag(\ulcorner\alpha^*\urcorner) = \ulcorner\alpha^*(\ulcorner\alpha^*\urcorner)\urcorner$. The right-hand side of the equality is $\ulcorner\beta\urcorner$. Thus $\mathbf{T} \vdash diag(\ulcorner\alpha^*\urcorner) = \ulcorner\beta\urcorner$. From this we get in pure logic: $\mathbf{T} \vdash \alpha(diag(\ulcorner\alpha^*\urcorner)) \leftrightarrow \alpha(\ulcorner\beta\urcorner)$. The left-hand side of the biconditional is β , hence $\mathbf{T} \vdash \beta \leftrightarrow \alpha(\ulcorner\beta\urcorner)$.

The brevity of the proof does not make for transparency; it has the aura of a magician's trick. How did Gödel ever come up with the idea? As a matter of fact, Gödel did not come up with that idea. The fixed point theorem is not stated in [Gödel 1931]. Instead, Gödel gives a direct construction of a sentence that is analogous to the Liar type sentence, where ‘true’ is replaced by ‘provable’; i.e., a sentence γ for which $\gamma \leftrightarrow \neg Provable(\ulcorner\gamma\urcorner)$ is provable in \mathbf{T} , where ‘*Provable*’ expresses, in the now well-known way, provability in \mathbf{T} . In section 1 I will show how this direct construction follows naturally from Cantor's diagonalization method, via a line of thought suggested in [Richard 1905].

To complete this brief historical note, Carnap in [1934] observed that Gödel's construction can be generalized so that it yields, for any given $\alpha(v)$, a fixed point. In [Gödel 34] Carnap is given credit for this and his work is cited. The generalization, which now appears obvious,

was not at all obvious in 1934. Anyone who will go through Gödel’s original proof will see that achieving the right perspective is not a trivial matter. I therefore suggest that Carnap’s contribution not be forgotten and that the theorem be referred to as the Gödel-Carnap fixed point theorem.

Finally, I should mention that the diagonal function, $\phi(v) \mapsto \phi(\ulcorner \phi(v) \urcorner)$, need not be represented by a term. It is sufficient that it be represented by a wff, say $\delta(x, y)$. This means that, for all $\phi(v)$, $\mathbf{T} \vdash \forall y[\delta(\ulcorner \phi(v) \urcorner, y) \leftrightarrow y = \ulcorner \phi(\ulcorner \phi(v) \urcorner) \urcorner]$. The argument works under the standard translation in which terms are replaced by wffs. We let $\alpha^*(v) = \forall u[\delta(v, u) \rightarrow \alpha(u)]$ and continue as above.

My first goal in this paper is a rational reconstruction of the development of some basic ideas in the history of logic. The natural way to Gödel’s original proof dispels the “magic-trick” aura and probably reflects Gödel’s original train of thought. One should of course be wary of speculations about the way a mathematician gets an idea, the “natural” way is often not the actual one. In the present case there is corroborating evidence from the introductory chapter of [Gödel 1931], where he gives a heuristic outline of the construction. If this outline reflects the way he got the idea of his proof, then his route was the one I shall sketch, or at least one very near to it.

My second goal is a mathematical analysis of self-reference as it is expressed in the fixed point theorem, and a proposal of a simple model that generalizes these phenomena in a useful way. Self-reference via diagonalization depends crucially on the following setup:

- (†) A collection of functions over some domain, D , such that each function has a name that is a member of D ; thus functions can be applied to their own names.

The application of functions to their own names opens possibilities of self-reference in the form of fixed point theorems, provided that some minimal assumptions are satisfied.

The model I shall propose, under the name of *naming system*, covers both the phenomena of sentences that “speak about themselves”, as well as Kleene’s recursion theorem [1938], and others. Kleene stated and proved that theorem by adapting for recursive functions the Gödel-Carnap fixed point technique. In terms of naming systems, both are particular cases of a general fixed point construction. This throws light on the connection between the semantic paradoxes, such as the Liar—which call for a logic with truth-value gaps—and undecidability phenomena in recursion theory—which necessitate the use of *partial* functions. The setup can be applied also to other results obtained via diagonalization, such as the Fisher-Rabin lower bound for Presburger’s arithmetic, [1974]. Naming systems give rise to some technical questions and, I think, merit investigation in their own right. The first two sections that make the first half of the paper are devoted to the conceptual and historical background. Naming system are treated in section 3.

Ramsey [1925] based his proposed reform of *Principia Mathematica* on the now accepted distinction between the set-theoretic paradoxes, which he called logical, and the semantic paradoxes, which he called epistemic. In this he followed Peano [1906] who distinguished between mathematical and linguistic paradoxes. I suggest that the distinction is not as sharp as it appears. Philosophically, the distinction is motivated by the fact that when we make statements we use language, not computations. Hence any account of the Liar says something about language *per se*, including the language in which the account is given. Yet the division is far from absolute. There is a linguistic aspect to recursion theory, or any theory that is broadly concerned with algorithms.

1 From Cantor to Gödel

In [1891] Cantor introduced the diagonalization method in a proof that the set of all infinite binary sequences is not denumerable. He deduced from this the non-denumerability of the set of all reals—something he had proven in [1874] by a topological argument. He refers in [1891] to the earlier work, remarking that the new method is simpler. He goes on to point out that the new method generalizes to two-valued functions defined over an arbitrary set. In [1891] the well-known picture of the infinite two-dimensional array appears for the first time in the form:

$$\begin{array}{l}
 E_1 = (a_{1,1}, a_{1,2}, \dots, a_{1,\nu}, \dots) \\
 E_2 = (a_{2,1}, a_{2,2}, \dots, a_{2,\nu}, \dots) \\
 \dots\dots\dots \\
 E_\mu = (a_{\mu,1}, a_{\mu,2}, \dots, a_{\mu,\nu}), \dots \\
 \dots\dots\dots
 \end{array}$$

Here the E_i 's are binary sequences, enumerated in a sequence E_1, \dots, E_μ, \dots . Cantor then defines $(b_1, \dots, b_\nu, \dots)$, where, for all ν , $b_\nu \neq a_{\nu,\nu}$. This sequence is different from all E_i 's, $i = 1, 2, \dots$

In the case of an arbitrary set S , one considers two-valued functions defined over it; say the values are 0 and 1 (Cantor considers two arbitrary fixed symbols, but this does not matter; from this point we shall not adhere to his original notations.) Suppose that, for each $x \in S$, f_x is a two valued function defined over S . We now define a function g by $g(x) = f_x(x) + 1 \pmod{2}$. Then g is different from all the functions f_x , $x \in S$.

Cantor's original statement is phrased as a non-existence claim: there is no function mapping all the members of a set S onto the set of all 0,1-valued functions over S . But the proof establishes a positive result: given any correlation that correlates functions with members of S , one can construct a function not correlated with any member of S . The construction involves the application of each function, f_x , correlated with x , to its own index. Further developments make use of this form of diagonalization, where the correlation $x \mapsto f_x$ is based on a semantic relation. The members of the domain code (or represent or are indices of) syntactic items that define functions; these are the functions correlated with them. Later, the

syntactic items appear as programs, or tables of Turing machines. In any case, the function, f_x , correlated with x , is the one defined, or described, or determined, or computed by x —or, to adopt a simple term *named* by x .

The first to introduce this move was Richard in [1905]. He argued that since our language (French, in his case) is based on a finite alphabet, one can define a sequence whose members are all finite strings of letters (and spaces), from which one gets the subsequence of all those strings that define positive real numbers. Let u_i , $i = 1, 2, \dots$, be the number defined by the i^{th} definition, and let $f_i(n)$ be the n^{th} member of the decimal expansion of u_i , $i = 1, 2, \dots$ (if there are two expansions choose the one that is 0 from a certain point on). We can now define a number, say u^* , whose decimal expansion is $0.g(1)g(2), \dots$, where g is defined by:

$$g(n) = f_n(n) + 1 \text{ if } f_n(n) < 8, \quad g(n) = 1 \text{ otherwise}$$

u^* is different from all u_i 's (its decimal expansion contains no 0's or 9's and it differs in the i^{th} place from the expansion of u_i). But the description of u^* just given shows that it can be defined in the same language; hence u^* should be equal to some u_i .

Richard's solution to his paradox, stated at the end of his paper, is that the "definition" of g is no definition, since it suffers from vicious circularity: it uses every definition in a sequence in which it already appears as one of the definitions. Richard's solution could have been stated in clearer form, but he, and Poincaré who in [1906] endorsed his solution, made a valid point. If we follow the procedure given in the definition of g , and if g is defined by the m^{th} string of letters that is classified as a definition, we find ourselves in a non-terminating loop: $g(m)$ is defined in terms of $g(m)$ and there is no other way to determine what $g(m)$ is. Peano [1906] classified the paradox as linguistic, not mathematical, a view that is more or less accepted nowadays: 'definition', it is argued, is relative to language and the definition of g is carried out in a language that is on a higher level than the language whose definitions were previously enumerated. What is overlooked in such accounts is the fact that the argument from linguistic levels and the argument from circularity are two sides of the same coin. We can either ascend to higher level languages, or we can keep within the same language paying the price of unavoidable gaps: truth-value gaps, or lack of defined values that results in partial functions. If the second alternative is pursued then indeed, for some m , $g = f_m$, and f_m satisfies an equation of the form: $f_m(x) = h(f_x(x))$, where $h(x) \neq x$ for all x . Putting $x = m$, we get $f_m(m) = h(f_m(m))$; but the contradiction is avoided by the fact that $f_m(m)$ is undefined.

Richard's semantic move can be used in a simpler, more direct way, by applying the technique to the diagonalization that proves that a set is not equinumerous with its power set. This variant (by now the most current one) derives immediately from Cantor's original form. Suppose that f associates with every $x \in S$ a subset $f(x) \subseteq S$. Define the subset S^* by:

$$(1) \quad x \in S^* \Leftrightarrow x \notin f(x)$$

The assumption that, for some z , $S^* = f(z)$, leads immediately to contradiction. (Note

that if we take S to consist of “all objects”, where sets count as objects, and if we take f to be the identity mapping we get Russell’s paradox.¹) Adapting Richard’s idea to this variant, we take S as the set of natural numbers and we consider an enumeration of all definitions of sets of natural numbers; that is, an enumeration of all wffs with one free variable: $\phi_0(v), \dots, \phi_n(v), \dots$. We now define a new set S^* by the condition:

$$(2) \quad n \in S^* \Leftrightarrow n \notin \text{set defined by } \phi_n(v)$$

where ‘ n ’ is a variable (in our own language) ranging over the natural numbers. Then S^* is not defined by any of the enumerated wffs; a paradox similar to Richard’s ensues if we claim that (2) provides a definition in the same language. To get a more accurate view, let us specify the right-hand side of (2) in terms of truth: $n \in \text{set defined by } \phi_n(v)$ iff $\phi(v)$ is true of the number n ; this is equivalent to saying that the sentence $\phi_n(\underline{n})$ is true, where \underline{n} , the numeral of n , is a name of n in the formal language. If the set is definable by $\phi_m(v)$, then the left-hand side can be expressed in the formal language as: $\phi_m(\underline{n})$. Hence, the assumption that the set is definable within the formal language yields m such that, for all n , the following biconditional is true:

$$(3) \quad \phi_m(\underline{n}) \leftrightarrow \neg \text{True}(\phi_n(\underline{n}))$$

where ‘ $\phi_n(\underline{n})$ ’ is a term describing the sentence under the quotes. For $n = m$ we get:

$$(4) \quad \phi_m(\underline{m}) \leftrightarrow \neg \text{True}(\phi_m(\underline{m}))$$

Thus, we get a version of the Liar paradox. Call it D-Liar. Standard Liar sentences achieve self-reference by referring to the sentence through “physical” parameters (‘what I am saying now is false [or not-true]’, or ‘the sentence written in location ℓ is false [or not-true]’). Diagonalization makes it possible to dispense with such extra-mathematical parameters.

The trouble with the semantic paradoxes is, of course, the use of a semantic predicate: ‘defines’—in the case of Richard, ‘is true’—in the case of the D-Liar. This, on one analysis, involves a vicious circle; on any analysis, it is illegitimate. Gödel’s idea was to replace ‘true’ by its syntactic approximation ‘provable’, which could be defined within the same language, if that language were rich enough. By 1931 the notion of a formal system was sufficiently clear to make such a project feasible. With ‘true’ replaced by ‘provable’, (4) becomes the following true biconditional:

$$(5) \quad \phi_m(\underline{m}) \leftrightarrow \neg \text{Provable}(\phi_m(\underline{m}))$$

¹On his own evidence [Russell 1967, p. 147], Russell got the idea of the paradox by reflecting, among other things, on Cantor’s proof. His exact train of thoughts is however not clear.

In the case of (4), each of the assumptions, that the sentence is true, and that its negation is true, leads to contradiction, implying a contradiction altogether. Similarly, if (5) holds, each of the assumptions that the sentence is provable and that its negation is provable leads to contradiction: If $\phi_m(\underline{m})$ is provable, then it is true, and so is the right-hand side of (5); since *Provable*(v) defines the set of provable sentences, $\phi_m(\underline{m})$ is not provable. If $\neg\phi_m(\underline{m})$ is provable, then it is true, hence the negation of the right-hand side is true; thus *Provable*(‘ $\phi_m(\underline{m})$ ’) is true, implying that $\phi_m(\underline{m})$ is provable, hence true; thus both the sentence and its negation are true. This does not lead to contradiction however, only to the conclusion that $\phi_m(\underline{m})$ is undecidable. $\phi_m(\underline{m})$ is also easily seen to be true. The construction of the right-hand side of (5) requires that the syntax of the language and its proofs be definable within the language, hence an arithmetization method is called for.

On the proposed reconstruction, the route to Gödel’s proof is the following:

- (i) Applying Richard’s move to Cantor’s construction, one gets the D-Liar.
- (ii) One hits on the idea of replacing ‘true’ by ‘provable’, realizing that this would yield an undecidable true sentence.
- (iii) The arithmetization of the language is undertaken in order to construct the right hand-side of (5). The implementation of this item takes a large part of Gödel’s proof, involving the messy details of the coding.
- (iv) The proof is tightened by showing that (5) is provable within the formal system, and by eliminating the appeal to the truth concept. The assumption that provability implies truth is replaced by weaker consistency assumptions (consistency—for the non-provability of $\phi_m(\underline{m})$, ω -consistency—for the non-provability of its negation).

One additional item, to complete the picture: the arithmetization of the language is used to enumerate the wffs, namely, ϕ_i is the wff whose Gödel number is i (it does not matter that these numbers constitute a proper subsequence of the natural numbers). Now, the name of α is the name of its Gödel number, $\text{GN}(\alpha)$; that is, $\ulcorner\alpha\urcorner = \text{GN}(\alpha)$. With $i = \text{GN}(\phi_i(v))$, we have: $\phi_i(\underline{i}) = \phi_i(\ulcorner\phi_i(v)\urcorner)$; the right-hand side is the result of applying the familiar diagonal function to $\phi_i(v)$.

The first section of [Gödel 1931] is devoted to an explanation of the paper’s basic ideas. As a formal system Gödel chooses *PM* (*Principia Mathematica*), remarking that Zermelo-Fraenkel set theory would have done equally well. It later emerges that his *PM* is not the actual *Principia Mathematica*—a setup built on the concept of a proposition whose formalization is far from clear—but a certain formal derivative of it: third order Peano arithmetic, with names for 0 and the successor function, and with quantifiable variables of orders 1, 2, 3. The axioms and inference rules are equivalent to what one would expect from Peano arithmetic in such a setting; in particular, it includes induction, and comprehension for the given orders. Gödel considers an enumeration of all wffs with one free numeric variable; he calls them ‘class-signs’,

treating them as definitions of classes of natural numbers. He uses ‘ $R(n)$ ’ for the n^{th} class-sign and ‘ $[R(n); k]$ ’ for the sentence obtained by replacing the free variable of $R(n)$ by the name of k (the usual term built from the 0-sign and the successor sign). He defines the class, K , of all numbers for which $[R(n); n]$ is not provable and observe that K is definable in PM , since all the notions employed in describing it are definable there. Hence, K is defined by some $R(q)$. He now considers the sentence $[R(q); q]$. If the sentence is provable, then it is true; hence $q \in K$, implying, by the definition of K , that $[R(q); q]$ is not provable. If its negation is provable, then $[R(q); q]$ is false; hence $q \notin K$, implying that $[R(q); q]$ is provable.

The construction derives clearly from Cantor’s diagonal method. The enumeration $\phi_0(v), \dots, \phi_n(v), \dots$, which underlies (2), appears here as $R(n)$, $n = 0, 1, 2, \dots$, and the set K is our set S^* , with ‘provable’ playing the role of ‘true’. Gödel notes at the end of his sketch the analogy to Richard’s paradox and the close connection to the Liar, adding in a footnote that “any epistemological antinomy could be used for a similar proof of the existence of undecidable propositions”. The term “epistemological antinomy” is inherited from Ramsey [1925], it means a paradox that employs linguistic elements.

At the end of the introduction he promises that by carrying out the proof in full precision one can get a stronger result, in which the assumption that every provable sentence is true is replaced by a “purely formal” and much weaker one. This will constitute step (iv) in the list above. He also promises surprising results concerning consistency proofs, that is, the second incompleteness theorem. These developments are syntactic and beyond the scope of the present paper. We are concerned with fixed points, in the sense that a biconditional $\beta \leftrightarrow \alpha(\ulcorner \beta \urcorner)$ is true. Kleene’s recursion theorem is of this kind.

Seen in the right perspective, the construction of the self-referential $[R(q); q]$ points to the general prescription for getting a sentence that “ascribes to itself” a property expressed by a given $\alpha(v)$. It is the one given at the beginning of the paper and the one that Kleene adapted for recursion theory.

2 Kleene’s Recursion Theorem

The fixed point technique emerged, as suggested above, as a byproduct of diagonalization. But once a useful technique is discovered it gains an autonomous status, becoming a reference point for further developments; its roots are usually forgotten. This is what happened to the fixed point construction. In 1934 Gödel gave a series of talks at the Institute of Advanced studies in Princeton. C.S. Kleene and J. B. Rosser took notes, which were approved by Gödel and published in [1934]. The notes contain also a definition of *total recursive function*, along lines suggested by Herbrand. It was the first published definition of the list of current equivalent characterizations (what in [Gödel 1931] are called ‘recursive functions’ are primitive recursive functions), though Church [1936] had by 1934 the characterization in terms of λ -definability, cf. [Shoenfield, 1995].

Roughly, a recursive functions is one that is definable by a *functional equation system*: a finite set of equations of the form $t_1 = t_2$, where the t_i 's are terms built from numeric variables, a name for 0, a name for the successor function, and uninterpreted function symbols, one of which is to serve as the name of the defined function. Definability here means that, with \underline{f} serving as the name of the function, every true equality $\underline{f}(\underline{a}_1, \dots, \underline{a}_n) = \underline{b}$, and no other, is derivable by substitution rules from $v = v$ and the given equations, where $\underline{a}_1, \dots, \underline{a}_n, \underline{b}$ are numerals, constructed as usual from the symbols for 0 and successor. The substitution rules allow to substitute any numeric variable by any term, and any occurrence of a term t by a term t' , for which $t = t'$ has been derived. Also a function, f , is *recursive in* a given set, Ξ , of functions, if f is definable by an functional equation system in which we have function symbols for the members of Ξ and can use in the derivation the true equalities $\underline{g}(\underline{c}_1, \dots, \underline{c}_k) = \underline{c}$, for these symbols.

Kleene [1936] applied the arithmetization technique in order to code such systems of equations, as well as derivations. One can then define the function's value, for given arguments, as the one obtained via the smallest coded derivation. In this way, every system of equations defines some partial function. He has thus shown that there is for every n a primitive recursive relation $T_n(z, x_1, \dots, x_n, y)$ such that each n -ary recursive function, f , can be written in the form: $f(x_1, \dots, x_n) = g(\mu y T_n(e, x_1, \dots, x_n, y))$, where g is some fixed primitive recursive function and $\mu y R(\dots y \dots)$ is the smallest y such that $R(\dots y \dots)$. The number e determines the function f ; I shall treat it as a name of f (Kleene speaks of it as 'defining the function'). $\Phi_n(z, x_1, \dots, x_n)$, defined by:

$$\Phi_n(z, x_1, \dots, x_n) = g(\mu y T_n(z, x_1, \dots, x_n, y))$$

is the recursive-function equivalent of a universal Turing machine.

[Kleene 1938] is a short paper, six pages in all, devoted to recursive ordinals. Yet, in part of section 1 and in section 2, which are devoted to the background of recursive functions, Kleene establishes three new points, which are cornerstones of recursion theory. (i) He realizes that the concept of a total recursive function should be replaced by that of a partial recursive function, owing to the fact that it is, in general, impossible to decide whether there exists y such that $T_n(e, x_1, \dots, x_n, y)$. He also proposes a three-valued logic in order to handle partial recursive relations.² (ii) He states and proves the so-called S_n^m theorem: For every $m, n \geq 0$, there is a primitive recursive $m+1$ -ary function S_n^m , such that, for every e that names an $(m+n)$ -ary function, f , $S_n^m(e, a_1, \dots, a_m)$ is a name of $\lambda x_1 \dots x_n f(a_1, \dots, a_m, x_1, \dots, x_n)$. (iii) He states and proves the recursion theorem: Given any $n+1$ -ary recursive function $F(z, x_1, \dots, x_n)$, there exists a number e , such that e is a name of the function $\lambda x_1, \dots, x_n F(e, x_1, \dots, x_n)$. Using (from now on) $\{e\}$ for the function named by e , we can state the claim as the existence of a fixed point: There is e such that, for all x_1, \dots, x_n ,

$$(6) \quad \{e\}(x_1, \dots, x_n) = F(e, x_1, \dots, x_n)$$

²Partial functions have been considered before in [Church 1936], but these were *potentially recursive*, i.e., they had recursive completions. Kleene noted that there are partial functions defined by systems of functional equations that have no recursive completions.

The statements of the two theorems (and, in parentheses, their proofs) are tucked in one short paragraph. The recursion theorem and its proof take two lines; the proof is: Let e_0 be the name of $F(S_n^1(y, y), x_1, \dots, x_n)$ and let $e = S_n^1(e_0, e_0)$. That is all. (Kleene uses other letters and writes ‘define’ instead of ‘be the name of’.)

Kleene relied on the reader to unfold $\{e\}(x_1, \dots, x_n)$ and get (6). He was deliberately simulating the, by that time well-understood, technique of the Carnap-Gödel proof. I shall now propose a setup, which, among other things, abstracts away particular features of the two theorems, so that both come out as special cases of a general theorem.

3 Naming Systems

Naming systems are intended as a basic framework for studying situations in which functions can be applied to their names. Highly expressive formal theories, as well as the setup of recursive functions, are primary sources of such situations. The λ -calculus is another rich system of this kind, where, moreover, the very names are structured entities that act as functions. But these are rich machineries that can do many other things. In a naming system we do not specify how the names are attached to functions, we assume only that there is such a correlation and that it satisfies certain minimal requirements. Thus we can study this aspect—which is manifested differently in different setups—by itself. It is analogous to the focusing on abstract groups, which are realized in different ways in richer structures.

A naming system is a structure of the form:

$$\mathcal{D} = (D, \text{type}(\), \{ \})$$

such that:

- (i) D is a non-empty set.
- (ii) type correlates with each $a \in D$ its type: $\text{type}(a)$. The type tells us if a is a name (of a function) and, if it is, the function’s arity. A name of arity n , or n -ary name, is one that names an n -ary function. Types can be construed as tuples: (0) —if a is not a name, $(1, n)$ — if it is an n -ary name.
- (iii) $\{ \}$ is a mapping that assigns to every n -ary name, a , a function: $\{a\} : D^n \rightarrow D$.

We allow “functions” of arity 0; they are members of D . Thus, if a is a 0-ary name, then $\{a\} \in D$; it is referred to as a *named object*. The subsumption of objects under “functions” might raise philosophical eyebrows. But I am not trying to make a philosophical point. In the present context the move is well motivated, leading to simpler more transparent claims. One may change the terminology and rewrite every statement as a longer, less transparent one, by replacing ‘function’ by ‘function or object’, and splitting various clauses. There is little to recommend this.

We stipulate, as part of the definition, that there is at least one named function of arity > 0 and that the following, referred to as *substitution of names* and *variable permutation*, hold:

(SN) If f is an n -ary named function, where $n > 0$, then, for every name a
 $\lambda x_2 \dots x_n f(a, x_2, \dots, x_n)$ is named.

(VarP) If f is an n -ary named function and π is a permutation of $\{1, \dots, n\}$, then
 $\lambda x_1 \dots x_n f(x_{\pi(1)}, \dots, x_{\pi(n)})$ is named.

The stronger substitution requirement (Sub) is obtained from (SN) by removing the restriction that a is a name. This usually holds. But (SN) is sufficient for our purposes and it fails in some exceptional cases. In the presence of (VarP), the choice of the first coordinate in (SN) is not significant.

(SN) implies that $f(a_1, \dots, a_n)$ is a named object for all names a_1, \dots, a_n . The stronger (SV) implies that all values of named functions are named. We can weaken (SN) by restricting it to functions of arity > 1 . This would make it possible to have naming systems without named objects. Then certain claims will have to be restricted in obvious ways. Under the present definition theorems can be stated in greater generality. Another weak natural requirement, referred to as variable-identification, which we do not assume, is:

(VarI) If f is an n -ary named function then $\lambda x_2 \dots x_n f(x_2, x_2, x_3, \dots, x_n)$ is named.

This usually holds, but it fails in one of the forthcoming examples.

Note: None of these requirements implies that we can add dummy coordinates (e.g., $\lambda x_1 \dots x_n x_{n+1} f(x_1, x_2, x_3, \dots, x_n)$). We *do* want to consider naming systems in which all named functions have arity $\leq n$. From any naming system \mathcal{D} , we can get the system $\mathcal{D} \upharpoonright n$, obtained by retaining only named functions of arity $\leq n$ (names of arity $> n$ remain in D but cease to be names).

Rich systems, such as those arising from arithmetical languages, or recursion theory, satisfy strong additional requirements.

Diagonal Functions

For $n > 0$, an n -diagonal function is a function that maps each n -ary name a to a name of $\lambda x_2 \dots x_n \{a\}(a, x_2, \dots, x_n)$. (Note that by (SN) that second function is named.) Thus, dl_n , satisfies, for every n -ary name a :

$$(7) \quad \{dl_n(a)\}(x_2, \dots, x_n) = \{a\}(a, x_2, \dots, x_n)$$

Since a function can have many names, there can be diagonal functions that assign different values to the same n -ary a (the values are names of the same function). For our purposes it does not matter which diagonal function we use. It also does not matter what values n -diagonal functions assigns to members of D that are not n -ary names. For the sake of convenience we ignore all such differences and speak of *the* n -diagonal function, dl_n . In the following ‘GFP’ stands for ‘General Fixed Point’.

GFP Theorem: If F is an $n+1$ -ary named function, $n \geq 0$, and the composition $F(dl_{n+1}(x_0), x_1, \dots, x_n)$ is named, then there is an n -ary name, e , such that:

$$(8) \quad \{e\}(x_1, \dots, x_n) = F(e, x_1, \dots, x_n)$$

The proof of the theorem merely repeats the standard construction: Let c be a name of $F(dl_n(x_0), x_1, \dots, x_n)$ and let $e = dl_{n+1}(c)$. Then, for $\vec{x} = x_1, \dots, x_n$:

$$\{e\}(\vec{x}) = \{dl_{n+1}(c)\}(\vec{x}) = \{c\}(c, \vec{x}) = F(dl_{n+1}(c), \vec{x}) = F(e, \vec{x})$$

Having abstracted away the particular features of each case, the proof becomes trivial. Our aim at this point is not hard theorems, but a general perspective that yields useful insights. For $n = 0$, the fixed point equation (8) becomes:

$$(8.1) \quad \{e\} = F(e), \text{ where } \{e\} \text{ is an object and } e \text{ is its name.}$$

Note: The assumption that F is named is not needed. But one can hardly find a useful application in which $F(dl_n(x_0), \vec{x})$ is named but F is not. Often, the fact that the composition is named follows from additional assumptions: that dl_n is named and the named functions are closed under compositions. But this is not always so. We *do* want to consider cases in which none of these assumptions hold.

Note that the named functions are total. In the case of partially recursive functions, the functions are completed by adding a “gap-value”. The price for this move is that there cannot be a named function h such that $h(a) \neq a$ for all $a \in D$. This is due to the fact that we have universal recursive functions, namely the functions Φ_n , mentioned in the previous section, which satisfy: $\Phi_n(z, \vec{x}) = \{z\}(\vec{x})$, for every n -ary name z , where $\vec{x} = x_1, \dots, x_n$. (If $n = 0$, \vec{x} is omitted.) This means that $\lambda z \vec{x} (\{z\}(\vec{x}))$ is a named function. The recursive functions are also closed under compositions. Hence, for every monadic named function h , we get a fixed point e such that: $\{e\}(\vec{x}) = h(\{e\}(\vec{x}))$. If $h(a) \neq a$ for all $a \in D$, we would get a contradiction. If the domain D consists of the natural numbers and a single gap-value, \perp , then it can be shown that \perp must be a “sink” for named monadic functions: if h is not a constant numeric function, then $h(\perp) = \perp$. But this leaves more than one option for defining how \perp behaves in various contexts. It also leaves open the possibilities of having many kinds of \perp that mark various kinds of gaps.

Call an $n+1$ -ary named function, $\{e^*\}$, *universal* for n -ary named functions if:

$$(U_n) \quad \{e^*\}(e, \vec{x}) = \{e\}(\vec{x}), \text{ for every } n\text{-ary name } e, \text{ and every } \vec{x} \in D^n$$

The big difference between the framework of formal arithmetical languages (with the Gödel-Carnap theorem) and the framework of recursive functions (with the recursion theorem) is that the first does not have universal functions. In the linguistic setup sentences denote their truth values. A truth-predicate, defined over all the sentences of the language is therefore a universal function for 0-ary names (it can also be used to define universal functions for n -ary names). Since we have a negation function, which toggles truth-values, we can use it as the above h and get a contradiction, unless we have also a gap-value. Thus the handling of \perp for recursive functions and the use of gap-logic for languages that include their truth predicates belong together. Let us first see how formal languages of arithmetic—in first-order, or higher order logic—give rise to naming systems, so that the Gödel-Carnap theorem is an instance of GFP.

Arithmetical Languages

Assume a formal language, interpreted in the domain of natural numbers, which is sufficiently expressive for the purpose of describing its own syntax, via some Gödel numbering. Let $D = N = \text{set of natural numbers}$. Any wff of our language, $\phi(v_1, \dots, v_k)$, whose free variables v_1, \dots, v_k range over N , defines a k -ary relation, which we treat in the usual way as a truth-valued function, where truth-values are identified with 0, 1. (The wff ϕ may have higher order bound variables, if the language has them.) The Gödel number of ϕ names this function. Our naming system consists of all these 0, 1-valued functions. Gödel numbers of sentences name truth-values, i.e., the numbers 0 or 1. These are the only named objects. (The language usually has terms denoting natural numbers, but this is a different matter.)

(SN) holds because there is—for each wff ϕ defining an n -ary relation $R \subseteq N^n$, $n > 0$, and for each natural number a —a wff defining the $(n-1)$ -ary relation $\lambda x_2 \dots x_n R(a, x_2 \dots x_n)$. Usually, the second wff results by substituting in ϕ some variable by \underline{a} , where \underline{a} is a term denoting a .

The assumption of the GFP theorem is always satisfied: for any given wff $\alpha(x_0, x_1, \dots, x_n)$ that defines the $n+1$ -ary relation $R(x_0, \dots, x_n)$, there is a wff defining the n -ary relation $R(dl_{n+1}(x_0), x_1, \dots, x_n)$. The second wff is obtained from the first, either by substituting for x_0 some term, $diag_{n+1}(x_0)$, which represents dl_{n+1} in the language, or by achieving the same effect through the use of some representing wff. Given any wff $\alpha(x_0, x_1, \dots, x_n)$, let F be the 0, 1-valued $n+1$ -ary function named by its Gödel number (i.e., defined by α). The GFP theorem claims the existence of a fixed point, e , satisfying (8). If e is the Gödel number of $\beta(x_1, \dots, x_n)$ then (8) means that the following universally generalized biconditional is true:

$$\beta(x_1, \dots, x_n) \leftrightarrow \alpha(\ulcorner \beta \urcorner, x_1, \dots, x_n)$$

For $n = 0$, the fixed point equation is of the form (8.1) and β is a sentence “ascribing to itself” the property expressed by $\alpha(x_0)$.

The provability of the biconditional: Naming systems *per se* do not represent *provability* aspects. Under some additional assumptions, it is possible to express syntactic features and

define the derivability of equations (a rough indication is given at the end of the paper). In the present case, however, we can see right away what is required for the provability of the biconditional. We need to prove, in our formal theory, the sequence of equalities in the proof of GFP, equalities that appear as biconditionals in the arithmetical language. Consider the case $n = 0$ (where there is no \vec{x}). Assume that the function $dl_1(x)$ is represented in our theory by a term $diag(x)$ and that, for each wff ϕ , $\ulcorner \phi \urcorner$ is a constant term denoting the Gödel number of ϕ . Then the sequence of equalities translates into the sequence of steps in the proof of the syntactic version of the fixed point theorem, given at the beginning of the paper. A proof for the case $n > 0$ is obtained in the same way.

Note: In this modeling the numbers $0, 1$ do double duty: as members of N and as truth-values. All functions are defined over Cartesian powers of N . Hence, an n -ary Boolean function, $f : \{0, 1\}^n \rightarrow \{0, 1\}$, is not a named function; but for k -ary names, e_1, \dots, e_n , the composition $f(\{e_1\}(\vec{x}), \dots, \{e_n\}(\vec{x}))$ is obviously named, being defined by a wff constructed by using sentential connectives. Alternatively, we can model the arithmetical setup by adding truth-values as two new objects. Of this later on.

Recursive Functions

Here and elsewhere “recursive function” means a partial recursive one, unless the context indicates otherwise. Let $D = N \cup \{\perp\}$, where \perp is a “gap-value”.

The names are numbers that code, in some standard way, the algorithms, or the functional equation systems, or the computing devices, which define the function in question. At this stage we do not need to go into details. We assume that the name encodes also the arity of the function; e.g., the coded pair (n, a) is the name of $\lambda x_1, \dots, x_n \Phi_n(a, x_1, \dots, x_n)$. The value is \perp in all cases where the partial recursive function is undefined.

Since $D = N \cup \{\perp\}$, we need to define the functions for cases in which some arguments are \perp . The simplest stipulation is analogous to Frege’s treatment of truth-value gaps:

(F) $\{e\}(\dots, \perp, \dots) = \perp$, i.e., if some argument is \perp , the function’s value is \perp .

It is easily seen that we obtain a naming system and that the conditions of the GFP theorem are satisfied. Thus we get the recursion theorem; the equation of the recursion theorem is the restriction of (8) to numerical values of the arguments x_1, \dots, x_n .

For arity 0 we get (8.1); in this case e names either a number or \perp . In terms of computing devices, e is a code of a machine without inputs, which outputs $F(e)$; if the machine does not halt e names \perp . Note that Kleene’s T_n predicates, the universal functions Φ_n , and the S_n^m theorem make perfect sense for $n = 0$; his [1938] proof covers this case as well, though the range of n is not explicit there. In his book, [1952 p. 342] the S_n^m theorem is stated

for $m, n \geq 0$, but the recursion theorem [p. 352] is stated for $n > 0$. In other books, both theorems are stated for $n > 0$. This is unfortunate, because it hides the complete analogy between the Carnap-Gödel theorem and the recursion theorem.

A more complicated way of treating \perp extends Kleene's handling of truth-value gaps. It requires however more specific computational models. Here is the idea. Let $J \subseteq \{1, \dots, n\}$ and let $\vec{a} \in N^n$. Define $\vec{a}_{J,\perp}$ as the n -tuple obtained by replacing in \vec{a} , every member a_i , where $i \notin J$ by \perp . That is, if $J = \{j_1, \dots, j_k\}$, where $1 \leq j_1 < j_2 < \dots < j_k \leq n$, then $\vec{a}_{J,\perp} = \dots, a_{j_1}, \dots, \perp, \dots, a_{j_2}, \dots, \perp, \dots, a_{j_k}, \dots$. Given an n -ary name, e , define:

(K₀) $\{e\}(\vec{a}_{J,\perp}) = c$, if there is a computation of e , whose output is c , which uses only the inputs in J (i.e., the a_i 's for $i \in J$); otherwise $\{e\}(\vec{a}_{J,\perp}) = \perp$.

The problem is how to define precisely the a computation that "uses only the inputs in J ". In terms of functional equation systems this is done as follows. Let e codes the equation system E , where \underline{f} is the function symbol for the defined function. Let $\vec{v} = v_1, \dots, v_n$, where the v_i 's are distinct variables that can figure in equations. Define:

$$t_i(\vec{a}_J, \vec{v}) = \text{the numeral } \underline{a_i}, \text{ if } i \in J, \text{ the variable } v_i, \text{ if } i \notin J.$$

Thus, $t_1(\vec{a}_J, \vec{v}), \dots, t_n(\vec{a}_J, \vec{v})$ results from $\vec{a}_{J,\perp}$ by replacing the a_i 's by their numerals and the different occurrence of \perp by distinct variables. Then the condition is:

(*) There exists a derivation of $\underline{f}(t_1(\vec{a}_J, \vec{v}), \dots, t_n(\vec{a}_J, \vec{v})) = \underline{c}$, $c \in N$.

Example: Let $e_i, i = 1, \dots, 4$, code the following equation systems for defining multiplication in terms of $+$ (the complete systems include also equations for $+$); for convenience we incorporate the usual notation, with its grouping convention, in the vocabulary of the equations.

$$\begin{aligned} e_1: & \quad 0 \cdot 0 = 0, \quad (v_1 + 1) \cdot v_2 = v_1 \cdot v_2 + v_2, \quad v_1 \cdot (v_2 + 1) = v_1 \cdot v_2 + v_1 \\ e_2: & \quad v_1 \cdot 0 = 0, \quad v_1 \cdot (v_2 + 1) = v_1 \cdot v_2 + v_1 \\ e_3: & \quad 0 \cdot v_2 = 0, \quad (v_1 + 1) \cdot v_2 = v_1 \cdot v_2 + v_2 \\ e_4: & \quad v_1 \cdot 0 = 0, \quad 0 \cdot v_2 = 0, \quad v_1 \cdot (v_2 + 1) = v_1 \cdot v_2 + v_1 \end{aligned}$$

Then our prescription, which consists of (K₀) and (*), gives:

$$\begin{aligned} \{e_1\}(0, \perp) &= \{e_1\}(\perp, 0) = \perp, & \{e_2\}(\perp, 0) &= 0, & \{e_2\}(0, \perp) &= \perp, \\ \{e_3\}(\perp, 0) &= \perp, & \{e_3\}(0, \perp) &= 0, & \{e_4\}(0, \perp) &= \{e_4\}(\perp, 0) = 0. \end{aligned}$$

To complete the story, with notations as above, define the relation $T_{n,J}(z, x_1, \dots, x_n, y)$ to hold just when y is a derivation from the equation system (coded by) z of an equation $\underline{f}(t_1(\vec{x}_J, \vec{v}), \dots, t_n(\vec{x}_J, \vec{v})) = \underline{c}$, where $c \in N$ and \underline{f} is the function symbol for the defined function. It is easily seen that $T_{n,J}$ is primitive recursive. Let g be the function that extracts from a (code of a) derivation the number represented by the right-hand side of the derived equation. Then our definition gives:

$$(K_1) \quad \{e\}(\vec{a}_{J,\perp}) = g(\mu y T_{n,J}(e, \vec{a}, y))$$

With this definition, e is a name provided that it codes a *consistent* functional equation system, where “consistency” means that for every $\vec{a} \in N^n$, there is at most one c for which $\underline{f}(\underline{a}_1, \dots, \underline{a}_n) = \underline{c}$ is derivable. This results in a non-recursive set of names. Nonetheless, we get a naming system, with a corresponding GFP theorem; there is no requirement that the names should form a recursive set. Still, it is desirable to have a primitive recursive set of names and, for this purpose, we adjust the definition so that every code of an equation system names a function. In Kleene’s original setup this was achieved by considering the smallest derivation that gives an output for the given input. In the present case, this is not sufficient, since the smallest derivation is relative to the set J . There might be a smallest derivation of $\underline{f}(\underline{a}_1, v_2) = \underline{c}$ and a smallest derivation of $\underline{f}(v_1, \underline{a}_2) = \underline{c}'$, where $c \neq c'$, which leads to incompatible values for $f(a_1, a_2)$. To avoid this, define two equations $\underline{f}(t_1(\vec{a}_J, \vec{v}), \dots, t_n(\vec{a}_J, \vec{v})) = \underline{c}$, $\underline{f}(t_1(\vec{b}_K, \vec{v}), \dots, t_n(\vec{b}_K, \vec{v})) = \underline{c}'$ to be *incompatible* if (i) $a_i = b_i$ for every $i \in J \cap K$, and (ii) $c \neq c'$. Define $T_{n,J}^*(z, x_1, \dots, x_k, y)$ as:

y is a derivation from z of an equation $\underline{f}(t_1(\vec{x}_J, \vec{v}), \dots, t_n(\vec{x}_J, \vec{v})) = \underline{c}$
and there is no derivation $< y$ of an equation incompatible with it.

It is easily seen that $T_{n,J}^*$ is primitive recursive. Now define:

$$(K) \quad \{e\}(\vec{a}_{J,\perp}) =_{Df} g(\mu y T_{n,J}^*(e, \vec{a}, y))$$

This prescription, it is easy to see, determines a naming system and GFP holds. In the framework of mathematical machines, “using only the inputs in J ” is definable by considering machines whose inputs are pairs, (\vec{a}, J) , where $\vec{a} \in N^n$, $J \subseteq N$. On such an input, the machine can read only the inputs $a_i, i \in J$. Again, there is a consistency requirement, which can be treated in a way similar to the above. There are other available strategies, which accomplish the same.

A third way of treating \perp is to adopt the idea underlying *supervaluation* semantics, which leads to:

$$(sv) \quad \{e\}(\vec{a}_{J,\perp}) = c, \text{ if } \{e\}(\vec{b}) = c, \text{ for every } \vec{b} \in N^n \text{ such that } b_j = a_j \text{ for all } j \in J;$$

$$\text{otherwise } \{e\}(\vec{a}_{J,\perp}) = \perp.$$

For a given J , the condition that the supervaluation value is c , where c and the a_i ’s, $i \in J$, are in N . is therefore:

$$(svc) \quad \text{For all } \vec{x} \in N^n, \text{ if } x_j = a_j \text{ for all } j \in J, \text{ then } \{e\}(\vec{x}) = c.$$

This is a Π_1 condition on $(a_i)_{i \in J}$ and c ; there are cases in which it is not Σ_1 . Hence the resulting system exceeds what is computable. Nonetheless it is naming system for which GFP holds.

We refer to the systems determined, respectively, by (F), (K) and (sv) as the F-system, the K-system and the sv-system. For any name e , let $\{e\}_F, \{e\}_K$ and $\{e\}_{sv}$ be, respectively, the functions named by e in these systems. The instances of GFP in the three systems are different theorems.

Note that the sv-system does not satisfy the variable-identifying requirement. Consider, for example, e such that, $\{e\}(x, y) = x \cdot |x - y|$, for all $x, y \in N$. Then $\{e\}_{sv}(\perp, \perp) = \perp$. Let $g(x) = \{e\}_{sv}(x, x)$; then $g(\perp) = \perp$. But, for all $x \in N$, $g(x) = 0$; hence, we cannot have $g(x) = e'_{sv}(x)$, because this would imply $g(\perp) = 0$. Note that $g(x) = f(I_1(x), I_1(x))$, where $I_1(x)$ is the identity function (which, trivially, is named); hence it also follows that in the sv-system the named functions are not closed under composition. It can be also shown that there is a binary named function $f = \{a\}_{sv}$, such that $\lambda x f(\perp, x)$ is not named: take a such that $\{x \in N : \{a\}_{sv}(\perp, x) \in N\}$ is not r.e. (such a 's exist) and note that, for a monadic e , $\{x \in N : \{e\}_{sv}(x) \in N\}$ is r.e.

The following theorem lists some basic properties of the systems. The proofs are either obvious, or straightforward via standard arguments. We omit them.

Theorem 1 Let e be an n -ary name. (i) The restrictions of $\{e\}_F, \{e\}_K$ and $\{e\}_{sv}$ to N^n are the same. (ii) For $\vec{x} \in D^n$, $\{e\}_{sv}(\vec{x}) = \perp \Rightarrow \{e\}_K(\vec{x}) = \perp \Rightarrow \{e\}_F(\vec{x}) = \perp$; none of the implications is reversible. (iii) $\{e\}_F$ and $\{e\}_{sv}$ are uniquely determined by their restrictions to N^n ; not so (in general) $\{e\}_K$. (iv) All named functions in the F-system are also named in the K-system, but not vice versa; the set of named functions in the sv-system does not include the set of named function of the F-system, and is not included in that of the K-system. (v) $\{e\}_{sv}$ is named in the K-system iff $\{\vec{x} \in N^n : \{e\}_{sv}(\vec{x}) \in N\}$ is recursively enumerable. (vi) The named functions in the F-system and in the K-system, but not in the sv-system, are closed under compositions. (viii) Each of the systems has a named universal function.

Some Significant Properties of Naming Systems

Various properties, which a system may or may not have, are significant. We have mentioned some already. Recall (VarI), closure under variable identification (a rather weak property); or (Sub), obtained from (SN) by removing the restriction on the substituted a . Recall also the existence of universal functions. Here are a few others. Some of these mean that various constructions of named functions are *internal*, i.e., the names of the constructed functions can be obtained by applying named functions.

Named $I_{n,i}$: The n -ary identity function $I_{n,i}(x_1, \dots, x_n) =_{Df} x_i$ is named. (Since we have (VarP), it is sufficient that $I_{n,1}$ is named.)

Closure Under Compositions: If the m -ary f and the n -ary functions $g_i, i = 1, \dots, m$, are named, then $f(g_1(\dots), \dots, g_m(\dots))$ is named. Note that if $I_{n,1}$ is named, then the variable-identification requirement for all n -ary functions is implied by closure

under compositions; e.g., let $\vec{x} = x_1, \dots, x_n$, then $\lambda x_1 x_3 \dots x_n f(x_1, x_1, x_3, \dots, x_n) = f(I_{n,1}(\vec{x}), I_{n,1}(\vec{x}), I_{n,3}(\vec{x}), \dots, I_{n,n}(\vec{x}))$.

Internal Composition: For every m, n there is a named function of arity $m+1$, which—on given names of the functions f, g_1, \dots, g_m (given above)—outputs a name of the composition, (whenever the composition is named).

Named dl_n : The diagonal function dl_n is named.

Internal S_n^m : The function S_n^m is named; this is an $m+1$ -ary function such that for every $(m+n)$ -ary name a , $S_n^m(a, a_1, \dots, a_m)$ names $\lambda x_1 \dots x_n \{a\}(a_1, \dots, a_m, x_1, \dots, x_n)$ (whenever that function is named). Obviously, if (Sub) and (VarI) hold and S_{n-1}^1 is named, then dl_n is named.

Internal Object Naming: There is a named monadic function *name* such that, for every a that has a name, $name(a)$ is a name of a .

3.1 Many Sorted Naming Systems

This variant incorporates distinctions between different data types. The objects are classified into *sorts*. A k -sorted system is of the form:

$$(D_1, \dots, D_k, type(), \{ \})$$

The system's named functions are of the form: $f : D_1^{j_1} \times D_2^{j_2} \times \dots \times D_k^{j_k} \rightarrow D_r$, where $j_i \geq 0$; $\sum_{i=1}^k j_i$ is the function's arity; if $j_i = 0$, there is no argument of sort D_i ; if all j_i 's are 0, this is a member of D_r . The *type* of f is (j_1, \dots, j_k, r) , written also as $(j_1, \dots, j_k; r)$; its *domain type* is (j_1, \dots, j_k) ; its *co-domain* is D_r . As before, $\{a\}$ is the function named by a ; $type(a)$ is the type of $\{a\}$. We say that the the function's type is τ , if it is $(\tau(1), \dots, \tau(k); \tau(k+1))$, where $\tau : \{1, \dots, k+1\} \rightarrow N$. We say that the domain type is σ , when it is $(\sigma(1), \dots, \sigma(k))$; we also put $\tau = \sigma; r$, when τ is the function type with domain type σ and co-domain D_r . We assign sorts to the variables used in our notation, so that a variable of sort D_i ranges over D_i . When writing ' $f(x_1, \dots, x_n)$ ' we assume that the sorts of the x_i 's conform to the type of f .

We assume that all the names belong to a single sort, which is referred to as the system's *main sort*. In our notation this sort is D_1 . One can also consider systems for which this assumption is not made. But the assumption simplifies considerably various definitions and, as far as I can see, does not limit the applications.

The variable permutation requirement (VarP) is imposed for those permutations that preserve the function's type. The substitution requirement (SN) is subject to the obvious restriction that in forming $\lambda x_2 \dots x_n f(a, x_2, \dots, x_n)$ the substituted name, a , is of the main sort.

Diagonal functions are defined as before, but now they depend on function type. For function type τ such that $\tau(1) > 0$, a τ -diagonal function, dl_τ , satisfies:

$$\{dl_\tau(a)\}(x_1, x_2, \dots, x_j) = \{a\}(a, x_2, \dots, x_j),$$

for all names, a , of type τ .

The GFP theorem claims that, for an $n+1$ -ary named F of type τ , if the composition $F(dl_\tau(x_0), x_1, \dots, x_n)$ is named, then there is a name that satisfies the fixed point equation $F(e, x_1, \dots, x_n) = \{e\}(x_1, \dots, x_n)$. The proof is the same as the proof for the one-sorted case.

Various properties of one-sorted systems are generalized in obvious ways to many-sorted ones. For example, identification of variables is now subject to the constraint that the variables be of the same sort; closure under compositions is now subject to obvious type restrictions. (Sub) has to be formulated with respect to each sort; in some cases it is natural to require it for objects of the main sort, leaving other cases open (e.g., when we have a sort of real numbers, cf. below). We shall not enter here into these details. The different sorts are supposed to be disjoint. We can regard the sorts as indexed sets, where the same set can give rise to several copies. An inclusion $D_i \subseteq D_j$ is expressed in the framework by a named function $D_i \xrightarrow{1-1} D_j$.

Here is a simple application of two-sorted systems. Consider again the framework of arithmetical languages. Let $D_1 = N$, $D_2 = \{T, F\}$, where T and F are truth-values (non-numeric objects). Gödel numbers of wffs are names of functions: $D_1^n \rightarrow D_2$. It is possible, though not essential, to have also names for Boolean functions: $D_2^n \rightarrow D_2$. What is important is to have names for definable numeric functions: $D_1^n \rightarrow D_1$. Assume, with no loss of generality, that our language contains terms of the form $\mu v \phi(\vec{u}, v)$, where the term's value, under a given assignment of numbers to the variables \vec{u} , is the smallest value of v satisfying $\phi(\vec{u}, v) \vee [(\neg \exists y \phi(\vec{u}, y)) \wedge v = 0]$. Gödel numbers of terms serve as names for functions: $D_1^n \rightarrow D_1$. The GFP theorem now yields fixed points of two kinds:

For a given wff $\alpha(z, \vec{x})$, a wff $\beta(\vec{x})$ such that: $\forall \vec{x} \{ \beta(\vec{x}) \leftrightarrow \alpha(\ulcorner \beta \urcorner, \vec{x}) \}$ is true.

For a given term $\sigma(z, \vec{x})$, a term $\tau(\vec{x})$ such that $\forall \vec{x} \{ \tau(\vec{x}) = \sigma(\ulcorner \tau \urcorner, \vec{x}) \}$ is true.

Many-sorted systems can be used to model setups involving arbitrary relational structures. For example, consider a 3-sorted system with sorts $D_1 = N$, $D_2 = \{T, F\}$, $D_3 = R$, where R = set of reals. The naming reflects the apparatus that is available for handling these data types. It is represented in N , our main sort, through some ‘‘Gödel numbering’’. The apparatus can, for example, consist in some language, for defining functions over reals; or it can be some computational framework that handles real numbers. Vice versa, naming systems might give rise to computational models for the data types in question. The main sort represents, in general, the syntactic, or computational machinery; named functions over other sorts are those that can be used in definitions, or in computations.

Simultaneous Fixed Point Equations

A system of simultaneous fixed point equations is a system of m equations of the following form, where F_1, \dots, F_m are given $m+n$ -ary functions and $\vec{x} = x_1, \dots, x_n$.

$$\begin{array}{l}
\{e_1\}(\vec{x}) = F_1(e_1, \dots, e_m, \vec{x}) \\
\{e_2\}(\vec{x}) = F_2(e_1, \dots, e_m, \vec{x}) \\
\text{.....} \\
\{e_m\}(\vec{x}) = F_m(e_1, \dots, e_m, \vec{x})
\end{array}
\quad (\text{SE}_{m,n})$$

In the framework of recursion theory the existence of a solution is known, assuming that the F_i 's are (partially) recursive. A similar result holds for arithmetical languages (where the e_i are Gödel numbers of wffs and the equalities correspond to biconditionals). The following is a more structural approach that establishes a general theorem for many-sorted naming systems.

For a given function type τ such that $1 \leq i \leq \tau(1)$, let $dl_{\tau,i}$ be the i^{th} -coordinate diagonal function for τ ; it is defined in the same way as dl_τ , except that the substitution takes place in the i^{th} coordinate. If a is a name of type τ , $dl_{\tau,i}(a)$ is a name of type τ' , where $\tau'(1) = \tau(1) - 1$ and $\tau'(j) = \tau(j)$ for $j > 1$, and the following holds, where l is the arity that goes with τ .

$$(9) \quad \{dl_{\tau,i}(a)\}(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_l) = \{a\}(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_l)$$

SE Theorem Let F_1, \dots, F_m be $m+n$ -ary named functions with the same domain type, σ and with co-domains D_{r_1}, \dots, D_{r_m} , respectively. Assume that $\sigma(1) \geq m$ and that, for all $i = 1, \dots, m$, the following compositions are named, where $\tau_i = \sigma; r_i$.

$$F_i(dl_{\tau_i,1}(x_1), \dots, dl_{\tau_i,m}(x_m), x_{m+1}, \dots, x_{m+n})$$

Then the equation system $(\text{SE}_{m,n})$ has a solution.

Note: In the case of arithmetical languages, modeled as two-sorted naming systems, this implies the existence of solutions to “mixed” equation systems: Some F_i 's are functions into $\{T, F\}$ and the corresponding e_i 's are Gödel numbers of wffs; others are functions into N and the corresponding e_i 's are Gödel numbers of terms. The following proof reduces $(\text{SE}_{m,n})$ to a single equation, by using a non-standard naming.

Proof: For $m = 1$ this is the GFP theorem. We shall now reduce the case $m > 1$ to the case $m = 1$. The idea is best seen if $n = 0$ (i.e., there is no \vec{x} in $(\text{SE}_{m,n})$) and the system is one-sorted. Say the sort is D . Let $\widehat{D} = D^m$. Given m -ary functions $f_i : D^m \rightarrow D$, $i = 1, \dots, m$, define a monadic function, $\langle f_1, \dots, f_m \rangle : \widehat{D} \rightarrow \widehat{D}$, by: $\langle f_1, \dots, f_m \rangle(\vec{b}) =_{Df} (f_1(\vec{b}), \dots, f_m(\vec{b}))$. Now consider the one-sorted system whose sort is \widehat{D} and whose names are all the tuples (a_1, \dots, a_m) such that either (i) all a_i 's are m -ary names, or (ii) all a_i 's are 0-ary. The named function is $\langle \{a_1\}, \dots, \{a_m\} \rangle$; if the arity is 0, this is a member of \widehat{D} . It is easy to verify that this is a naming system. The (monadic) diagonal function for it, \widehat{dl} , satisfies:

$$\widehat{dl}(a_1, \dots, a_m) = (\{a_1\}(a_1, \dots, a_m), \dots, \{a_m\}(a_1, \dots, a_m))$$

Hence it is $\langle dl_{m,1}, \dots, dl_{m,m} \rangle$, where $dl_{m,i}$ is the i^{th} -coordinate diagonal function for m -ary functions of the original system. The composition of $\langle F_1, \dots, F_m \rangle$ with $\langle dl_{m,1}, \dots, dl_{m,m} \rangle$ is $\langle G_1, \dots, G_m \rangle$, where $G_i(\vec{x}) = F_i(dl_{m,1}(\vec{x}), \dots, dl_{m,m}(\vec{x}))$. The claim now follows from the

GFP theorem. If the system is one-sorted, but $n > 0$, we need to use a two-sorted system with a main sort \widehat{D} and an additional sort D . We need D as the sort over which the variables x_1, \dots, x_n of $(SE_{m,n})$ range. The following is the general construction for an arbitrary many-sorted \mathcal{D} and $n \geq 0$.

Let D_1, \dots, D_k be the the sorts. Let $\widehat{D} = D_1^m$. Suppose that f_1, \dots, f_m are l -ary functions, all defined over the domain: $D_1^{j_1} \times D_2^{j_2} \times \dots \times D_k^{j_k}$, where $l = j_1 + \dots + j_k$. Let $\langle f_1, \dots, f_m \rangle$ be the function, whose values are m -tuples, defined as follows:

If $j_1 \geq m$, then $\langle f_1, \dots, f_m \rangle$ is the $(l-m+1)$ -ary function over $\widehat{D} \times D_1^{j_1-m} \times \dots \times D_k^{j_k}$ such that: $\langle f_1, \dots, f_m \rangle((x_1, \dots, x_m), x_{m+1}, \dots, x_l) = (f_1(x_1, \dots, x_l), \dots, f_m(x_1, \dots, x_l))$

If $j_1 < m$, then $\langle f_1, \dots, f_m \rangle$ is the l -ary function over $D_1^{j_1} \times D_2^{j_2} \times \dots \times D_k^{j_k}$ such that: $\langle f_1, \dots, f_m \rangle(x_1, \dots, x_l) = (f_1(x_1, \dots, x_l), \dots, f_m(x_1, \dots, x_l))$. If $l = 0$ this is a member of \widehat{D}

Let D^* be the Cartesian product of the co-domains: $D_{r_1} \times \dots \times D_{r_m}$. Let \widehat{D} be the naming system with $k+2$ sorts: $\widehat{D}, D_1, D_2, \dots, D_k, D^*$ (if $D^* = \widehat{D}$, there are $k+1$ sorts, the repeated occurrence of D^* at the end is omitted), defined by the following specifications. (i) The main sort is \widehat{D} , (ii) $(a_1, \dots, a_m) \in \widehat{D}$ is a name iff the a_i 's are names in \mathcal{D} having the same domain type, of arity $\leq m+n$, and either all are 0-ary names of members of D_1 , or the co-domain of $\{a_i\}$ is D_{r_i} , $i = 1, \dots, m$, (iii) $\{(a_1, \dots, a_m)\} = \{\{a_1\}, \dots, \{a_m\}\}$. Note that a named function has at most one argument (the first) of sort \widehat{D} . In this system, the diagonal function for the type $(1, \sigma(1), \dots, \sigma(k); k+2)$ (or $(1, \sigma(1), \dots, \sigma(k); 1)$, if $D^* = \widehat{D}$) is: $\langle dl_{\tau,1}, \dots, dl_{\tau,m} \rangle$. The rest of the argument is the same as the argument in the one-sorted case with $n = 0$. QED

Conjecture: For every m there is a one-sorted naming system, \mathcal{D} , for which there is an equation system $(SE_{m+1,0})$ without a solution, such that: (i) \mathcal{D} has named identity functions of all arities, (ii) the named functions are closed under compositions, and (iii) \mathcal{D} has a named S_{m-1}^1 (hence a named dl_m). This will show that a named m -diagonal function, dl_m , is not sufficient for solving $m+1$ equations; one should have at least a named $m+1$ -diagonal function, dl_{m+1} . I have proved the conjecture for $m = 1$.

Some Possible Research Lines

The conjecture above and the proof for $m = 1$ have to do with naming systems with limited resources. Various naming system of limited resources come up also in the modeling of small classes of algorithms. For example, the diagonalization that proves the Fischer-Rabin result [1974], about the super-exponential lower bound for Presburger arithmetic, can be derived as an instance of GFP for limited naming systems with non- standard names. I shall not go into these here. On the other hand, the following suggested directions concern systems that are very rich.

Provability: The syntax can be represented by assuming that various syntactic operations are internal. Let com_m be a named function, such that $com_m(e', e_1, \dots, e_m)$ is a name of the function obtained by composition: $\{e'\}(\{e_1\}(\vec{x}), \dots, \{e_m\}(\vec{x}))$. This means that, if $com_m(e', e_1, \dots, e_m) = e$, we can recognize e as naming the composed function; hence, from $\{e_i\}(\vec{a}) = b_i, i = 1, \dots, m, \{e'\}(b_1, \dots, b_m) = b$, we can deduce $\{e\}(\vec{a}) = b$. Connectives and quantifiers—in the case of formal languages—can be treated along these lines. We may use a setup similar to that of Herbrand-Gödel for deriving functional equations. Fixed point equations of the type discussed above will be, as a rule, provable.

General Computability: In a similar, perhaps more straightforward way, we can represent computations. It is well known that proofs and computations are of a kind. A computation of $f(\vec{a})$ that yields the value b is also a proof of: $f(\vec{a}) = b$; a proof of α is also a computation that yields the truth-value T on input α (i.e., establishes $\{\alpha\} = T$). Many-sorted systems described above can be used to model computations with many data types. This was briefly discussed in section 3.1.

Extensions and Gap Values: One can extend a naming system by adding named functions. One can generate such extensions in a systematic way. The addition of a universal function to a system that does not have one will, as a rule, generate gap-values. In the case of formal languages, such an extension is the same as the addition of a truth-predicate. One can also add named oracle functions; e.g., a function that recognizes gap values: $f(\perp) = 0, f(n) = 1$, for $n \in N$, and this will generate additional, “higher order” gap values.

References

- Cantor, G. 1874 “Über eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen” *Crelles J. f. Math*, 77, pp. 258-262. Also in Cantor’s collected papers *Gesammelte Abhandlungen, Mathematischen und Philosophischen Inhalts*, 1932, edited by Zermelo, Berlin, Verlag von Julius Springer, p. 115.
- Cantor, G. 1891 “Über eine elementare Frage der Mannigfaltigkeitslehre” *Jahresbericht der Deutsche Math. Verieng* vol I, pp. 75-78. Also in Cantor’s collected papers, *ibid.* p. 278.
- Carnap, R. 1934 *Logische Syntax der Sprache*, Viena: Springer. Translated into English as *The Logical Syntax of Language*, 1937.
- Church, A 1936 “An Unsolvable Problem of Elementary Number Theory” *The American Journal of Mathematics*, vol. 58. Presented to the Am. Math. Soc. in April 1935.
- Fischer, M and Rabin, M 1974 “Super-Exponential Complexity of Presburger Arithmetic” MAC Technical Memorandum.

- Gödel, K. 1931 “Über formal unentscheidbare Sätze der *Principia Mathematica* und verwandter System I” *Monatshefte für Mathematik und Physik*, 38 pp. 173-198. English translation in *Kurt Gödel’s Collected Works, vol I*.
- Gödel, K. 1934 “On undecidable propositions of formal mathematical systems”, mimeographed lecture notes taken by Kleene and Rosser. In *Kurt Gödel’s Collected Works, vol I*, pp. 338-371.
- Kleene, S. 1936 “General recursive functions of natural numbers” *Mathematische Annalen*, 112, pp. 727-742.
- Kleene, S. 1938 “On notations for ordinal numbers” *Journal of Symbolic Logic*, 3, pp. 15–155.
- Kleene, S. 1952 *Introduction to Methamatematics*, Van Nostrand.
- Peano, G. 1906 “Additione” *Revista de mathematica* 8, pp. 143-157 (follow up on his “Super Theorema de Cantor-Bernstein” in the same volume, pp. 136-143.) Reprinted in his collected works *Opere scelte*, Edizione cremonese, vol.I, 1957.
- Poincaré, H. “Les mathématiques et la logique”, *Revue de metaphysique et de morale* 14, pp. 17-34, 294-317.
- Ramsey, F. 1925 “The Foundations of Mathematics” *Proceedings of the London Mathematical Society*, 25. pp. 338-384. Also in *F.P. Ramsey Philosophical Papers*, ed. Mellor, Cambridge University Press, 1990.
- Richard, J. 1905 “Les Principes des Mathematiques et les problèmes des ensembles” *Revue general des sciences pures et appliqués*, 16, 541, pp. 142-144. Also in English translation in van Heijenoort’s anthology *From Frege to Gödel*
- Russell, B. 1967 *The Autobiography of Bertrand Russell*, published by George Allen & Unwin