Journal of Philosophical Logic (2024) 53:247–291
https://doi.org/10.1007/s10992-023-09731-6

A Step Towards Absolute Versions of Metamathematical Results

Balthasar Grabmayr[1]

Received: 16 October 2021 / Accepted: 6 November 2023 / Published online: 29 November 2023

Abstract
There is a well-known gap between metamathematical theorems and their philosophical interpretations. Take Tarski's Theorem. According to its prevalent interpretation, the collection of all arithmetical truths is not arithmetically definable. However, the underlying metamathematical theorem merely establishes the arithmetical undefinability of a set of specific Gödel codes of certain artefactual entities, such as infix strings, which are true in the standard model. That is, as opposed to its philosophical reading, the metamathematical theorem is formulated (and proved) relative to a specific choice of the Gödel numbering and the notation system. A similar observation applies to Gödel's and Church's theorems, which are commonly taken to impose severe limitations on what can be proved and computed using the resources of certain formalisms. The philosophical force of these limitative results heavily relies on the belief that these theorems do not depend on contingencies regarding the underlying formalisation choices. The main aim of this paper is to provide metamathematical facts which support this belief. While employing a fixed notation system, I showed in previous work (*Review of Symbolic Logic*, 2021, 14(1):51–84) how to abstract away from the choice of the Gödel numbering. In the present paper, I extend this work by establishing versions of Tarski's, Gödel's and Church's theorems which are invariant regarding *both* the notation system and the numbering. This paper thus provides a further step towards absolute versions of metamathematical results which do not rely on contingent formalisation choices.

Keywords Self-reference · Diagonalisation · Incompleteness · Notation systems

✉ Balthasar Grabmayr
balthasar.grabmayr@uni-tuebingen.de

[1] Department of Philosophy, University of Tübingen, Bursagasse 1, 72072 Tübingen, Germany

1 Introduction

1.1 Self-Reference and Notation

In mathematical writings, the choice of the notation system is typically taken to be irrelevant. This attitude is usually based on the idea that notational features are on par with typographical details of English texts. While one font may be more convenient than another for certain purposes, nothing essential depends on this choice. An important exception are self-referential contexts, which are particularly sensitive to the underlying notation system. To illustrate, consider the following two English sentences which only differ with regard to their fonts:

(1) This sentence contains italics.
(2) *This sentence contains italics.*

Here, the choice of font clearly matters, as (1) is false while (2) is true.[1]

Self-reference features centrally in fundamental metamathematical theorems. For example, the standard proofs of Tarski's Theorem and Gödel's First Theorem proceed by constructing arithmetical sentences which deny their own truth and provability respectively. The metamathematical counterparts of such self-referential sentences are fixed points.[2] Let $\ulcorner\cdot\urcorner$ be a naming device, i.e., a mapping from formal expressions to closed terms which serve as their names. Given an arithmetical formula $\varphi(x)$, we call a sentence λ a *fixed point* of φ, if we can establish in some designated theory or model that $\varphi(\ulcorner\lambda\urcorner)$ iff λ. Once such fixed points are established, Tarski's, Gödel's and Church's theorems can be easily derived.[3] Thus, the invariance of these theorems regarding notation systems and numberings reduces to the problem of constructing fixed points for any given notation system and numbering.

In analogy to the informal heuristics above, the choice of a notation system significantly affects the metamathematical study of fixed points. While fixed points are usually constructed by means of diagonalisation, they can also be obtained by Smullyan's [56, 59] method of *normalisation*. Informally, the norm of an expression is the expression followed by its own quotation. Smullyan's method thus formalises the self-referential mechanism which underlies Quine's famous version of the Liar paradox:

> "Yields falsehood when appended to its own quotation" yields falsehood when appended to its own quotation. [51, p. 9]

The success of this method to obtain fixed points highly depends on the employed notation system. Specifically, normalisation yields fixed points for Polish notations,

[1] Another example is given by the sentence "This sentence contains 1 Arabic numeral" and its notational variant "This sentence contains I Arabic numeral" based on Roman numerals.

[2] Here, self-reference is reduced to the weak fixed point property. While this reduction is sufficient for the purposes of this paper, it is too weak to capture "true" self-reference. For a study of the latter the reader is referred to [24, 25, 27, 28, 30, 49].

[3] In this paper, I am concerned with the version of Gödel's Second Theorem which is based on Löb's derivability conditions. By the De Jongh-Sambin fixed point theorem (see, e.g., [4, chapter 8]), this version of Gödel's Second Theorem is in fact equivalent to the (weak) Diagonal Lemma.

but collapses when employing infix notation [59, pp. 87–88]. Similarly, the variant of *near normalisation* [59, p. 86] delivers fixed points for infix notations, but fails for notation systems which only contain well-formed expressions (for examples of such notation systems see below). In other words, Smullyan's construction methods for fixed points are highly sensitive to the employed notation system.

In this paper I will show that *every* method for constructing fixed points necessarily fails for certain deviant notation systems. Even worse, I will introduce notation systems which give rise to provable consistency sentences, definable truth predicates and computable decision procedures, in violation of the famous metamathematical theorems due to Tarski, Gödel and Church. However, I will argue that these deviant constructions are not genuine counterexamples of these theorems, since they rely on inadmissible choices in the formalisation process. By building on work in [22], I will prove the invariance of Tarksi's, Gödel's and Church's theorems with regard to admissible notation systems and admissible Gödel numberings. This work thus provides absolute versions of metamathematical results which do not rely on contingent formalisation choices. In doing so, I make use of a novel method of constructing fixed points which avoids the tedious arithmetisation of the numeral and the substitution function.

The plan of this paper is as follows. In Section 2, I introduce some terminology and give further motivation. Section 3 provides a brief survey of the abundance of notation systems found in the literature. In Section 4, I introduce a general algebraic framework for notation systems. In particular, this framework accommodates all the examples reviewed in Section 3. Section 5 provides deviant notation systems which resist diagonalisation and violate famous metamathematical theorems due to Tarski, Gödel and Church. In Section 6, I put forward admissibility criteria of notation systems which are analysed relative to the given metamathematical context. Finally, I prove the invariance of the syntactic and the semantic fixed point lemma with regard to admissible notation systems and Gödel numberings in Section 7. Moreover, I establish the invariance of important versions of Tarksi's, Gödel's and Church's theorems.

2 Technical and Philosophical Preliminaries

2.1 Arithmetical Languages

We start by introducing formal languages.

Definition 2.1 A *language* $\mathcal{L}$ is a quadruple $\langle \Lambda, \mathcal{F}, \mathcal{R}, a \rangle$ such that

1. $\Lambda, \mathcal{F}, \mathcal{R}$ are pairwise disjoint sets and $a : \mathcal{F} \cup \mathcal{R} \to \omega$ is a function;
2. Λ is the set of the *logical* symbols '$\neg$', '$\wedge$', '$\vee$', '$\rightarrow$', '$\leftrightarrow$', '$=$', '$\forall$', '$\exists$';
3. Each $f \in \mathcal{F}$ is called a *function symbol* of $\mathcal{L}$ of arity $a(f)$.
4. Each $R \in \mathcal{R}$ is called a *relation symbol* of $\mathcal{L}$ of arity $a(R)$.

A function symbol of arity 0 is also called a *constant symbol*. The elements of $\mathcal{F} \cup \mathcal{R}$ are the *non-logical symbols* of $\mathcal{L}$.

According to our definition, the logical symbols of a language are fixed, while the non-logical symbols may vary.[4] In this paper, I am only concerned with finite languages, i.e., languages which only contain finitely many non-logical symbols. An important example is the *language $\mathcal{L}_0$ of Peano Arithmetic* whose non-logical symbols are the constant symbol '0', the unary function symbol 'S' and the binary function symbols '+' and '×'.

Definition 2.2 Let $\mathcal{L} = \langle \Lambda, \mathcal{F}, \mathcal{R}, a \rangle$ and $\mathcal{L}' = \langle \Lambda, \mathcal{F}', \mathcal{R}', a' \rangle$ be languages. We say that $\mathcal{L}$ *is a sublanguage of* $\mathcal{L}'$, in symbols: $\mathcal{L} \subseteq \mathcal{L}'$, if $\mathcal{F} \subseteq \mathcal{F}'$, $\mathcal{R} \subseteq \mathcal{R}'$ and $a(s) = a'(s)$ for all $s \in \mathcal{F} \cup \mathcal{R}$.

Let $A_{\mathcal{L}}$ be the finite alphabet which contains the symbols of $\mathcal{L}$ together with the auxiliary symbols 'v', '′', '(' and ')'. Let $A^*_{\mathcal{L}}$ denote the set of finite strings over $A_{\mathcal{L}}$ and let $*$ denote the concatenation operation on $A^*_{\mathcal{L}}$. For better readability, we often omit the use of quotation symbols and the concatenation operation. For instance, for $s, t \in A^*_{\mathcal{L}_0}$ we write s' instead of $s * '′' $ and $(\mathsf{S}t)$ instead of $'(\mathsf{S}' * t * ')'$, etc.

Fixed points rely on a naming device for formal expressions. We call an injective function $\alpha \colon S \to \omega$ a *numbering of* S. We write $\ulcorner s \urcorner^{\alpha}$ for the standard numeral $\overline{\alpha(s)}$ of the α-code of s.

Definition 2.3 Let $\mathcal{L}$ be a language. An $\mathcal{L}$*-structure* is a pair $\langle D, I \rangle$ such that D is a non-empty set and I is a function that assigns to each k-ary function symbol of $\mathcal{L}$ a function from D^k to D and to each k-ary relation symbol of $\mathcal{L}$ a k-ary relation on D, for $k > 0$. Moreover, I assigns to each 0-ary function symbol an element of D and to each 0-ary relation symbol a designated truth value (say, **T** or **F**).

Definition 2.4 Let $\mathcal{L}, \mathcal{L}'$ be languages with $\mathcal{L} \subseteq \mathcal{L}'$. An $\mathcal{L}'$-structure $\langle D', I' \rangle$ is called an $\mathcal{L}'$*-expansion* of an $\mathcal{L}$-structure $\langle D, I \rangle$, if $D = D'$ and I and I' coincide on the non-logical symbols of $\mathcal{L}$, i.e., I and I' interpret the non-logical symbols of $\mathcal{L}$ in the same way.

Definition 2.5 Let $\mathbb{N}$ denote the standard model of $\mathcal{L}_0$. We call a pair $\langle \mathcal{L}, \mathcal{N} \rangle$ an *interpreted language*, if $\mathcal{N}$ is an $\mathcal{L}$-structure. We say that $\langle \mathcal{L}, \mathcal{N} \rangle$ is *arithmetically interpreted*, if $\mathcal{L} \supseteq \mathcal{L}_0$ and if $\mathcal{N}$ is an $\mathcal{L}$-expansion of $\mathbb{N}$. We also call $\mathcal{N}$ an *arithmetical interpretation* of $\mathcal{L}$.

2.2 Informal vs. Precise Metamathematical Theorems

Tarski is commonly taken to have shown that the collection of all arithmetical truths is not arithmetically definable (see, e.g., [31, p. 43]). This philosophical or informal rendering of Tarski's Theorem can be based on the following metamathematical result:

Claim 2.6 Let $\langle \mathcal{L}, \mathcal{N} \rangle$ be an arithmetically interpreted language and let $\mathrm{Sent}_{\mathcal{L}}$ be the set of $\mathcal{L}$-sentences. The set $\{\varphi \in \mathrm{Sent}_{\mathcal{L}} \mid \mathcal{N} \models \varphi\}$ is not definable in $\mathcal{N}$.

[4] This particular choice of logical symbols will allow for a more convenient presentation. The results of this paper also hold for any other set of logical symbols which contains a quantifier and a functionally complete set of connectives.

The vigilant reader will justifiably complain that Claim 2.6 is still not sufficiently precise to qualify as a metamathematical theorem. This is because neither the set $\mathrm{Sent}_{\mathcal{L}}$ has been fully specified, nor what it means for a subset of $\mathrm{Sent}_{\mathcal{L}}$ to be (un-)definable in $\mathcal{N}$ (at least if that set contains objects different to numbers). A precise definition of $\mathrm{Sent}_{\mathcal{L}}$ requires the choice of a notation system ι for $\mathcal{L}$. Notation systems are based on *data structures*, which consist of designated objects such as strings, trees, sets, etc. together with means for their manipulation. Moreover, a notation system ι specifies the grammatical roles of these data items and thus provides sets Term_{ι}, Fml_{ι} and Sent_{ι} of ι-terms, ι-formulas and ι-sentences respectively (all of this will be made precise in Section 4). For example, the infix notation system inf for $\mathcal{L}$ comes with the set $A^*_{\mathcal{L}}$ of strings over $A_{\mathcal{L}}$, which contains the set $\mathrm{Sent}_{\mathsf{inf}}$ of infix sentences of $\mathcal{L}$.

The second indeterminacy can be eliminated by employing a Gödel numbering. Accordingly, we say that a subset $S \subseteq D$ is *definable in $\mathcal{N}$ relative to a numbering* $\alpha\colon D \to \omega$, if $\alpha(S)$ is definable in $\mathcal{N}$. That is, by using a numbering we reduce definability in $\mathcal{N}$ of sets of non-numbers to definability in $\mathcal{N}$ of sets of numbers, which is defined model-theoretically as usual.[5] For example, let γ be a standard numbering of $A^*_{\mathcal{L}}$ based on prime factorisation. Using infix notation and the numbering γ, we can transform Claim 2.6 into the following precise version of Tarski's Theorem:

Theorem 2.7 *Let $\langle \mathcal{L}, \mathcal{N}\rangle$ be an arithmetically interpreted language. Then the set $\{\varphi \in \mathrm{Sent}_{\mathsf{inf}} \mid \mathcal{N} \models \varphi\}$ is not definable in $\mathcal{N}$ relative to γ.*

While being precise, Theorem 2.7 is no longer sufficiently general to adequately justify the philosophical interpretation of Tarski's Theorem, which is independent of any specific choice of formalisation. In order to overcome this shortcoming, a version of Tarski's Theorem is required which conjoins the generality of Claim 2.6 with the preciseness of Theorem 2.7. In other words, a precise metamathematical result is called for, which establishes the invariance of Tarski's Theorem regarding reasonable choices of the notation system and the numbering.

The importance of formalisation independent metamathematical results has been already observed by Visser [66], though he is sceptical as to whether this can be achieved: "We believe that for all reasonable choices we have [Gödel's second] theorem. However, the quantifier over reasonable choices, is 'unmathematical'. We must articulate just what a reasonable choice is, and this seems scarcely possible" (p. 544). In this paper, I will take up Visser's challenge. Building on previous work [22], I will show how we can abstract away from the choice of the numbering and the notation system in the formulation of Gödel's, Tarski's and Church's theorems, thus obtaining precise metamathematical invariance results.[6]

[5] For a definition of definability see [43, p. 19]. We will see later that the *definiens* does not depend on the choice of the notation system.

[6] This work will not abstract away from all formalisation choices in the case of Gödel's Second Theorem. For an overview of the theorem's dependency on these choices see [7].

3 An Abundance of Notation Systems

While it is evident that the choice of the numbering is highly arbitrary, the contingency resulting from the choice of notation might appear less obvious. In order to convince the reader that also this formalisation choice comes with plenty of alternatives, I briefly survey the abundance of notation systems used in the literature.

3.1 Notation Systems on Strings

Recall that a precise rendering of Claim 2.6 requires the formalisation of arithmetical truths as certain artefactual formal entities, which are typically called *well-formed expressions* of a formal language. Consider, for example, the simple arithmetical truth that

(S) Every natural number has a successor.

One option is to formalise (S) as the infix string

$$(\forall \mathsf{v}(\exists \mathsf{v}'((\mathsf{S}\mathsf{v}) = \mathsf{v}')))$$

as we did in the formulation of Theorem 2.7. However, this choice of notation is highly arbitrary, as there is a multitude of *prima facie* equally adequate alternatives.

To begin with, there are several minor variants of the infix notation system on strings, including omission of the pair of the outermost parentheses, omission of parentheses around strings of the form $\mathsf{S}t$, etc. Moreover, instead of using parentheses, we may alternatively use clusters of dots to disambiguate expressions (see below). A notation system on strings which satisfies unique readability without any use of auxiliary symbols proceeds by placing the "constructor symbols" before its arguments. This notation is called prefix notation or Polish notation, in reference to the nationality of its originator Jan Łukasiewicz. Using Polish notation, (S) can be represented by $\forall \mathsf{v} \exists \mathsf{v}' = \mathsf{S}\mathsf{v}\mathsf{v}'$.[7] Its postfix variant, which proceeds by placing constructor symbols directly after their arguments, is called Reverse Polish notation or postfix notation and yields the formalisation $\mathsf{v}\mathsf{v}'\mathsf{v}\mathsf{S}\mathsf{v}' = \exists \forall$ of (S).

Moreover, the left-to-right direction of written formal language is clearly contingent. It is easy to imagine a different course of history in which we would write ordinary English as usual from left-to-right, but formal expressions in a right-to-left direction, in reverse to today's Arabic or Hebrew writing, where ordinary language is written from right-to-left, while formal mathematical expressions are written from left-to-right. Hence, there are no principled reasons against the adequacy of a "right-to-left" infix notation.[8]

All string notations described above are based on the alphabet $A_{\mathcal{L}}$ which contains a symbol for each quantifier, connective and non-logical symbol of the given language $\mathcal{L}$. This is not without alternative either. For example, using Peano's [47] notation system, which later heavily influenced the notation system of the *Principia Mathematica*, (S)

[7] In this example, quantifiers Q are conceived of as binary constructors which map its arguments x and φ to the string $Qx\varphi$.

[8] This notation system is employed in some Hebrew textbooks of the 19th century [9, Section 5].

can be formalised as $(x).(\exists y).x+ = y$.[9] Here, universal quantification is not implemented by an alphabetical symbol, but by a construction rule based on variables and auxiliary symbols. In a similar vein, Ramsey [52] suggests to represent negation not by an alphabetical symbol such as $\neg$, but by writing the negated expression upside down. According to Ramsey's system, for example, $(\forall \mathsf{v}(\exists \mathsf{v}\prime\ (\prime\mathsf{v} = (\mathsf{vS}))))$ formalises the result of negating the matrix in (S).[10] According to Ramsey "such a symbolism is only inconvenient because we are not trained to conceive complicated symmetry about a horizontal axis" (p. 162), yet there are no principled reasons against the choice of such a notation system (at least in the context of classical reasoning). A consequence of this notation system is that each formula is syntactically identified with its double negation, thus directly incorporating the (classical) logical redundancy of double negation proclaimed by Ramsey [52] and Frege [19].

So far we have considered a number of distinct, *prima facie* equally adequate, notation systems on strings of symbols. But why should strings be treated as a privileged data structure for notation systems in the first place? Surely, strings provide a suitable base for the representation of language, since they allow for the (effective) generation of infinitely many data items from a given (finite) alphabet. For example, any string can be generated by finitely many applications of the concatenation operation to alphabetical symbols. However, in the next section we will see that there are alternative data structures which seem to serve as equally suitable bases for notation systems.[11]

3.2 Algebraic and Tree-theoretical Notation Systems

One may take the ability of strings to faithfully reflect the linearity of spoken or written as a reason to ascribe them a privileged status for the representation of language. However, precisely this proximity to ordinary language is repeatedly taken as a reason *against* the adequacy of language representation based on strings. For example, Dummett notes that "the linear arrangement of the words in a sentence of natural language conceals ... the complexity of the rules which govern the way a sentence may be formed out of its constituent words or other subordinate expressions" [11, p. 2]. Kleene advocates non-linear notation of proofs along similar lines by arguing that "[t]he linear arrangement of proofs has been traditional, no doubt because oral language is necessarily linear, and written language more conveniently so for ordinary purposes" [35, p. 307], however, "trees show the logical structure better, and thus help us in our reasoning about that structure" [ibid]. Two-dimensional notation systems, such as Frege's *Begriffsschrift* [18] or tree notation, exhibit the construction history of the well-formed expressions of $\mathcal{L}$ more directly, since they abstract away from certain artefacts of linear notation. For instance, even though Polish and postfix notation sys-

[9] Peano uses the symbol + both for addition and the successor operation. See [47, p. XVI].

[10] The underlying alphabet must be chosen carefully in order to ensure unique readability. For example, for no alphabetical symbols a, b should a be identical to the result of rotating b along the horizontal axis.

[11] An alternative structure, i.e., generation procedure, for strings can be found in [8]. Instead of using concatenation, strings are constructed by "successor functions" $\mathfrak{s}_a$, for each $a \in A_{\mathcal{L}}$, which append the alphabetical symbol a to any input string.

tems on strings permit unique readability without the use of auxiliary symbols, they still incorporate the ad-hoc convention that the conjunction symbol either precedes or succeeds its conjuncts. These artefacts can be eliminated, once the conjunction of two formulas φ and ψ is formalised as the ordered labelled tree

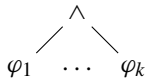
(1)

According to the resulting notation system, well-formed expressions of $\mathcal{L}$ are identified with their parsing trees (this notation system is employed in [63]). Returning to our example, (S) can then be represented by the ordered labelled tree displayed in Fig. 1.

The transition from surface syntax to more abstract notation systems can be pursued even further. To begin with, we may conceive of the distinction between the infix notations $(\varphi \wedge \psi)$ and $(\psi \wedge \varphi)$ as yet another artefact of linear notation, obscuring the fact that both expressions fulfil exactly the same linguistic or logical role in certain contexts. In Frege's [20] words,

> [t]hat "B and A" has the same sense as "A and B" we may see without proof by merely being aware of the sense. Here we have a case where two linguistically different expressions correspond to the same sense. This divergence of expressive symbol and expressed thought is an inevitable consequence of the difference between spatio-temporal phenomena and the world of thoughts. (p. 393)

By employing abstract instead of linear notation, we can to some extent narrow this gap between symbols and thought. Accordingly, the conjunction of φ and ψ (or more accurately, of the unordered pair $\{\varphi, \psi\}$) may be represented by the *un*ordered labelled tree visualised in Eq. 1, thus directly incorporating the commutativity of conjunction into the resulting notation system. Other symmetrical features of the represented language can be implemented similarly, such as the commutativity of identity, disjunction, etc. In yet another step towards abstract syntax, we may conceive of the distinction between the conjunctions of the two unordered pairs $\{\varphi, \{\psi, \chi\}\}$ and $\{\{\varphi, \psi\}, \chi\}$ as another artefact of linearity, veiling the underlying logical role of conjunction. Accordingly, we may represent the conjunction of the unordered tuple $\{\varphi_1, \ldots, \varphi_k\}$, with $k \geq 2$, by the labelled tree



(2)

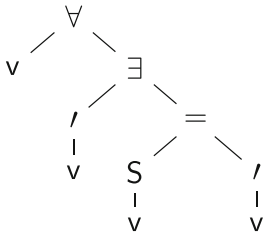


Fig. 1 Parsing tree notation

The resulting notation system thus directly implements both the commutativity and associativity of conjunction.[12]

Notation systems are frequently based on other tree-like structures. Let $\langle\rangle_n$ be the n-ary operation on a set A which is given by $\langle\rangle_n(b_0, \ldots, b_{n-1}) := \langle b_0, \ldots, b_{n-1}\rangle$. The result of closing an alphabet $A_{\mathcal{L}}$ under a given finite number of such operations yields a set of nested lists, which are also called *S-expressions*. The set of S-expressions, together with its constructor operations $\langle\rangle_n$, not only serves as the fundamental data structure for the programming language Lisp, but also provides the basis for notation systems used in metamathematics. While Feferman [15] employs nested ordered pairs, which are obtained by closing $A_{\mathcal{L}}$ under $\langle\rangle_2$, Kleene's *generalized arithmetic entities* [34, §50] result from the closure of $A_{\mathcal{L}}$ under the operations $\langle\rangle_1$, $\langle\rangle_2$ and $\langle\rangle_3$. Kleene then formalises well-formed expressions of $\mathcal{L}$ on nested lists by a prefix system, similarly to the case of Polish string notation. For example, according to Kleene's notation system, (S) can be formalised as the nested list

$$\langle\forall, \mathsf{v}, \langle\exists, \langle\prime, \mathsf{v}\rangle, \langle=, \langle\mathsf{S}, \mathsf{v}\rangle, \langle\prime, \mathsf{v}\rangle\rangle\rangle\rangle.$$

Moving even further towards an abstract algebraic approach, Rasiowa and Sikorski [53], Hájek and Pudlák [26] and Béziau [3] conceive of formal expressions simply as elements of certain absolutely free algebras which are generated by the constructor symbols of $\mathcal{L}$ (see also [28, p. 711]).[13] According to this approach, formal expressions are identified up to isomorphism with elements of the {var, ter, fml}-sorted term algebra $\mathrm{T}_{\Sigma(\mathcal{L})}$ of $\Sigma(\mathcal{L})$, where $\Sigma(\mathcal{L})$ contains suitable constants and constructor symbols for $\mathcal{L}$ (see Definition 4.2 below). For example, (S) can be formalised as the $\Sigma(\mathcal{L})$-term

$$\mathtt{univ}(\mathtt{v}, \mathtt{exists}(\mathtt{nvar}(\mathtt{v}), \mathtt{equ}(\mathtt{fct_S}(\mathtt{e}(\mathtt{v})), \mathtt{e}(\mathtt{nvar}(\mathtt{v})))))$$

of $\mathrm{T}_{\Sigma(\mathcal{L})}$, where $\mathtt{v}$ is a constant symbol of sort var and $\mathtt{univ}$, $\mathtt{exists}$, $\mathtt{equ}$, $\mathtt{fct_S}$, $\mathtt{nextvar}$ and $\mathtt{e}$ are function symbols of $\Sigma(\mathcal{L})$ of suitable type. (For details see Definition 4.2.)

3.3 Notation Systems on Sets and Numbers

The notation systems considered thus far are essentially based on string-theoretical or tree-theoretical data structures. Another approach is to base notation systems on set-theoretical data structures, such as the structure of hereditarily finite sets (this approach can be found in [60, p. 5], [16, section 5.1], [17] and [39, p. 83]). This is usually done by first encoding finite strings or trees as hereditarily finite sets. The well-formed expressions of $\mathcal{L}$ can then be formalised as strings or trees in any old way. Already the

[12] Unordered tuples should here be understood as multi-sets, which distinguish between different occurrences of the same expression. Once we consider sets, i.e., if we no longer distinguish between multiple occurrences of elements, the resulting notation system also directly implements the idempotence of conjunction (see [20, p. 393, footnote 21]).

[13] Béziau [3] is only concerned with propositional expressions, however the scope of his arguments can also be extended to first-order expressions.

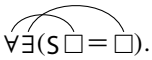
first step of this approach involves arbitrary choices, for instance, regarding the set-theoretical encoding of finite strings or sequences. To begin with, finite sequences may be represented by functions whose domain consist of an initial segment of ω. Alternatively, following Hausdorff [29], finite sequences can be defined recursively as nested ordered pairs, where ordered pairs are defined as $\langle x, y\rangle_H := \{\{x, \varnothing\}, \{y, \{\varnothing\}\}\}$, or by the nowadays more standard definition $\langle x, y\rangle_K := \{\{x\}, \{x, y\}\}$ due to Kuratowski [40]. This approach yields a multiplicity of notation systems on sets. For example, any notation system on strings can be either implemented on functions on initial segments of ω, on finite Hausdorff-sequences, on Kuratowski-sequences, etc.

Even though the resulting notation systems on sets mimic notation systems on strings or trees, set-theoretical notation systems differ from their string- or tree-theoretical counterparts with respect to their underlying data structure. I briefly illustrate the relevance of this difference by considering the notation system which implements finite conjunctions of arbitrary arity greater than 1 outlined above. If such a notation system is based on a tree-theoretical data structure as in Eq. 2, then for each number $n > 1$ a constructor operation for n-ary conjunction is required. Hence, the underlying data structure cannot be finitely generated (see also Appendix A). However, by encoding finite conjunctions as hereditarily finite sets, the resulting notation system can be based on a set-theoretical data structure which is finitely generated. This is because hereditarily finite sets can be generated from the empty set either by the union and singleton operations, or by the adjunction operation $\mathcal{A}$ given by $\mathcal{A}(x, y) := x \cup \{y\}$.

Finally, notation systems can be based on numerical data structures by taking expressions to be numbers. For example, this approach is pursued by Feferman [13, p. 42] and McGee [44, p. 19]. Tree-like notation systems on numbers result from Feferman's [13] or Kleene's [34] approaches. String-like notation systems on numbers result from Gödel's [21] or Smullyan's [58] numberings (identifying expressions with their codes).

3.4 Quantification and Cross Reference

Another aspect of formalisation pertains to the ways cross-reference is achieved in the context of quantification. While traditional notation systems use variables to link quantifiers to their positions in predicate expressions, *Quine-Bourbaki notation* achieves cross-reference by drawing curved lines between quantifier occurrences and their argument positions. This notation was developed by Quine [50, p. 70] and Bourbaki [5, Section 1].[14] Using Quine-Bourbaki notation on strings, for example, (S) can be formalised as



(3)

Similarly, Quine-Bourbaki notation can be implemented on trees, according to which (S) can be represented by the graph given in Fig. 2 (this notation is used in [54] in

[14] Bourbaki uses this notation for Hilbert's epsilon operator rather than quantifiers. Further use and discussion of this notation can be found in [1, section 1.6], [12, Section 4], [62, pp. 13-15], [33, section IX], [55, section 23.4], [65, footnote 7], [6, section 1.4] and [68, Section 4].

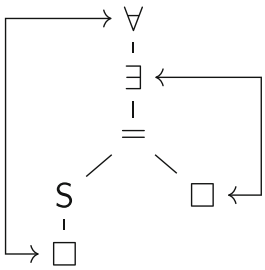


Fig. 2 Quine-Bourbaki tree notation

the context of λ-terms). Both notation systems abstract away from the use of bounded variables, thus directly incorporating the logical redundancy of expressions which are identical up to α-conversion, i.e., up to renaming of bounded variable occurrences. These notation systems therefore do justice to the intuition that "we want to think of bound variables as mere placeholders" and "think of $[\forall x(x = x)]$ as being the same as $\forall y(y = y)$, much the way we think of $\sum_{i=1}^{n} i^2$ and $\sum_{j=1}^{n} j^2$ is being the same sum" [1, p. 21].

Another notation system which dispenses with the use of bounded variables is due to de Bruijn [10] and was initially developed for λ-terms (for an application of this notation in the context of first-order languages see [46]). Instead of drawing lines, here an argument position is uniquely linked with a quantifier occurrence by indicating the number of quantifiers that appear on the unique path in the formula's parsing tree between the argument and quantifier occurrence. The resulting number is called the *de Brujn index* of the considered argument position. For example, when representing (S) in this notation, the de Brujn index of the argument position which corresponds to the universal quantifier is 1, since the existential quantifier occurs in the path between them (see also Fig. 3). Since here expressions are formalised as ordered trees, as

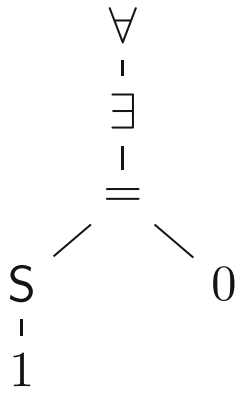


Fig. 3 De Bruijn-index notation

opposed to graphs used in Quine-Bourbaki tree notation, de Bruijn-index notation allows for proofs by tree-induction.

4 A Unified Framework for Notation Systems

Plan of the Paper The previous section provides a multiplicity of, *prima facie* equally adequate, notation systems. Each choice of notation turns Claim 2.6 into a different precise metamathematical statement. In the remainder of this paper, I will examine the extent to which the specific choice of notation system can be abstracted away in the formulation of metamathematical results such as Tarski's Theorem. I will proceed in three steps.

Step 1 The main goal of this paper is to prove that Tarski's, Gödel's and Church's theorems hold for all (reasonable) notation systems. The problem with formulating such invariance claims is that notation systems are not precisely delimited mathematical objects. Hence, quantification over all notation systems is "unmathematical" (see [66, p. 544]). I will solve this problem by introducing a general algebraic framework for notation systems, which models each notation system by a well-defined mathematical object (see Section 4).[15] The introduced framework thus permits a mathematically precise quantification over all notation systems. Moreover, the framework is sufficiently general to accommodate all the examples provided in Section 3.

Step 2 Within this framework, I will introduce deviant notation systems which give rise to provable consistency sentences, definable truth predicates and computable decision procedures, in violation to the famous metamathematical theorems due to Tarski, Gödel and Church (see Section 5). However, I will argue that these constructions are *inadmissible*. Here, I will formulate and motivate mathematically precise notions of admissibility for notation systems (see Section 6).

Step 3 Finally, in Section 7, I will prove that Tarski's, Gödel's and Church's theorems are invariant regarding admissible notation systems (as well as admissible numberings).

4.1 Outline of the Framework

Recall that a language $\mathcal{L}$ consists of a stock of symbols such as '$\wedge$' and '$+$'. Hence, $\mathcal{L}$ merely specifies the basic vocabulary from which complex expressions can be built, but it is left entirely open 1) what kind of objects the well-formed expressions of $\mathcal{L}$ exactly are and 2) how complex expressions are built from simpler ones. The latter two items are specified by a notation system for $\mathcal{L}$.

In order to accommodate as many notation systems as possible, our framework needs to satisfy two desiderata. First, it has to be ontologically neutral, i.e., permit

[15] Algebraic approaches to formal language can be also found in [36] and [48]. However, given the different aims of these authors, the details of their approaches differ from the framework presented in this paper.

expressions to be all kind of objects such as strings, trees, sets, etc. Second, it has to capture all possible ways complex expressions can be built from simpler ones.

In order to satisfy these desiderata, I first define for any given language $\mathcal{L}$ the *proto-expressions* of $\mathcal{L}$ in abstract algebraic terms. Proto-expressions capture the very "essence" of well-formed expressions such as terms and formulas and are as independent from specific implementation details as possible. This will be made precise by taking proto-expressions to be the elements of certain absolutely free algebras (see below).

A notation system for $\mathcal{L}$ can then be seen as an implementation (or projection) of the abstract proto-expressions of $\mathcal{L}$ into a set of designated objects such as strings, trees, sets, etc. Different implementations thus correspond to different construction methods for complex expressions.

4.2 Proto-Expressions

In Section 3, we encountered several ways of forming conjunctions. For example, given two formulas φ and ψ, we may form the infix-conjunction $(\varphi \wedge \psi)$, or the Polish-conjunction $\wedge\varphi\psi$. Both incorporate accidental features which are imposed by the linearity of strings. So neither of them are *the correct* conjunction. Rather, $(\varphi \wedge \psi)$ and $\wedge\varphi\psi$ should be viewed as two different implementations of the *proto-conjunction* of φ and ψ, which itself is as free from specific representational features as possible (see also [28, p. 711]). I will now show how proto-expressions can be obtained by abstracting away from the accidental features that befall systems such as infix and Polish notation.

While the notation systems reviewed in Section 3 incorporate different accidental features, they all have in common that they are *functional* with regard to the relevant subexpressions. For example, the Polish and the infix conjunction-forming operation can be viewed as functions which map any two formulas φ and ψ to the string $\wedge\varphi\psi$ and $(\varphi \wedge \psi)$ respectively. Also the other symbols of $\mathcal{L}$ correspond to certain *constructor operations*. For instance, the infix identity-forming operation maps two terms s and t to the formula $(s = t)$, while the Polish identity-forming operation transforms s and t into $=st$, and so on for the other symbols of $\mathcal{L}$. Together, these constructor operations permit the successive construction of each well-formed expression. Each notation system ι thus corresponds to constructor operations which generate the set of well-formed notations of ι.

According to this algebraic approach, we think of each symbol of $\mathcal{L}$ as a syntactic function transforming expressions of the appropriate grammatical category into compound expressions. In this way, well-formed expressions are successively constructed by operations which correspond to the symbols of $\mathcal{L}$ and which respect the grammatical categories of variables, terms and formulas. In order to make this algebraic approach precise, we map each language $\mathcal{L}$ (in the sense of Definition 2.1) to a many-sorted signature $\Sigma(\mathcal{L})$ only consisting of function symbols. Corresponding to the three grammatical categories of well-formed expressions, $\Sigma(\mathcal{L})$ will distinguish between the sorts var, ter and fml (see Appendix A). For example, each binary connective of $\mathcal{L}$ will correspond to a function symbol in $\Sigma(\mathcal{L})$ of type $\text{fml} \times \text{fml} \to \text{fml}$ (see clause

(8) of the definition below). In addition to containing functional counterparts of the symbols of $\mathcal{L}$, the signature $\Sigma(\mathcal{L})$ will also contain function symbols permitting the generation of variables (see clauses (1)–(3)). This will allow us to view the set of well-formed expressions of $\mathcal{L}$ as a $\Sigma(\mathcal{L})$-algebra.

Remark 4.1 An Ω-algebra simply is an Ω-structure such that the signature Ω exclusively contains function symbols (possibly of arity 0). Throughout this paper, I will denote algebras by bold letters. The domain of an algebra $\mathbf{A}$ is denoted by $|\mathbf{A}|$. For better readability, I will also write $\mathrm{T}_{\Sigma(\mathcal{L})}$ for the domain $|\mathbf{T}_{\Sigma(\mathcal{L})}|$ of the term algebra $\mathbf{T}_{\Sigma(\mathcal{L})}$. See Appendix A for further definitions and conventions.

We now carefully define the mapping Σ:

Definition 4.2 For each language $\mathcal{L} = \langle \Lambda, \mathcal{F}, \mathcal{R}, a \rangle$, we define the $\{\mathrm{var}, \mathrm{ter}, \mathrm{fml}\}$-sorted signature $\Sigma(\mathcal{L})$ as follows:

(1) There is a constant symbol $\mathtt{v}$ of sort var in $\Sigma(\mathcal{L})$;
(2) There is a function symbol $\mathtt{nvar}$ of type var $\to$ var in $\Sigma(\mathcal{L})$;
(3) There is a function symbol $\mathtt{e}$ of type var $\to$ ter in $\Sigma(\mathcal{L})$;
(4) If $f \in \mathcal{F}$, then $\mathtt{fct}_f$ is a function symbol of type $\mathrm{ter}^{a(f)} := \underbrace{\mathrm{ter} \times \cdots \times \mathrm{ter}}_{a(f)-\text{times}} \to \mathrm{ter}$ in $\Sigma(\mathcal{L})$;[16]
(5) There is a function symbol $\mathtt{equ}$ of type ter $\times$ ter $\to$ fml in $\Sigma(\mathcal{L})$;
(6) If $R \in \mathcal{R}$, then $\mathtt{pred}_R$ is a function symbol of type $\mathrm{ter}^{a(R)} \to$ fml in $\Sigma(\mathcal{L})$;
(7) There is a function symbol $\mathtt{neg}$ of type fml $\to$ fml in $\Sigma(\mathcal{L})$;
(8) There are function symbols $\mathtt{conj}, \mathtt{disj}, \mathtt{cond}$ and $\mathtt{bicond}$ of type fml $\times$ fml $\to$ fml in $\Sigma(\mathcal{L})$;
(9) There are function symbols $\mathtt{univ}$ and $\mathtt{exists}$ of type var $\times$ fml $\to$ fml in $\Sigma(\mathcal{L})$;
(10) Nothing else is in $\Sigma(\mathcal{L})$.

We call the elements of $\Sigma(\mathcal{L})$ *constructor symbols* for $\mathcal{L}$. Moreover, we call the fundamental operations of any $\Sigma(\mathcal{L})$-algebra *constructor operations* for $\mathcal{L}$. That is, the constructor operations are the interpretations of the constructor symbols in a given algebra. See Appendix A for further definitions.

So far, we established that the well-formed expressions of a language $\mathcal{L}$ can be viewed as forming a $\Sigma(\mathcal{L})$-algebra. We now provide an abstract characterisation of well-formed expressions which is no longer tied to any accidental features. Let C be the set of constant symbols of $\Sigma(\mathcal{L})$, i.e., C contains $\mathtt{v}$ and $\mathtt{fct}_c$, for each constant symbol c of $\mathcal{L}$. We require a $\Sigma(\mathcal{L})$-algebra $\mathbf{A}$ of well-formed expressions of $\mathcal{L}$ to satisfy the following two conditions:[17]

(1) $\mathbf{A}$ doesn't contain any unintended objects, i.e., objects which cannot be generated by means of the constructor operations for $\mathcal{L}$ from C. More precisely, $|\mathbf{A}|$ is generated by $\mathbf{A}$'s fundamental operations from the set $\{c_{\mathbf{A}} \mid c \in C\}$ of objects that are denoted by the constant symbols of $\Sigma(\mathcal{L})$.

[16] We use the convention that ter^0 is the empty string λ. So in this case $\mathtt{fct}_f$ is a constant symbol of sort ter.

[17] The used terminology is introduced in Appendix A.

(2) The elements of $|\mathbf{A}|$ satisfy unique readability. That is,

- no element $c_{\mathbf{A}}$ is $f(a_1, \ldots, a_n)$, for any fundamental operation f of $\mathbf{A}$, elements $a_1, \ldots, a_n$ of $|\mathbf{A}|$ and $c \in C$ (where $c_{\mathbf{A}}$ is the object in $\mathbf{A}$ denoted by c);
- for all fundamental operations f, g of $\mathbf{A}$ and elements $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$ of $|\mathbf{A}|$, we have that

$$f(a_1, \ldots, a_n) = g(b_1, \ldots, b_m)$$

implies $m = n$, $f = g$ and $a_i = b_i$, for $i = 1, \ldots, n$.

Put more succinctly, condition (1) requires that $\mathbf{A}$ is generated *inductively* by the constructor operations for $\mathcal{L}$ and condition (2) requires that $\mathbf{A}$ is generated *freely* by the constructor operations for $\mathcal{L}$ (see [1, chapter 1]).

These two conditions single out an $\Sigma(\mathcal{L})$-algebra up to (unique) isomorphism.[18] Namely, an $\Sigma(\mathcal{L})$-algebra satisfies (1)–(2) if and only if it is an absolutely free $\Sigma(\mathcal{L})$-algebra. Moreover, for any two absolutely free $\Sigma(\mathcal{L})$-algebras $\mathbf{A}$ and $\mathbf{B}$, there is a unique $\Sigma(\mathcal{L})$-isomorphism mapping $\mathbf{A}$ to $\mathbf{B}$.

The canonical example of an absolutely free algebra is the term algebra $\mathbf{T}_{\Sigma(\mathcal{L})}$ of $\Sigma(\mathcal{L})$. Here, I will only provide a rough description of $\mathbf{T}_{\Sigma(\mathcal{L})}$, while relegating the detailed definition to the appendix (Definition A.5). The domain $\mathrm{T}_{\Sigma(\mathcal{L})}$ of $\mathbf{T}_{\Sigma(\mathcal{L})}$ is the set of closed $\Sigma(\mathcal{L})$-terms, i.e., terms with no variables, and is divided into the sorts var, ter and fml. The set $\mathrm{T}_{\Sigma(\mathcal{L})}$ is sometimes called the Herbrand universe of $\Sigma(\mathcal{L})$. The set $\mathrm{T}_{\Sigma(\mathcal{L})}$ is turned into the $\Sigma(\mathcal{L})$-algebra $\mathbf{T}_{\Sigma(\mathcal{L})}$ by interpreting each constant symbol c of $\Sigma(\mathcal{L})$ as the symbol c itself and by interpreting each function symbol σ of $\Sigma(\mathcal{L})$ as the corresponding term-forming operation. For example, the $\Sigma(\mathcal{L}_0)$-constant symbol $\mathtt{fct_0}$ of sort ter is interpreted in $\mathbf{T}_{\Sigma(\mathcal{L})}$ as $\mathtt{fct_0}$. Moreover, $\mathtt{fct_S}$ is interpreted in $\mathbf{T}_{\Sigma(\mathcal{L})}$ as a unary function that maps, for example, $\mathtt{fct_0}$ to the term $\mathtt{fct_S(fct_0)}$.

We have seen that the elements of $\mathrm{T}_{\Sigma(\mathcal{L})}$ are canonical representatives of $\Sigma(\mathcal{L})$-equivalence classes. Of course, only the latter are entirely free from accidental features. However, in what follows, it will be convenient (albeit of course not necessary) to work with canonical representatives instead of equivalence classes. Therefore, I define the proto-expressions of $\mathcal{L}$ as the elements of the term algebra of $\Sigma(\mathcal{L})$.

Definition 4.3 We call the elements of the term algebra of $\Sigma(\mathcal{L})$ the *proto-expressions of* $\mathcal{L}$, or in short: *p-expressions of* $\mathcal{L}$. In particular, we call $\Sigma(\mathcal{L})$-terms of $T^{\mathrm{var}}_{\Sigma(\mathcal{L})}$, $T^{\mathrm{ter}}_{\Sigma(\mathcal{L})}$ and $T^{\mathrm{fml}}_{\Sigma(\mathcal{L})}$ the *p-variables*, *p-terms* and *p-formulas* of $\mathcal{L}$ respectively (see also Definition A.4).

Remark 4.4 The reader might wonder about clause (3) of Definition 4.2. The intended interpretation of $\mathtt{e}$ is a sortal transfer function which embeds p-variables into p-terms. A different and perhaps more natural way to embed variables into terms is to generalise many-sorted logic to so-called *order-sorted* logic, by defining a partial ordering relation on the set of sorts. The sort var can then be specified at the outset to be a sub-sort of ter, without the use of an artificial transfer function. However, since this

[18] See, e.g., [32, Observation 0.24.5].

framework is not very well known, I retain many-sorted logic and refer the interested reader to [45].

The following remark provides an alternative characterisation of the well-formed expressions of $\mathcal{L}$ in terms of absolutely free $\Sigma(\mathcal{L})$-algebras.

Remark 4.5 An important desideratum for well-formed expressions is that we can define functions on them by means of structural recursion. That is, we want to be able to define a function on well-formed expressions by specifying the values of the constant symbols of $\Sigma(\mathcal{L})$ and by specifying the value of a complex expression $\varphi(t_1, \ldots, \varphi_k)$ in terms of the values of $t_1, \ldots, t_k$.

Each such specification yields a unique and well-defined function iff the set of well-formed expressions forms an absolutely free $\Sigma(\mathcal{L})$-algebra. This is because $\mathbf{A}$ is an absolutely free $\Sigma(\mathcal{L})$-algebra iff $\mathbf{A}$ satisfies the following universal property (see Definition A.3):[19]

For every $\Sigma(\mathcal{L})$-algebra $\mathbf{B}$, there is a unique $\Sigma(\mathcal{L})$-homomorphism $f : \mathbf{A} \to \mathbf{B}$.

Hence, the desideratum of structural recursion provides another (and equivalent) abstract characterisation of the well-defined expressions of $\mathcal{L}$ as absolutely free $\Sigma(\mathcal{L})$-algebras.

4.2.1 Logical Notions and Notations for Proto-Expressions

We can develop syntax for p-expressions in the same fashion as it is done for standard notation systems such as infix strings. The p-variables are given by the ω-sequence

$$\mathtt{v}, \mathtt{nvar(v)}, \mathtt{nvar(nvar(v))}, \mathtt{nvar(nvar(nvar(v)))}, \ldots$$

We will use x, y, z as metavariables ranging over p-variables. As usual, we will presuppose that if, for example, x and y are used in one p-formula, they are distinct.

A p-formula of $\mathcal{L}$ is atomic, if it is of the form $\mathtt{equ}(t_1, t_2)$ or $\mathtt{pred}_R(t_1, \ldots, t_k)$, where $t_1, \ldots, t_k$ are p-terms of $\mathcal{L}$ and R is a k-ary predicate symbol of $\mathcal{L}$. So according to the grammatical rules for p-expressions, only p-terms can feature in atomic p-formulas. For this reason, each p-variable x can be turned into a p-term $\mathtt{e}(x)$ (see Remark 4.4). Clearly, $\mathtt{e}(x)$ and $\mathtt{e}(y)$ are different p-terms for $x \neq y$. For example, the identity statement $\mathtt{equ}(\mathtt{e}(x), \mathtt{e}(y))$ is a p-formula, for any p-variables x and y, while $\mathtt{equ}(x, y)$ is not.

Definition 4.6 Let φ be a p-formula. We call every occurrence of the string $\mathtt{e}(x)$ in φ an occurrence of the p-variable x in φ. We now define what it means that an occurrence of a p-variable is free in a p-formula.

- All occurrences of p-variables in atomic p-formulas are free.
- All occurrences of p-variables that are free in φ are also free in $\mathtt{neg}(\varphi)$.

[19] See [1, pp. 16–7] for a discussion of structural recursion and the universal property, which Avigad aptly calls "recursion theorem".

- All occurrences of p-variables that are free in φ or ψ are also free in $\mathtt{conj}(\varphi, \psi)$, $\mathtt{disj}(\varphi, \psi)$, $\mathtt{cond}(\varphi, \psi)$ and $\mathtt{bicond}(\varphi, \psi)$.
- No occurrence of a p-variable x is free in p-formulas of the form $\mathtt{univ}(x, \varphi)$ and $\mathtt{exists}(x, \varphi)$. All occurrences of p-variables different to x that are free in φ are also free in $\mathtt{univ}(x, \varphi)$ and $\mathtt{exists}(x, \varphi)$.

We say that a p-variable x is free in a p-formula φ, if there is a free occurrence of x in φ. If no occurrence of x is free in φ, we say that x is bound in φ. Let $\mathrm{FV}(\varphi)$ be the set of p-variables that are free in φ. Finally, a p-formula φ is called a p-sentence, if $\mathrm{FV}(\varphi) = \varnothing$.

I will often write $\varphi(x_1, \ldots, x_n)$ instead of φ, if $x_1, \ldots, x_n$ are among the free p-variables in the p-formula φ. In this case, I will also write $\varphi(t_1, \ldots, t_n)$ for the result of simultaneous substitution of the p-term t_i for all free occurrences of x_i in φ, for $i = 1, \ldots, n$. I tacitly assume that substitution is always performable, with bound variables in φ being renamed if necessary. In order to avoid misunderstandings, I will never use the letters 'φ', 'ψ' and 'χ' to denote functions.

Definition 4.7 We define for every $n \in \omega$ the *standard p-numeral* $\mathtt{num}_n$ of n recursively as follows:

$$\mathtt{num}_0 := \mathtt{fct}_0$$
$$\mathtt{num}_{n+1} := \mathtt{fct}_5(\mathtt{num}_n)$$

As usual, we write $\ulcorner \varphi \urcorner^{\alpha}$ for the Gödel numeral $\mathtt{num}_{\alpha(\varphi)}$ of the p-expression φ, where α is some numbering of p-expressions.

We can define certain arithmetical expressions in terms of the vocabulary of $\mathcal{L}_0$ as usual. For example, we take $\mathtt{lessthan}(s, t)$ to abbreviate $\mathtt{exists}(\mathrm{v}, \mathtt{equ}(\mathtt{fct}_+\ (\mathrm{e}(\mathrm{v}), s), t))$, for any given p-terms s and t of $\mathcal{L}_0$.

An $\mathcal{L}$-theory is a set T of p-sentences of $\mathcal{L}$, called the axioms. We say that a p-formula φ is derivable from T, in symbols: $T \vdash \varphi$, if there is a Hilbert-style derivation of φ which only employs the axioms of T and the axioms of predicate logic including the identity axioms.

Important examples of $\mathcal{L}_0$-theories are Peano arithmetic PA and the Tarski-Mostowski-Robinson theory R (see [61, p. 53]). While these theories are typically formulated using infix notation, here the axioms of PA and R are p-sentences. For example, the infix schema $\overline{m} + \overline{n} = \overline{m+n}$ is usually taken as the first axiom schema of R. Here, the corresponding axiom schema of R is given by $\mathtt{equ}(\mathtt{fct}_+(\mathtt{num}_m, \mathtt{num}_n), \mathtt{num}_{m+n})$ and similarly for the other axioms of R and PA.

Let $\mathcal{M}$ be an $\mathcal{L}$-structure and let φ be a p-sentence of $\mathcal{L}$. We define $\mathcal{M} \vDash \varphi$ by adapting the usual Tarskian clauses to p-expressions (see, e.g., [43, p. 11]).

4.3 Notation Systems

In the previous section, I introduced the proto-expressions of a given language $\mathcal{L}$. I will now make the idea precise that a notation system for $\mathcal{L}$ is an implementation of

the abstract proto-expressions of $\mathcal{L}$ into some data structure, that contains designated objects such as strings or trees. A notation system will thus be specified by two ingredients: a data structure and an implementation. While the data structure delivers the linguistic raw expressions as well as means for their manipulation, the implementation maps each p-expression to some raw expression, which serves as its notation.

4.3.1 Data Structures

Data structures serve as fundamental concepts in the study of programming languages in computer science.[20] In order to satisfy our desideratum of ontological neutrality, a data structure may consist of any kind of objects, including (types of) acoustic utterances or inscriptions, nested lists, trees, graphs, numbers, sets, etc. What all these objects have in common, is that they can be generated from designated basic objects by means of finitely many operations.[21] For example, strings over some finite alphabet A can be constructed by successively concatenating the alphabetical symbols of A, trees can be constructed by certain tree-forming operations, and so on. A data structure can roughly be characterised as consisting of a finite collection of data domains, designated basic data items and fundamental operations on the data domain, such that all data items of the data domains can be accessed from the basic data items by use of the fundamental operations. The following definition makes this idea precise.

Definition 4.8 A *data structure* is a many-sorted finitely generated algebra (see Definition A.1).

For an example of a data structure that is not single-sorted see Example A.2.

Remark 4.9 Why are notation systems based on finitely generated algebras and not merely on unstructured sets? As I will show in Section 4.4, the additional algebraic structure ensures that we have robust and well-defined notions of computability and definability for functions over arbitrary data structures. These notions will later play an important role in the analysis of admissible notation systems (see Section 6).

By providing the "raw material" of the notation system, a data structure specifies what kind of objects notations are. Without further specification, this raw material is syntactically idle. For instance, the question whether the string $=00$ is a well-formed formula can only be meaningfully asked relative to a notation device. To illustrate, $=00$ is well-formed w.r.t. Polish notation, but not well-formed w.r.t. infix notation. Hence, in addition to providing a data structure $\mathbf{D}$, we also need to specify the way p-expressions are implemented into $\mathbf{D}$. If, as in our example, the data structure comprises strings, different implementations correspond to different notations on strings, such as Polish or infix notation. In the next section, we make this idea precise.

[20] For an algebraic treatment of data structures in computer science see [2].

[21] The introduced framework is not capable to represent expressions as tokens of utterances or inscriptions. This is because the underlying generation procedures correspond to total operations on the given data structure. For instance, for any two formula notations of a given data structure there exists a notation which represents their conjunction. While this principle is satisfied by types of utterances or inscriptions, it fails when we consider tokens. In particular, concrete physical objects can only serve as notations if there are infinitely many of them, since any data structure employed in the presented framework is countably infinite.

4.3.2 Implementations of Proto-Expressions

Let $\mathcal{L}$ be a language and let $\mathbf{D}$ be a fixed data structure. Recall that the algebra $\mathbf{D}$ may have different sorts. We write $\bigcup|\mathbf{D}|$ for the set of all elements of $\mathbf{D}$'s domain, irrespective of their sort. See also Definition A.1.

An implementation of the p-expressions of $\mathcal{L}$ into $\mathbf{D}$ is a function

$$\iota\colon \mathrm{T}_{\Sigma(\mathcal{L})} \to \bigcup|\mathbf{D}|,$$

which preserves the syntactic structure of p-expressions and a modicum of logic (I explain what that means in a moment).

Remark 4.10 Since $\mathrm{T}_{\Sigma(\mathcal{L})}$ is a many-sorted set, ι is actually a family of functions $\langle \iota^{\mathrm{var}}, \iota^{\mathrm{ter}}, \iota^{\mathrm{fml}} \rangle$, such that ι^{var} maps p-variables to elements of the domain $\bigcup|\mathbf{D}|$ of $\mathbf{D}$, ι^{ter} maps p-terms to elements of $\bigcup|\mathbf{D}|$ and ι^{fml} maps p-formulas to elements of $\bigcup|\mathbf{D}|$. For the sake of better readability, I will often write ι instead of $\langle \iota^{\mathrm{var}}, \iota^{\mathrm{ter}}, \iota^{\mathrm{fml}} \rangle$. Any specification of a function ι from p-expressions to elements of a data structure should thus be officially thought of as specifying three functions ι^{var}, ι^{ter} and ι^{fml}, defined on p-variables, p-terms and p-formulas respectively. See Appendix A for a careful introduction of the many-sorted setting.

Let now ι be such a mapping of p-expressions to elements of $\bigcup|\mathbf{D}|$. We then call $\iota(\varphi)$ the *ι-implementation of φ into $\boldsymbol{D}$*, which is also denoted by φ_ι. We also call φ_ι a *ι-notation*, or more specifically, a *ι-variable*, *ι-term* or *ι-formula*, depending on whether φ is a p-variable, p-term or p-formula (we will see in a moment that this is well-defined).

An implementation ι will not be required to be injective, i.e., different p-expressions may be implemented as the same object in $\mathbf{D}$. An example of a non-injective implementation is given by the Quine-Bourbaki-notation (see Appendix B). We call two p-expressions φ and ψ *ι-equivalent*, in symbols: $\varphi \approx_\iota \psi$, if $\iota(\varphi) = \iota(\psi)$. Clearly, $\approx_\iota$ is an equivalence relation, which we call *ι-equivalence*.

When implementing p-expressions into $\mathbf{D}$, it is essential that the grammatical structure of p-expressions is preserved by the resulting ι-notations and that ι-equivalence preserves a modicum of logic. More specifically, a function ι qualifies as an implementation if the following three requirements are satisfied.

First, we require the syntactic categories of ι-variables, ι-terms and ι-formulas to be well-defined. In other words, only p-expressions of the same syntactic category can be mapped to the same object in $\mathbf{D}$. More formally, for any $\varphi \in T^s_{\Sigma(\mathcal{L})}$ and $\psi \in \mathrm{T}^t_{\Sigma(\mathcal{L})}$ we have that $\varphi \approx_\iota \psi$ implies $s = t$. That is, ι-equivalent p-expressions are of the same sort.

Second, all the constructor operations for $\mathcal{L}$ must be performable on ι-notations. For example, the conjunction of two ι-formulas needs to be well-defined. That is, we require that ι-equivalent expressions yield ι-equivalent outputs when applied to the fundamental operations of the term algebra $\mathbf{T}_{\Sigma(\mathcal{L})}$. More formally, for every k-ary

fundamental operation f of $\mathbf{T}_{\Sigma(\mathcal{L})}$ and p-expressions $\varphi_1, \ldots, \varphi_k$ and $\psi_1, \ldots, \psi_k$ of suitable type we have that

$$\varphi_i \approx_\iota \varphi_i \text{ for } i = 1, \ldots, k \text{ implies } f(\varphi_1, \ldots, \varphi_k) \approx_\iota f(\psi_1, \ldots, \psi_k).$$

The first and the second requirement are satisfied iff ι-equivalence is a *congruence relation* on $\mathbf{T}_{\Sigma(\mathcal{L})}$.

Third, we require that it is well-defined to say that a ι-formula has a certain number of free variables. That is, any two ι-equivalent p-formulas are required to have the same number of free variables. As a result of this requirement, the grammatical category of ι-sentences is well-defined, namely, as the set of ι-formulas which do not have any free variables.

Fourth, we require that substitution of ι-terms for ι-variables in a ι-formula is well-defined.

Finally, we require that any two ι-equivalent p-sentences are logically equivalent.

We summarise these minimal requirements for implementations in the following definition:

Definition 4.11 Let $\mathbf{D}$ be a data structure. We call a function $\iota\colon \mathrm{T}_{\Sigma(\mathcal{L})} \to \bigcup|\mathbf{D}|$ an *implementation of p-expressions of $\mathcal{L}$ into $\boldsymbol{D}$*, if the following holds:

(1) $\approx_\iota$ is a congruence relation on $\mathbf{T}_{\Sigma(\mathcal{L})}$;
(2) If $\varphi \approx_\iota \psi$, then $\#\,\mathrm{FV}(\varphi) = \#\,\mathrm{FV}(\psi)$ for all p-formulas φ, ψ;
(3) If $\varphi \approx_\iota \psi$ and $\#\,\mathrm{FV}(\varphi) = 0$, then $\vdash \mathtt{bicond}(\varphi, \psi)$, for all p-formulas φ, ψ;
(4) If $\varphi \approx_\iota \psi$, $\#\,\mathrm{FV}(\varphi) = k$ and $s_i \approx_\iota t_i$ for all $i < k$, then

$$\varphi(s_0, \ldots, s_{k-1}) \approx_\iota \psi(t_0, \ldots, t_{k-1}),$$

for all p-terms s_i, t_i and p-formulas φ, ψ.

Finally, we call $\langle \mathbf{D}, \iota \rangle$ a *notation system for $\mathcal{L}$*, if $\mathbf{D}$ is a data structure and ι is an implementation of p-expressions of $\mathcal{L}$ into D.

All the examples exhibited in Section 3 can be viewed as notation systems in the sense of Definition 4.11. A worked-out example for the Quine-Bourbaki notation system can be found in Appendix B.

Remark 4.12 The term algebra $\mathbf{T}_{\Sigma(\mathcal{L})}$ of proto-expressions can itself be viewed as a notation system $\langle \mathbf{T}_{\Sigma(\mathcal{L})}, \mathrm{id} \rangle$ for $\mathcal{L}$, where id is the identity function on $\mathrm{T}_{\Sigma(\mathcal{L})}$. Here, the proto-expressions themselves are the objects of our theorising, without any "detour" via notation. I think that this option has its merits. However, the goal of this paper is to show that Tarski's, Gödel's and Church's theorems hold for *all* reasonable conceptions of syntax. Hence, I adopt the more flexible and general approach to syntax in this paper, which accommodates the abstract conception captured by $\langle \mathbf{T}_{\Sigma(\mathcal{L})}, \mathrm{id} \rangle$ as a special case.

The next lemma shows that implementations map distinct p-variables and p-terms to distinct notations. In particular, each notation system contains infinitely many variables.

Lemma 4.13 *Let ι be an implementation of p-expressions of $\mathcal{L}$ into* **D**. *Let s, t be p-terms or p-variables. Then $s \approx_\iota t$ iff $s = t$.*

Proof Let s, t be p-terms with $s \approx_\iota t$. By clause (4) of Definition 4.11 we have

$$\mathtt{equ}(s, s) \approx_\iota \mathtt{equ}(s, t).$$

If s doesn't contain any p-variables, set $\varphi := \mathtt{equ}(s, s)$ and $\psi := \mathtt{equ}(s, t)$. If s contains the p-variables $x_1, \ldots, x_k$, let φ be the universal closure of $\mathtt{equ}(s, s)$, i.e.,

$$\varphi := \mathtt{univ}(x_1, \mathtt{univ}(x_2, \ldots \mathtt{univ}(x_k, \mathtt{equ}(s, s)))).$$

Since $\approx_\iota$ is a congruence relation, φ is ι-equivalent to the result of universally closing $\mathtt{equ}(s, t)$ under $x_1, \ldots, x_k$, i.e., to the p-formula ψ of the form

$$\psi := \mathtt{univ}(x_1, \mathtt{univ}(x_2, \ldots \mathtt{univ}(x_k, \mathtt{equ}(s, t)))).$$

Since $\#\mathrm{FV}(\varphi) = 0$, we have $\#\mathrm{FV}(\psi) = 0$ by Definition 4.11.2. Moreover, $\vdash \varphi$. Hence, by Definition 4.11.3, we have $\vdash \psi$. But then $s = t$, since pure predicate logic does not prove any p-sentence of the form ψ where $s \neq t$.

If s, t are p-variables, run the same argument for the p-terms $\mathtt{e}(s)$ and $\mathtt{e}(t)$. □

The following lemma shows that the derivability relation is well-defined for ι-formulas.

Lemma 4.14 *Let U and T be sets of p-sentences such that $\iota(T) = \iota(U)$. Then $T \vdash \chi$ iff $U \vdash \chi$ for every p-formula χ.*

Proof Let $\varphi \in T$. Since $\iota(T) = \iota(U)$, there is a $\psi \in U$ such that $\varphi_\iota = \psi_\iota$. But then $\vdash \mathtt{bicond}(\varphi, \psi)$ by clause (3) of Definition 4.11. Hence, $U \vdash \varphi$. This shows that U proves every member of T. Hence, U proves all theorems of T. The other direction follows similarly. □

4.3.3 Logical Notions and Notations for ι-Expressions

A large number of logical notions can be transferred to any notation system $\langle \mathbf{D}, \iota \rangle$ for $\mathcal{L}$. To begin with, we say that a ι-formula φ_ι is a ι-sentence, if φ is a p-sentence. This is well-defined by clause (2) of Definition 4.11. We write Var_ι, Term_ι, Fml_ι and Sent_ι for the sets of ι-variables, ι-terms, ι-formulas and ι-sentences respectively. For any ι-formula φ_ι with k free variables and ι-terms $s_{0\iota}, \ldots, s_{k-1\iota}$ we define $\varphi_\iota(s_{0\iota}, \ldots, s_{k-1\iota}) := \iota(\varphi(s_0, \ldots, s_{k-1}))$. In order to show that this substitution operation is well-defined, let ψ be a p-formula and let $t_0, \ldots, t_{k-1}$ be p-terms such that $\varphi \approx_\iota \psi$ and $s_j \approx_\iota t_j$ for each $j < k$. By clauses (2) and (4) of Definition 4.11, we then have

$$\varphi(s_0, \ldots, s_{k-1}) \approx_\iota \psi(t_0, \ldots, t_{k-1}),$$

and therefore $\iota(\varphi(s_0, \ldots, s_{k-1})) = \iota(\psi(t_0, \ldots, t_{k-1}))$.

Since $\approx_\iota$ is a congruence relation on $\mathbf{T}_{\Sigma(\mathcal{L})}$, each of the fundamental operations of $\mathbf{T}_{\Sigma(\mathcal{L})}$ correspond to well-defined constructor operations on ι-expressions. For example, there is a well-defined binary operation $\dot{\wedge}_\iota$ on Fml_ι given by

$$\dot{\wedge}_\iota(\iota(\varphi), \iota(\psi)) := \iota(\mathtt{conj}(\varphi, \psi)).$$

The operation $\dot{\wedge}_\iota$ thus maps any two ι-formulas to their ι-conjunction. Also the other fundamental operations of $\mathbf{T}_{\Sigma(\mathcal{L})}$ correspond to well-defined operations on ι-expressions. For the sake of better readability, I will take $(\varphi_\iota \dot{\wedge}_\iota \psi_\iota)$ to abbreviate $\iota(\mathtt{conj}(\varphi, \psi))$. That is, I use the familiar infix notation in order to abbreviate certain ι-expressions. If the implementation ι is specified by the context, I will also occasionally drop the index ι, i.e., simply write $(\varphi \dot{\wedge} \psi)$. The aim of these conventions is simply to make the subsequent material more readable. This convention extends to the other constructor operations for $\mathcal{L}$ along the following lines:

Definition 4.15 Let $\langle \mathbf{D}, \iota \rangle$ be a fixed notation system for $\mathcal{L}$. We now define operations $\mathtt{nv\dot{a}r}_\iota$, $\dot{\mathtt{e}}_\iota$, $\dot{f}_\iota$, $\dot{=}_\iota$, $\dot{R}_\iota$, $\dot{\neg}_\iota$, $\dot{\wedge}_\iota$, $\dot{\vee}_\iota$, $\dot{\rightarrow}_\iota$, $\dot{\leftrightarrow}_\iota$, $\dot{\forall}_\iota$ and $\dot{\exists}_\iota$ (for each $f \in \mathcal{F}$ and $R \in \mathcal{R}$) for p-variables x, p-terms $s, t, s_1, \ldots s_k$ and p-formulas $\varphi, \psi, \varphi_1, \ldots \varphi_k$ as follows:

$$\begin{aligned}
\dot{\mathtt{e}}_\iota(x_\iota) &:= \iota(\mathtt{e}(x)) \\
\mathtt{nv\dot{a}r}_\iota(x_\iota) &:= \iota(\mathtt{nvar}(x)) \\
\dot{f}_\iota s_{1\iota} \ldots s_{k\iota} &:= \iota(\mathtt{fct}_f(s_1, \ldots s_k)), \text{ for each k-ary } f \in \mathcal{F} \\
s_\iota \dot{=}_\iota t_\iota &:= \iota(\mathtt{equ}(s, t)); \\
\dot{R}_\iota s_{1\iota} \ldots s_{k\iota} &:= \iota(\mathtt{pred}_R(s_1, \ldots, s_k)), \text{ for each k-ary } R \in \mathcal{R} \\
\dot{\neg}_\iota \varphi_\iota &:= \iota(\mathtt{neg}(\varphi)) \\
(\varphi_\iota \dot{\wedge}_\iota \psi_\iota) &:= \iota(\mathtt{conj}(\varphi, \psi)) \\
(\varphi_\iota \dot{\vee}_\iota \psi_\iota) &:= \iota(\mathtt{disj}(\varphi, \psi)) \\
(\varphi_\iota \dot{\rightarrow}_\iota \psi_\iota) &:= \iota(\mathtt{cond}(\varphi, \psi)) \\
(\varphi_\iota \dot{\leftrightarrow}_\iota \psi_\iota) &:= \iota(\mathtt{bicond}(\varphi, \psi)) \\
\dot{\forall}_\iota x_\iota \varphi_\iota &:= \iota(\mathtt{univ}(x, \varphi)) \\
\dot{\exists}_\iota x_\iota \varphi_\iota &:= \iota(\mathtt{exists}(x, \varphi))
\end{aligned}$$

Each clause is well-defined, since $\approx_\iota$ is a congruence relation on $\mathbf{T}_{\Sigma(\mathcal{L})}$. For the sake of better readability, I will frequently identify $\dot{\mathtt{e}}_\iota(x_\iota)$ with x_ι. For example, I will write $x_\iota \dot{=} y_\iota$ instead of $\dot{\mathtt{e}}_\iota(x_\iota) \dot{=} \dot{\mathtt{e}}_\iota(y_\iota)$, where x, y are p-variables. I will also employ the usual bracketing conventions, such as omitting outer brackets, etc. For any $n \in \omega$, we write $\dot{\overline{n}}_\iota$ for the *standard ι-numeral* $\iota(\mathtt{num}_n)$ of n. Once again, this is well-defined, since by Lemma 4.13, $\iota(\mathtt{num}_n) = \iota(\mathtt{num}_m)$ implies $n = m$. If the context is clear, we sometimes omit the index ι.

Finally, we define the derivability and satisfiability relations for ι-notations in a similar "parasitic" fashion. For a set T of p-expressions we set $T_\iota := \{\iota(\varphi) \mid \varphi \in T\}$. We say that a ι-formula φ_ι can be derived from a set of ι-sentences T_ι, in symbols: $T_\iota \vdash \varphi_\iota$, if $T \vdash \varphi$. In order to see that this definition is well-defined, let U and ψ be

given such that $T_\iota = U_\iota$ and $\varphi_\iota = \psi_\iota$. What we need to show is that $T \vdash \varphi$ iff $U \vdash \psi$. Since $\vdash \mathtt{bicond}(\varphi, \psi)$ by clause (3) of Definition 4.11, it is sufficient to show that $T \vdash \varphi$ iff $U \vdash \varphi$, which follows from Lemma 4.14.

Similarly, we say that a ι-sentence φ_ι is true in $\mathcal{N}$, in symbols: $\mathcal{N} \models \varphi_\iota$, if $\mathcal{N} \models \varphi$. Once again, this is well-defined by clause (3) of Definition 4.11.

4.4 Gödel Numberings

For any notation system $\langle \mathbf{D}, \iota \rangle$, a Gödel numbering of $\mathbf{D}$ is an injective function from $\mathbf{D}$'s domain $\bigcup |\mathbf{D}|$ to ω. A numbering thus codes all the raw expressions of a data structure, and not only the well-formed ι-notations. This framework reflects the widespread custom in metamathematics that numberings are defined on the whole data structure by simulating its fundamental operations. For instance, in the case of finite strings, one typically first assigns numbers to the alphabetical symbols and then goes on to define the Gödel number of a string as a function of the Gödel numbers of its entries (e.g. based on prime factorisation in the tradition of Gödel [21] or on k-adic notation following Smullyan [57, 58]). Hence, these numberings are defined for the whole data structure, i.e., the set of finite strings over the given alphabet, and not only for the set of well-defined notations on strings. Moreover, these numberings essentially simulate the structure of strings together with the concatenation operation (as given by the semigroup $\mathbf{A}^*_{\mathcal{L}}$ with domain $A^*_{\mathcal{L}}$ and fundamental operation $*$), rather than simulating the structure of p-expressions directly (as given by the algebra $\mathbf{T}_{\Sigma(\mathcal{L})}$).[22]

In this section, I will define what it means for a numbering of a given data structure to be recursive or definable. These notions will play a central role in the subsequent discussion of admissible numberings and notation systems. The main idea is that a numbering is recursive, if it "simulates" all the fundamental operations of the data structure by recursive functions. Similarly, a numbering is definable, if it "simulates" the fundamental operations by definable functions. Here, I simply survey the material relevant to this paper. Further details and proofs can be found in [23].

Let $\mathbb{REC}$ denote the set of (total) recursive functions. Let $\mathbb{DEF}(\mathcal{N})$ denote the set of numerical functions which are definable in $\mathcal{N}$. In what follows, let $\mathcal{C} = \mathbb{REC}$ or $\mathcal{C} = \mathbb{DEF}(\mathcal{N})$, for some arithmetical interpretation $\mathcal{N}$ of some $\mathcal{L} \supseteq \mathcal{L}_0$.

It will be useful to allow also numerical functions which are only defined on a proper subset of ω to be contained in $\mathcal{C}$. Let $A \subseteq \omega$ and $f : A \to \omega$ be given. We then define that $f \in \mathcal{C}$ if the characteristic function of A is in $\mathcal{C}$ and the total extension f' of f is in $\mathcal{C}$, which is given by

$$f'(n) = \begin{cases} f(n) & \text{if } n \in A; \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, we say that a numerical relation R is in $\mathcal{C}$, if its characteristic function is in $\mathcal{C}$. In particular, a function $f : B \to \omega$ is called recursive, if $B \subseteq \omega$ is decidable and there exists a total recursive function $f' : \omega \to \omega$ with $\mathrm{dom}(f') = B$ and $f' \restriction B = f$.

[22] In the special case that the data structure of the notation system is taken to be $\langle \mathbf{T}_{\Sigma(\mathcal{L})}, \mathrm{id} \rangle$ (see Remark 4.12), the Gödel numbering only codes the well-formed expressions and no residual "junk material".

The following notion of a tracking function will be of central importance in the remainder of this paper:

Definition 4.16 Let $A_0, \dots, A_k$ and B be sets, for some $k > 0$, and let $\alpha_i : A_i \to B$ be an injective function, for each $i \leq k$. Let $f : A_0 \times \cdots \times A_{k-1} \to A_k$ be any function. We call the operation

$$f_{\alpha_0,\dots,\alpha_k} : \alpha_0(A_0) \times \cdots \times \alpha_{k-1}(A_{k-1}) \to \alpha_k(A_k)$$

given by

$$f_{\alpha_0,\dots,\alpha_k}(b_0, \dots, b_{k-1}) = \alpha_k(f(\alpha_0^{-1}(b_0), \dots, \alpha_{k-1}^{-1}(b_{k-1}))),$$

the $\langle \alpha_0, \dots, \alpha_k \rangle$-*tracking function of* f. If $\alpha = \alpha_i$ for all $i \leq k$, we also call $f_{\alpha_0,\dots,\alpha_k}$ the α-tracking function of f. In that case we also write f_α for $f_{\alpha_0,\dots,\alpha_k}$.

Intuitively, the tracking function $f_{\alpha_0,\dots,\alpha_k}$ simulates the function f on the set of codes. This can be illustrated by the observation that $f_{\alpha_0,\dots,\alpha_k}$ is the unique function that makes the following diagram commute

$$\begin{array}{ccc} A_0 \times \cdots \times A_{k-1} & \xrightarrow{\quad f \quad} & A_k \\ \downarrow{\scriptstyle \alpha_0 \times \cdots \times \alpha_{k-1}} & & \downarrow{\scriptstyle \alpha_k} \\ \alpha_0(A_0) \times \cdots \times \alpha_{k-1}(A_{k-1}) & \xrightarrow{f_{\alpha_0,\dots,\alpha_k}} & \alpha_k(A_k) \end{array} \tag{4}$$

where $\alpha_0 \times \cdots \times \alpha_{k-1}(a_0, \dots, a_{k-1}) := \langle \alpha_0(a_0), \dots, \alpha_{k-1}(a_{k-1}) \rangle$.

Tracking functions allow us to precisely capture the idea that a function on an arbitrary data structure is recursive or definable relative to a coding.

Definition 4.17 Let $k > 0$ and let A_i be a set and α_i be a numbering of A_i for each $i \leq k$. We say that a function $f : A_0 \times \cdots \times A_{k-1} \to A_k$ is in $\mathcal{C}$ *relative to* $\langle \alpha_0, \dots, \alpha_k \rangle$, if its $\langle \alpha_0, \dots, \alpha_k \rangle$-tracking function is in $\mathcal{C}$. We say that a set (or relation) $S \subseteq A_0 \times \cdots \times A_{k-1}$ is in $\mathcal{C}$ *relative to* $\langle \alpha_0, \dots, \alpha_{k-1} \rangle$, if

$$(\alpha_0 \times \cdots \times \alpha_{k-1})(S) := \{\langle \alpha_0(a_0), \dots, \alpha_{k-1}(a_{k-1}) \rangle \mid \langle a_0, \dots, a_{k-1} \rangle \in S\}$$

is in $\mathcal{C}$.

Despite this apparent relativity to coding, I will now introduce robust notions of computability and definability. I first make precise what it means that a numbering simulates all the fundamental operations of the data structure by recursive functions (and similarly for definability).

Definition 4.18 Let $\mathbf{D}$ be a many-sorted algebra and let α *be a numbering of* $\mathbf{D}$. We call α a $\mathcal{C}$-*numbering of* $\mathbf{D}$, if

(1) $\alpha(|\mathbf{D}|^s)$ is in $\mathcal{C}$ for each sort s of $\mathbf{D}$;
(2) the α-tracking function of each fundamental operation of $\mathbf{D}$ is in $\mathcal{C}$.

We also call $\mathbb{REC}$-numberings and $\mathbb{DEF}(\mathcal{N})$-numberings *recursive* and $\mathcal{N}$*-definable* respectively. If there is a recursive numbering of $\mathbf{D}$, then also call $\mathbf{D}$ *recursive*.

I will now show that the restriction to recursive numberings yields an absolute notion of recursiveness. We first observe that $\mathcal{C}$ induces an important pre-order of numberings:

Definition 4.19 Let α and β be numberings of a set A. We say that α is $\mathcal{C}$*-reducible to* β if there exists a function $f \in \mathcal{C}$ such that $\alpha^{-1}(n) = \beta^{-1} \circ f(n)$ for all $n \in \alpha(A)$. We say that α and β are $\mathcal{C}$*-equivalent*, in symbols $\alpha \sim_{\mathcal{C}} \beta$, if α is $\mathcal{C}$-reducible to β and β is $\mathcal{C}$-reducible to α.

The relation of $\mathcal{C}$-reducibility is a pre-order on the set of numberings of A. Hence, $\sim_{\mathcal{C}}$ is an equivalence relation. For example, $\alpha \sim_{\mathbb{REC}} \beta$ iff the "translation functions" between α and β are recursive.

Indeed, any two $\mathcal{C}$-numberings of a data structure are equivalent in this sense:[23]

Theorem 4.20 *We have $\alpha \sim_{\mathcal{C}} \beta$, for any two $\mathcal{C}$-numberings α and β of a data structure.*

Moreover, the recursiveness and definability of a function over an arbitrary data structure is invariant regarding equivalent numberings.

Lemma 4.21 *Let A be a set and let α and β be numberings of A such that $\alpha \sim_{\mathcal{C}} \beta$. We then have for every $k = 1, \ldots k$:*

(1) A function $f : A^k \to A$ is in $\mathcal{C}$ relative to α iff f is in $\mathcal{C}$ relative to β;
(2) A relation $R \subseteq A^k$ is in $\mathcal{C}$ relative to α iff R is in $\mathcal{C}$ relative to β.

We have thus shown that the restriction to recursive and definable numberings yield robust notions of recursiveness and definability of functions over arbitrary data structure.

4.5 Absolute Versions of Metamathematical Results

The introduced algebraic framework provides us with precise quantifiers over notation systems and numberings. That is, the informal claim that a given metamathematical theorem is invariant regarding the choice of the notation system and the Gödel numbering can be rephrased as a precise metamathematical statement. We now return to the problem to what extent we can abstract away from these choices in the formulation of Tarski's, Church's and Gödel's theorems.

Question 4.22 Let $\langle \mathcal{L}, \mathcal{N} \rangle$ be an arithmetically interpreted language, let $\langle \mathbf{D}, \iota \rangle$ be a notation system for $\mathcal{L}$ and let α be a numbering of $\mathbf{D}$. Under which assumptions on $\langle \mathbf{D}, \iota \rangle$ and α is $\{\varphi_\iota \in \mathrm{Sent}_\iota \mid \mathcal{N} \models \varphi_\iota\}$ not definable in $\mathcal{N}$ relative to α?

Question 4.23 Let $\mathcal{L} \supseteq \mathcal{L}_0$, let $\langle \mathbf{D}, \iota \rangle$ be a notation system for $\mathcal{L}$ and let α be a numbering of $\mathbf{D}$. Under which assumptions on $\langle \mathbf{D}, \iota \rangle$ and α is the function $f : \mathrm{Sent}_\iota \to \omega$, given by

$$f(\varphi_\iota) := \begin{cases} 0 & \text{if } \models \varphi_\iota; \\ 1 & \text{if } \not\models \varphi_\iota; \end{cases}$$

[23] This is a generalisation of a powerful theorem due to Mal'cev [42].

not recursive relative to $\langle \alpha, \mathsf{id}_\omega \rangle$?

We now turn to a popular formulation of Gödel's Second Theorem which is based on Löb's [41] derivability conditions. Note that Löb's conditions are usually formulated with respect to a specific numbering and notation system. We first make the used notation system and numbering in the formulation of Löb's condition explicit:

Definition 4.24 Let $\mathcal{L} \supseteq \mathcal{L}_0$ and let T be an $\mathcal{L}$-theory. Let $\langle \mathbf{D}, \iota \rangle$ be a notation system for $\mathcal{L}$ and let α be a numbering of $\mathbf{D}$. A ι-formula $\mathsf{Pr}^\alpha_\iota(x)$ is then said to satisfy Löb's conditions *relative to* α *and* ι *for* T, in short: Löb(T, ι, α), if for all ι-sentences φ_ι and ψ_ι:

Löb1$^{\alpha,\iota}$ $T_\iota \vdash \varphi_\iota$ implies $T_\iota \vdash \mathsf{Pr}^\alpha_\iota(\ulcorner \varphi_\iota \urcorner^\alpha)$;
Löb2$^{\alpha,\iota}$ $T_\iota \vdash (\mathsf{Pr}^\alpha_\iota(\ulcorner \varphi_\iota \urcorner^\alpha) \mathbin{\dot\wedge} \mathsf{Pr}^\alpha_\iota(\ulcorner \varphi_\iota \to \psi_\iota \urcorner^\alpha)) \mathbin{\dot\to} \mathsf{Pr}^\alpha_\iota(\ulcorner \psi_\iota \urcorner^\alpha)$;
Löb3$^{\alpha,\iota}$ $T_\iota \vdash \mathsf{Pr}^\alpha_\iota(\ulcorner \varphi_\iota \urcorner^\alpha) \mathbin{\dot\to} \mathsf{Pr}^\alpha_\iota(\ulcorner \mathsf{Pr}^\alpha_\iota(\ulcorner \varphi_\iota \urcorner^\alpha) \urcorner^\alpha)$.

We ask to what extent we can abstract away from the notation system and the numbering in the formulation of Gödel's Second Theorem.

Question 4.25 Let $\mathcal{L} \supseteq \mathcal{L}_0$ and let $T \supseteq \mathsf{R}$ be a consistent r.e. $\mathcal{L}$-theory.[24] Let $\langle \mathbf{D}, \iota \rangle$ be a notation system for $\mathcal{L}$ and let α be a numbering of $\mathbf{D}$. Under which assumptions on $\langle \mathbf{D}, \iota \rangle$ and α do we have that $T_\iota \nvdash \dot\neg \mathsf{Pr}^\alpha_\iota(\ulcorner \bot_\iota \urcorner^\alpha)$ for every $\mathcal{L}$-formula $\mathsf{Pr}^\alpha_\iota(x)$ satisfying Löb(T, ι, α)?

The following partial answers to these questions can be extracted from [22]. Let $\mathbf{A}^*_\mathcal{L}$ denote the semi-group with domain $A^*_\mathcal{L}$ and fundamental operation $*$. Let the notation system be fixed and taken to be the infix system $\langle \mathbf{A}^*_\mathcal{L}, \mathsf{inf} \rangle$ on strings. Then extra assumptions need to be imposed on the numbering, since there are deviant numberings of strings which violate Tarski's, Church's and Gödel's theorems. Moreover, the condition of being a $\mathcal{N}$-definable numbering of $\mathbf{A}^*_\mathcal{L}$ is sufficient for the invariance of Tarski's Theorem, while Church's and Gödel's theorems are invariant regarding recursive numberings of $\mathbf{A}^*_\mathcal{L}$.

In the remainder of this paper we will examine the conditions under which these theorems are invariant regarding numberings *and* notation systems. In doing so, we will obtain complete answers to our questions.

5 Deviancy

In addition to there being deviant numberings, I show in this section that deviancy can also occur on the level of the notation system. That is, there are contrived notation systems which give rise to definable truth predicates, provable consistency sentences and computable decision procedures, in violation to the fundamental theorems due to Tarski, Gödel and Church respectively. Hence, additional assumptions on the notation system need to be imposed in order to establish the invariance of Tarski's, Church's and Gödel's theorems.

[24] Strictly speaking, a numbering is needed to make the claim precise that an $\mathcal{L}$-theory is r.e. Lemma 4.21 shows that the choice of the numbering does not matter, as long as the numbering is recursive (see Definition 4.18).

5.1 Deviant Implementations

A simple way of constructing deviant notation systems proceeds as follows. Let A be some finite alphabet. Let $(e_i)_{i\in\omega}$ and $(o_i)_{i\in\omega}$ be enumerations (without repetitions) of the A-strings with even and odd length respectively. Moreover, let $(\varphi_i)_{i\in\omega}$ be an enumeration (without repetitions) of the p-sentences of $\mathcal{L}_0$ that are true in $\mathbb{N}$ and let $(\psi_i)_{i\in\omega}$ be an enumeration (without repetitions) of all the p-expressions of $\mathcal{L}_0$ that are not p-sentences true in $\mathbb{N}$. Let κ map φ_i to e_i and ψ_i to o_i, for each $i \in \omega$.[25] Hence, κ is an implementation of p-expressions of $\mathcal{L}_0$ into the semigroup $\mathbf{A}^*$ of strings. So $\langle \mathbf{A}^*, \kappa\rangle$ is a notation system for $\mathcal{L}_0$. Since the true κ-sentences are precisely A-strings of even length, Tarski's Theorem fails. More precisely, the set $\{\varphi \in \mathrm{Sent}_\kappa \mid \mathbb{N} \models \varphi\}$ of true κ-sentences is arithmetically definable (and even decidable), relative to some standard numbering γ of A^*.

We obtain further deviant metamathematical results in a similar way. For example, instead of enumerating all true sentences as above, let now $(\varphi_i)_{i\in\omega}$ enumerate all PA-theorems. Let $(\psi_i)_{i\in\omega}$ be an enumeration (without repetitions) of all the p-expressions of $\mathcal{L}_0$ that are not PA-theorems. Let μ be the implementation of p-expressions of $\mathcal{L}_0$ into $\langle A^*, *\rangle$ defined along the same lines as above, such that a μ-notation is a PA-theorem iff it has even length. Let now $\mathsf{Even}(x)$ be a p-formula which binumerates[26] the set of γ-codes of A-strings of even length. Then $\mathsf{Even}(x)_\mu$ satisfies $\mathrm{Löb}(\mathsf{PA}, \mu, \gamma)$ and we have $\mathsf{PA}_\mu \vdash \dot{\neg}\mathsf{Even}(\ulcorner \bot_\mu \urcorner^\gamma)_\mu$ by the Σ^0_1-completeness of PA, in violation of Gödel's Second Theorem. Using the same idea, we obtain an implementation ξ of p-expressions of $\mathcal{L}_0$ into $\mathbf{A}^*$ by letting $(\varphi_i)_{i\in\omega}$ enumerate all logically valid p-sentences. By definition, the characteristic function of the set $\mathrm{Val}(\mathcal{L}_0)_\xi$ of logically valid ξ-sentences is recursive (relative to $\langle\gamma, \mathsf{id}_\omega\rangle$), hence violating Church's Theorem.

More careful constructions yield deviant implementations which ensure the computability and decidability of several syntactic operations and relations respectively. For example, we can construct an implementation ν of p-expressions of $\mathcal{L}_0$ into $\mathbf{A}^*$ such that the constructor operations $\dot{\mathsf{S}}_\nu, \dot{+}_\nu, \dot{\times}_\nu, \dot{\mathtt{nvar}}_\nu, \dot{=}_\nu, \dot{\neg}_\nu, \dot{\wedge}_\nu, \dot{\vee}_\nu, \dot{\rightarrow}_\nu$ and $\dot{\leftrightarrow}_\nu$ are recursive (relative to the standard numbering γ), while the set $\{\varphi \in \mathrm{Sent}_\nu \mid \mathbb{N} \models \varphi\}$ of true ν-sentences remains decidable.

Similarly,[27] we can construct an implementation π of p-expressions of $\mathcal{L}_0$ into $\mathbf{A}^*$ such that the constructor operations $\dot{\mathsf{S}}_\pi, \dot{+}_\pi, \dot{\times}_\pi, \dot{\mathtt{nvar}}_\pi, \dot{=}_\pi, \dot{\wedge}_\pi$ and $\dot{\forall}_\pi$ are recursive relative to γ, while the set (of codes of) PA-theorems is binumerable by a formula $\mathsf{Pr}_\pi(x)$. In particular, $\mathsf{Pr}_\pi(x)$ satisfies $\mathrm{Löb}(T, \pi, \gamma)$ but

$$\mathsf{PA}_\pi \vdash \dot{\neg}\mathsf{Pr}_\pi(\ulcorner \bot_\pi \urcorner^\gamma).$$

[25] Strictly speaking, an implementation is a many-sorted function, see Remark 4.10. So officially, κ is the many-sorted function $\langle \kappa^{\mathrm{var}}, \kappa^{\mathrm{ter}}, \kappa^{\mathrm{fml}}\rangle$, such that κ^{var}, κ^{ter} and κ^{fml} map p-expressions of the relevant sort to the A-strings specified above.

[26] An arithmetical formula $\varphi(x)$ *binumerates* a set of numbers A, if (1) $n \in A$ iff $T \vdash \varphi(\overline{n})$ and (2) $n \notin A$ iff $T \vdash \neg\varphi(\overline{n})$. Similarly, for k-ary relations. If no theory T is specified, we mean that $\varphi(x)$ binumerates A in R. Some authors use "strongly represents" instead of "binumerates".

[27] The construction of ν and π proceeds along the same lines as the construction of the deviant numberings η and δ introduced in [22]. For example, let A contain a single element such that $\mathbf{A}^*$ is isomorphic to the semigroup $\langle \mathbb{N}, +\rangle$. Setting $\nu := \eta \circ \mathsf{inf}$ and $\pi := \delta \circ \mathsf{inf}$ then yields the desired results, where inf is the standard infix implementation of p-expressions into $\mathbf{A}^*$.

Both notation systems $\langle \mathbf{A}^*, \nu \rangle$ and $\langle \mathbf{A}^*, \pi \rangle$ thus yield deviant results, even though they ensure the effectiveness of a large portion of syntax (relative to γ). For instance, the num-function (mapping numbers to their standard numerals) and the substitution function for terms as well as for atomic formulas are recursive relative to γ. In particular, both notation systems permit the binumeration of these important syntactic functions. As we will see in Section 7.2, the constructions of ν and π are optimal in the following sense. There is no implementation ν' which in addition to the constructor operation above, also turns $\dot{\forall}_{\nu'}$ into a recursive operation, while still inducing a definable set of arithmetical truths. Similarly, there is no implementation π' which in addition to the constructor operation above, also turns $\dot{\neg}_{\pi'}$ into a recursive operation, while still violating Gödel's theorem. Hence, neither ν nor π permit the (effective) arithmetisation of the substitution function for complex formulas. Since the substitution function is needed in the standard proof of the Diagonal Lemma, this observation explains why the standard proofs of Tarski's and Gödel's results cannot be carried out using these deviant notation systems.

5.2 Deviant Data Structures

In [22] deviant numberings are employed while keeping the notation system standard and fixed. In the previous examples, deviancy results from a contrived implementation of p-expressions, while employing a standard data structure as well as a standard numbering. Deviant results can also be obtained from contrived choices of the data structure. To see this, let $(\psi_i)_{i\in\omega}$ be an enumeration of all $A_{\mathcal{L}_0}$-strings that are infix-sentences true in $\mathbb{N}$ and let $(\chi_i)_{i\in\omega}$ be an enumeration of all other $A_{\mathcal{L}_0}$-strings. We can then successively generate $A^*_{\mathcal{L}_0}$ in a "zigzag" fashion by the unary operation $\mathfrak{z}$ on $A_{\mathcal{L}_0}$-strings given as follows:

$$\mathfrak{z}(\varphi) = \begin{cases} \chi_i & \text{if } \varphi = \psi_i; \\ \psi_{i+1} & \text{if } \varphi = \chi_i. \end{cases}$$

Since each $A_{\mathcal{L}_0}$-string can be generated by $\mathfrak{z}$ from ψ_0, $\langle A^*_{\mathcal{L}_0}, \mathfrak{z} \rangle$ is a data structure. So $\langle \langle A^*_{\mathcal{L}_0}, \mathfrak{z} \rangle, \mathsf{inf} \rangle$ is a notation system for $\mathcal{L}_0$. Consider now the numbering α of $A^*_{\mathcal{L}_0}$ which is given by

$$\alpha(\varphi) = \begin{cases} 2i & \text{if } \varphi = \psi_i; \\ 2i+1 & \text{if } \varphi = \chi_i. \end{cases}$$

Once again, the numbering α permits the construction of an arithmetical truth predicate in violation to Tarski's Theorem. In [22], α is shown to be inadmissible, since α is not a recursive numbering of the "standard" data structure $\mathbf{A}^*_{\mathcal{L}_0}$, whose fundamental operation is $*$. However, here α cannot be ruled out as inadmissible by the same reasoning, since α is a recursive numbering of the given data structure $\langle A^*_{\mathcal{L}_0}, \mathfrak{z} \rangle$. This is because the α-tracking function $\mathfrak{z}_\alpha$ of $\mathfrak{z}$ is the successor operation $\lambda n.n+1$ on ω. So the culprit in this example is the contrived choice of the notation system's data structure.

These considerations show that there are three independent sources from which deviant results can emerge: the data structure, the implementation of p-expressions and the numbering.

5.3 The Failure of Diagonalisation

What all our deviant constructions have in common is that they violate some version of the Diagonal Lemma. This matches our previous observation that no deviant notation system permits the arithmetisation of the substitution function for complex formulas, which is a crucial ingredient of the standard proof of the Diagonal Lemma.

More specifically, there is no μ-sentence which is a PA-provable (γ-)fixed point of the μ-formula $\dot{\neg}\mathsf{Even}(x)_\mu$, where $\mathsf{Even}(x)_\mu$ is the deviant provability predicate introduced in Section 5.1. To show this, assume that there is a μ-sentence λ_μ such that

$$\mathsf{PA}_\mu \vdash \dot{\neg}\mathsf{Even}_\mu(\ulcorner \lambda_\mu \urcorner^\gamma) \dot{\leftrightarrow} \lambda_\mu.$$

If $\mathsf{PA}_\mu \vdash \lambda_\mu$, then $\mathsf{PA}_\mu \vdash \mathsf{Even}_\mu(\ulcorner \lambda_\mu \urcorner^\gamma)$, since $\mathsf{Even}(x)_\mu$ numerates the set of PA-provable μ-sentences. But then $\mathsf{PA}_\mu \vdash \dot{\neg}\lambda_\mu$, in contradiction to the consistency of PA_μ. If $\mathsf{PA}_\mu \nvdash \lambda_\mu$, then $\mathsf{PA}_\mu \vdash \dot{\neg}\mathsf{Even}_\mu(\ulcorner \lambda_\mu \urcorner^\gamma)$, since $\mathsf{Even}(x)_\mu$ also *bi*numerates the set of PA-provable μ-sentences. But then we get $\mathsf{PA}_\mu \vdash \lambda_\mu$, another contradiction. Hence, no μ-sentence is a PA-provable fixed point of $\dot{\neg}\mathsf{Even}(x)_\mu$.

A similar argument shows that there is no fixed point λ_κ such that

$$\mathbb{N} \models \dot{\neg}\mathsf{Even}_\kappa(\ulcorner \lambda_\kappa \urcorner^\gamma) \dot{\leftrightarrow} \lambda_\kappa.$$

6 Admissible Notation Systems

The previous section contains deviant notation systems which give rise to definable truth predicates, provable consistency sentences and computable decision procedures. In this section, I argue that these constructions are not genuine counterexamples to Tarski's, Gödel's and Church's theorems. This is because the deviant notation systems employ resources exceeding the considered formalism and therefore are inadmissible choices in the formalisation process.

We start with the admissibility of numberings. In the context of Gödel's Second Theorem for a consistent and r.e. $\mathcal{L}$-theory $T \supseteq \mathsf{R}$, I take every admissible numbering to be a recursive numbering of the underlying data structure. In the context of Tarski's Theorem and a given interpretation $\mathcal{N}$, every admissible numbering is an $\mathcal{N}$-definable numbering. For a discussion and defence of these criteria see [22].[28] Roughly speaking, the requirement of recursiveness ensures that the data structure's fundamental operations can be simulated by recursive operations on the set of codes. This requirement, in turn, captures the idea that the arithmetisation of the given linguistic expressions does not exceed the resources of the formal theory T [22]. From

[28] The discussion in [22] is confined to the data structure $\langle A^*_{\mathcal{L}}, *\rangle$ of strings together with concatenation. Yet, my analysis and defence of these admissibility criteria given in [22] apply *mutatis mutandis* to arbitrary data structures.

similar considerations we extract the requirement that each admissible numbering in the context of Church's Theorem is a recursive numbering.

I now turn to the admissibility of notation systems. More specifically, I extract minimal conditions for admissible notation systems.[29] Once again, admissibility is analysed relative to the metamathematical context, i.e., the given theory and the given interpretation. If this analysis is correct, admissibility is a highly context- and theory-sensitive notion.

6.1 Provability

I start by analysing the admissibility of the first component of notation systems. In the context of Gödel's Second Theorem for a consistent and r.e. $\mathcal{L}$-theory $T \supseteq \mathsf{R}$, we are essentially concerned with the question whether a sentence expressing T's consistency is derivable by means of T's resources. It is therefore particularly important in this context that admissible notation systems ensure that reasoning about the well-formed expressions of $\mathcal{L}$ via their notations requires no resources which lie outside the formal system T. In order to make this precise, let $\langle \mathbf{D}, \iota \rangle$ be a notation system for $\mathcal{L}$. The first requirement of the notation system's admissibility is that the data structure $\mathbf{D}$ can be encoded "effectively". That is, admissible data structure are required to be recursive (see Definition 4.18).

We now turn to the second component of a notation system. Recall that each implementation ι induces constructor operations on ι-expressions, such as the ι-conjunction operation $\dot{\wedge}_\iota$ (see Definition 4.15). The claim that the implementation ι only employs resources of the formal theory T can be made precise by requiring that T "recognises" or "knows", i.e., proves certain facts about ι. Specifically, we require (i) that T "knows" whether or not two ι-expressions denote the same element, (ii) that T "knows" whether or not an element of the data structure is a ι-variable, ι-term or a ι-formula and (iii) that T "knows" for any given ι-formulas φ_ι, ψ_ι and χ_ι whether or not $\dot{\neg}_\iota \varphi_\iota = \psi_\iota$, whether or not $\varphi_\iota \mathbin{\dot{\wedge}_\iota} \psi_\iota = \chi_\iota$, and similarly for the other constructor operations introduced in Definition 4.15.

We now consider each of these three clauses in detail.

(i) The first clause can be made precise by requiring that T binumerates the ι-equivalence relation $\approx_\iota$ relative to some recursive numbering α of $\mathbf{T}_{\Sigma(\mathcal{L})}$. Since a r.e. theory $T \supseteq \mathsf{R}$ binumerates exactly the recursive sets (see [61, corollary II.7]), this is tantamount to requiring that the relation $\approx_\iota$ is recursive.

(ii) The second clause can be made precise by requiring that T binumerates the sets Var_ι, Ter_ι and Fml_ι relative to some recursive numbering α of $\mathbf{D}$. This is tantamount to requiring that the sets Var_ι, Ter_ι and Fml_ι are recursive relative to α.

(iii) The third clause can be made precise by requiring that T binumerates the graph of each of the constructor operations $\dot{\mathtt{nvar}}_\iota$, $\dot{\mathtt{e}}_\iota$, $\dot{f}_\iota$, $\dot{=}_\iota$, $\dot{R}_\iota$, $\dot{\neg}_\iota$, $\dot{\wedge}_\iota$, $\dot{\vee}_\iota$, $\dot{\rightarrow}_\iota$, $\dot{\leftrightarrow}_\iota$, $\dot{\forall}_\iota$ and $\dot{\exists}_\iota$ (for each $f \in \mathcal{F}$ and $R \in \mathcal{R}$). By [61, corollary II.7] this is equivalent to requiring that these operations are recursive (relative to α).

[29] For the purposes of this paper it is sufficient to isolate necessary conditions of admissibility. I do not claim that these conditions are sufficient for a notation system to be admissible.

These extracted necessary requirements for admissible notation systems are summarised in the next definition (taking $\mathcal{C} := \mathbb{REC}$):

Definition 6.1 Let ι be an implementation of p-expressions of $\mathcal{L}$ into a data structure $\mathbf{D}$. We say that $\langle \mathbf{D}, \iota \rangle$ is a *$\mathcal{C}$-notation system for $\mathcal{L}$*, if the following holds:

(1) There is a $\mathcal{C}$-numbering α of $\mathbf{D}$;
(2) The relation $\approx_\iota$ on $\mathrm{T}_{\Sigma(\mathcal{L})}$ is in $\mathcal{C}$;
(3) Var_ι, Ter_ι and Fml_ι are in $\mathcal{C}$ relative to α;
(4) The constructor operations $\dot{\mathtt{nvar}}_\iota$, $\dot{\mathtt{e}}_\iota$, $\dot{f}_\iota$, $\dot{=}_\iota$, $\dot{R}_\iota$, $\dot{\neg}_\iota$, $\dot{\wedge}_\iota$, $\dot{\vee}_\iota$, $\dot{\rightarrow}_\iota$, $\dot{\leftrightarrow}_\iota$, $\dot{\forall}_\iota$ and $\dot{\exists}_\iota$ (for each $f \in \mathcal{F}$ and $R \in \mathcal{R}$) are in $\mathcal{C}$ relative to α.

We also call $\mathbb{REC}$-notation systems and $\mathbb{DEF}(\mathcal{N})$-notation systems *recursive* and *$\mathcal{N}$-definable* respectively.

Remark 6.2 The clause (2) of Definition 6.1 should be understood relative to some $\mathcal{C}$-numbering of $\mathbf{T}_{\Sigma(\mathcal{L})}$. That is, the relation $\approx_\iota$ on $\mathrm{T}_{\Sigma(\mathcal{L})}$ is in $\mathcal{C}$, relative to some $\mathcal{C}$-numbering of $\mathbf{T}_{\Sigma(\mathcal{L})}$ and Var_ι, Ter_ι and Fml_ι are in $\mathcal{C}$ relative to some $\mathcal{C}$-numbering of $\mathbf{D}$. Note that for every $\mathcal{L}$ there is some standard $\mathcal{C}$-numbering of $\mathbf{T}_{\Sigma(\mathcal{L})}$ (see, e.g., the proof of Lemma 7.3). Clauses (3) and (4) are relative to the numbering α. By Lemma 4.21 and Theorem 4.20, membership-in-$\mathcal{C}$ is independent of the specific choices of these numberings. Hence, Definition 6.1 does not rely on any particularities of the underlying numberings.

Clauses (3) and (4) in the above definition entail that α induces a $\mathcal{C}$-numbering of the "$\Sigma(\mathcal{L})$-algebra of ι-expressions". This can be made precise as follows.

Definition 6.3 Let $\langle \mathbf{D}, \iota \rangle$ be a notation system for $\mathcal{L}$. Let $\mathbf{D}_\iota$ denote the $\Sigma(\mathcal{L})$-algebra with domain $\langle \mathrm{Var}_\iota, \mathrm{Ter}_\iota, \mathrm{Fml}_\iota \rangle$ and with the fundamental operations $\dot{\mathtt{nvar}}_\iota$, $\dot{\mathtt{e}}_\iota$, $\dot{f}_\iota$, $\dot{=}_\iota$, $\dot{R}_\iota$, $\dot{\neg}_\iota$, $\dot{\wedge}_\iota$, $\dot{\vee}_\iota$, $\dot{\rightarrow}_\iota$, $\dot{\leftrightarrow}_\iota$, $\dot{\forall}_\iota$ and $\dot{\exists}_\iota$ (for each $f \in \mathcal{F}$ and $R \in \mathcal{R}$).

If we ignore the codes of "junk expressions" of $\mathbf{D}$, then we obtain a $\mathcal{C}$-numbering of the algebra $\mathbf{D}_\iota$ from any given $\mathcal{C}$-numbering of $\mathbf{D}$.

Lemma 6.4 *Let $\langle \mathbf{D}, \iota \rangle$ be a $\mathcal{C}$-notation system for $\mathcal{L}$. For every $\mathcal{C}$-numbering α of $\mathbf{D}$, the restriction $\alpha \restriction$ of α to $\mathrm{Var}_\iota \cup \mathrm{Ter}_\iota \cup \mathrm{Fml}_\iota$ is a $\mathcal{C}$-numbering of $\mathbf{D}_\iota$.*

Proof This follows immediately from Remark 6.2 and clauses (3) and (4) of Definition 6.1. □

We have shown above that in the context of Gödel's Second Theorem which is concerned with provability in a theory T, every admissible notation system $\langle \mathbf{D}, \iota \rangle$ is recursive.[30] We now turn to admissibility in the context of definability.

[30] The admissibility of a notation system has been analysed relative to the theory T. Yet, the extracted criterion is theory-independent. This is because r.e. and consistent theories extending R binumerate precisely the same sets and functions, namely, the decidable and recursive ones respectively. As soon as we consider Gödel's Theorem in the context of non-r.e. theories, we obtain theory-dependent criteria of admissibility.

6.2 Definability

According to its prevalent interpretation, Tarski's Theorem imposes limitations on the expressive resources of the given arithmetically interpreted language $\langle \mathcal{L}, \mathcal{N} \rangle$. In this context it is therefore crucial that the used representation devices, such as the notation system, do not employ resources beyond $\langle \mathcal{L}, \mathcal{N} \rangle$'s expressive power. So while in the case of Gödel's Theorem, admissibility was analysed in terms of the deductive resources of the theory T, here admissibility must ensure that reasoning about expressions does not resort to expressive resources exceeding the language $\langle \mathcal{L}, \mathcal{N} \rangle$. This can be made precise by requiring that for any admissible notation system $\langle \mathbf{D}, \iota \rangle$, there is an admissible numbering of the notation system's data structure and that $\mathcal{L}$ "expresses" certain facts about ι. Recall that every admissible numbering is an $\mathcal{N}$-definable numbering of $\mathbf{D}$. That $\mathcal{L}$ "expresses" certain facts about ι can now be made precise by requiring (i) that ι-equivalence $\approx_\iota$ is definable in $\mathcal{N}$, (ii) that the sets Var_ι, Ter_ι and Fml_ι are definable in $\mathcal{N}$ and (iii) that the constructor operations $\dot{\mathtt{nvar}}_\iota$, $\dot{\mathtt{e}}_\iota$, $\dot{f}_\iota$, $\doteq_\iota$, $\dot{R}_\iota$, $\dot{\neg}_\iota$, $\dot{\wedge}_\iota$, $\dot{\vee}_\iota$, $\dot{\rightarrow}_\iota$, $\dot{\leftrightarrow}_\iota$, $\dot{\forall}_\iota$ and $\dot{\exists}_\iota$ (for each $f \in \mathcal{F}$ and $R \in \mathcal{R}$) are $\mathcal{N}$-definable. Thus, when we scrutinise the expressive limits of an arithmetical interpretation of $\mathcal{N}$ (as in Tarski's Theorem), every admissible notation system is $\mathcal{N}$-definable.

6.3 Computability

We now turn to Church's Theorem, according to which the characteristic function of the set of logically valid sentences is not computable. Hence, as opposed to provability and definability above, here we are concerned with computational properties. Admissible choices of notation systems in this context ensure that computations on expressions do not resort to resources exceeding the used model of computation, i.e., the set $\mathbb{REC}$ of recursive functions. In order to make this precise, we first require that every admissible notation system $\langle \mathbf{D}, \iota \rangle$ contains a recursive data structure $\mathbf{D}$. This ensures that the objects contained in the data structure can be encoded "effectively". The claim that computations on expressions do not resort to resources exceeding $\mathbb{REC}$ can moreover be made precise by requiring that (functions of) $\mathbb{REC}$ decide and compute relevant properties about ι. By employing a minimal approach, we specifically require (i) that ι-equivalence $\approx_\iota$ is a decidable relation, (ii) that it is decidable whether or not an element of the data structure is a ι-variable, ι-term or a ι-formula and (iii) that the constructor operations $\dot{\mathtt{nvar}}_\iota$, $\dot{\mathtt{e}}_\iota$, $\dot{f}_\iota$, $\doteq_\iota$, $\dot{R}_\iota$, $\dot{\neg}_\iota$, $\dot{\wedge}_\iota$, $\dot{\vee}_\iota$, $\dot{\rightarrow}_\iota$, $\dot{\leftrightarrow}_\iota$, $\dot{\forall}_\iota$ and $\dot{\exists}_\iota$ (for each $f \in \mathcal{F}$ and $R \in \mathcal{R}$) are recursive. We therefore conclude that every admissible notation system in this context is recursive.

7 Invariance Results

7.1 Invariant Diagonalisation

Recall that the derivability relation on ι-notations introduced in Section 4.3.3 is "parasitic" on p-expressions. As a result, the choice of notation does not affect which

theorems are derivable from given axioms. That is, for any two notation systems $\langle \mathbf{D}, \iota \rangle$ and $\langle \mathbf{E}, \kappa \rangle$ for $\mathcal{L}$, $T_\iota \vdash \varphi_\iota$ iff $T_\kappa \vdash \varphi_\kappa$, for every $\mathcal{L}$-theory T and p-formula φ. Yet, the invariance of diagonalisation does not come for free. For as we have seen in Section 5.3, some notation systems invalidate the Diagonal Lemma.

To further illustrate this point, let $T \supseteq \mathsf{R}$ and let $\langle \mathbf{A}^*_{\mathcal{L}}, \mathsf{inf} \rangle$ denote the infix notation system for $\mathcal{L}$. By the standard version of the Diagonal Lemma we can find for every inf-formula φ_{inf} with one free variable an inf-sentence λ_{inf} such that

$$T_{\mathsf{inf}} \vdash \varphi_{\mathsf{inf}}(\ulcorner \lambda_{\mathsf{inf}} \urcorner^{\gamma}) \leftrightarrow \lambda_{\mathsf{inf}},$$

for some standard numbering γ of $A^*_{\mathcal{L}}$. Let now $\langle \mathbf{D}, \iota \rangle$ be any given notation system for $\mathcal{L}$. We can then conclude that

$$T_\iota \vdash \varphi_\iota(\ulcorner \lambda_{\mathsf{inf}} \urcorner^{\gamma}) \dot{\leftrightarrow} \lambda_\iota.$$

However, this is not the desired Diagonal Lemma for $\langle \mathbf{D}, \iota \rangle$, since the fixed point λ is "used" in ι-representation, while it is "mentioned" in inf-representation. Hence, more work is needed to obtain a version of the Diagonal lemma which is independent of the underlying (admissible) notation system.

In this section, we establish the following syntactic and the semantic versions of the Diagonal Lemma which are invariant regarding admissible notation systems and numberings.

Lemma 7.1 *Let $\mathcal{L} \supseteq \mathcal{L}_0$ be a language and let $T \supseteq \mathsf{R}$ be a $\mathcal{L}$-theory. Let $\langle \mathbf{D}, \iota \rangle$ be a recursive notation system for $\mathcal{L}$ and let α be a recursive numbering of $\mathbf{D}$. For every ι-formula φ_ι with one free variable there is a ι-sentence λ_ι such that*

$$T_\iota \vdash \varphi_\iota(\ulcorner \lambda_\iota \urcorner^{\alpha}) \dot{\leftrightarrow} \lambda_\iota.$$

Lemma 7.2 *Let $\langle \mathcal{L}, \mathcal{N} \rangle$ be an arithmetically interpreted language. Let $\langle \mathbf{D}, \iota \rangle$ be an $\mathcal{N}$-definable notation system for $\mathcal{L}$ and let α be an $\mathcal{N}$-definable numbering of $\mathbf{D}$. For every ι-formula φ_ι with one free variable there is a ι-sentence λ_ι such that*

$$\mathcal{N} \models \varphi_\iota(\ulcorner \lambda_\iota \urcorner^{\alpha}) \dot{\leftrightarrow} \lambda_\iota.$$

In order to prove these lemmas, we first define a *self-referential* numbering for the algebra $\mathbf{D}_\iota$ of ι-expressions.[31] Recall that $\mathcal{C}$ is $\mathbb{REC}$ or $\mathbb{DEF}(\mathcal{N})$, for some arithmetical interpretation $\mathcal{N}$ of some language $\mathcal{L} \supseteq \mathcal{L}_0$.

Lemma 7.3 *Let $\langle \mathbf{D}, \iota \rangle$ be a $\mathcal{C}$-notation system for $\mathcal{L}$. Then there is a self-referential $\mathcal{C}$-numbering δ of $\mathbf{D}_\iota$. That is, for each ι-formula φ_ι with exactly one free variable there is a number $m \in \omega$ such that $\delta(\varphi_\iota(\dot{\overline{m}})) = m$.*

[31] Self-referential numberings were introduced by Kripke [37, 38], Feferman [14] and Visser [64]. The construction given in this paper closely mimics Visser's [64] numbering ※. See [25] for a systematic study of self-referential numberings.

Proof We start by defining a "standard" numbering γ of $\mathrm{T}_{\Sigma(\mathcal{L})}$. For definiteness, let $\sigma^1, \ldots, \sigma^q$ be an enumeration of the symbols of $\Sigma(\mathcal{L})$. We now recursively define γ by setting

$$\gamma(t) := \begin{cases} 2i+1 & \text{if } t = \sigma^i \text{ is a constant symbol;} \\ 2^i p_1^{\gamma(t_1)} \cdots p_k^{\gamma(t_k)} & \text{if } t = \sigma^i(t_1, \ldots, t_k); \end{cases}$$

where p_i is the $(i+1)$-th prime number and $t, t_1, \ldots, t_k$ are p-expressions. Clearly, γ is a recursive numbering of $\mathbf{T}_{\Sigma(\mathcal{L})}$. Moreover, γ induces a numbering $\widetilde{\gamma}$ of $\mathbf{D}_\iota$ by setting

$$\widetilde{\gamma}(\psi_\iota) := \min\{\gamma(\varphi) \mid \varphi \approx_\iota \psi\}.$$

Since $\approx_\iota$ is in $\mathcal{C}$, $\widetilde{\gamma}$ is a $\mathcal{C}$-numbering of $\mathbf{D}_\iota$.

Let $(\psi_{\iota,j})_{j\in\omega}$ be an enumeration without repetitions of the ι-formulas with exactly one free variable, which is induced by $\widetilde{\gamma}$ in the following sense: for all $i, j \in \omega$ we have $i < j$ iff $\widetilde{\gamma}(\psi_{\iota,i}) < \widetilde{\gamma}(\psi_{\iota,j})$. We observe that

(1) The function $n \mapsto \psi_{\iota,n}$ is in $\mathcal{C}$ relative to $\langle \mathrm{id}, \widetilde{\gamma}\rangle$;
(2) The numeral function $n \mapsto \dot{\overline{n}}$ is in $\mathcal{C}$ relative to $\langle \mathrm{id}, \widetilde{\gamma}\rangle$;
(3) The substitution function $\psi_{\iota,j}, t_\iota \mapsto \psi_{\iota,j}(t_\iota)$ is in $\mathcal{C}$ relative to $\widetilde{\gamma}$.

We now define δ by setting

$$\delta(x_\iota) = 2 \cdot \widetilde{\gamma}(x_\iota), \text{ for every } \iota\text{-variable } x_\iota;$$
$$\delta(t_\iota) = 2 \cdot \widetilde{\gamma}(t_\iota), \text{ for every } \iota\text{-term } t_\iota.$$

For every ι-formula φ_ι we set

$$\delta(\varphi_\iota) = \begin{cases} 2n+1 & \text{if } n = \min\{k \mid \varphi_\iota = \psi_{\iota,k}(\dot{\overline{2k+1}})\}, \\ 2 \cdot \widetilde{\gamma}(\varphi_\iota) & \text{if } \varphi_\iota \neq \psi_{\iota,k}(\dot{\overline{2k+1}}), \text{ for all } k. \end{cases}$$

Clearly, δ is a numbering of $\mathbf{D}_\iota$. In order to show that δ is self-referential, let φ_ι be a ι-formula with exactly one free variable. Let $l \in \omega$ such that $\varphi_\iota = \psi_{\iota,l}$. Let

$$n := \min\left\{k \mid \psi_{\iota,l}(\dot{\overline{2l+1}}) = \psi_{\iota,k}(\dot{\overline{2k+1}})\right\}.$$

By definition of δ, we then get $\delta(\varphi_\iota(\dot{\overline{m}})) = m$ for $m := 2n+1$. Hence, δ is self-referential.

We observe that the sets $\delta(\mathrm{Var}_\iota)$, $\delta(\mathrm{Ter}_\iota)$ and $\delta(\mathrm{Fml}_\iota)$ are in $\mathcal{C}$. To see this in the case of $\delta(\mathrm{Fml}_\iota)$, we distinguish two cases.

Case 1: $m = 2n+1$, for some $n \in \omega$. Now check whether for all $k < n$

$$\widetilde{\gamma}(\psi_{\iota,n}(\dot{\overline{2n+1}})) \neq \widetilde{\gamma}(\psi_{\iota,k}(\dot{\overline{2k+1}})).$$

If yes, then $m \in \delta(\mathrm{Fml}_\iota)$. If no, then $m \notin \delta(\mathrm{Fml}_\iota)$. Note that by (1)–(3), this "decision procedure" is in $\mathcal{C}$.

Case 2: $m = 2n$, for some $n \in \omega$. Now check whether $n \in \widetilde{\gamma}(\mathrm{Fml}_\iota)$. If yes, then $m \in \delta(\mathrm{Fml}_\iota)$. If no, then $m \notin \delta(\mathrm{Fml}_\iota)$. Since $\widetilde{\gamma}(\mathrm{Fml}_\iota)$ is in $\mathcal{C}$, we have thus shown that $\delta(\mathrm{Fml}_\iota)$ is in $\mathcal{C}$.

Next, we show that (i) $\widetilde{\gamma} \circ \delta^{-1} \in \mathcal{C}$ and (ii) $\delta \circ \widetilde{\gamma}^{-1} \in \mathcal{C}$. In order to show (i), let $m \in \delta(\mathbf{D}_\iota)$. If $m = 2n + 1$ for some $n \in \omega$, then output $\widetilde{\gamma}(\psi_{\iota,n}(\dot{\overline{2n+1}}))$. By (1)–(3) this procedure is in $\mathcal{C}$.

If $m = 2n$ for some $n \in \omega$, then output n. Once again, this procedure is in $\mathcal{C}$. Hence, $\widetilde{\gamma} \circ \delta^{-1} \in \mathcal{C}$, which concludes the proof of (i).

In order to show (ii), let $m \in \widetilde{\gamma}(\mathbf{D}_\iota)$. Decide whether m is (the $\widetilde{\gamma}$-code) of a ι-formula of the form $\psi_{\iota,l}(\dot{\overline{2l+1}})$ for some $l \in \omega$. If not, then output $2m$. If yes, then output

$$2 \cdot \min\left\{k \mid \psi_{\iota,l}(\dot{\overline{2l+1}}) = \psi_{\iota,k}(\dot{\overline{2k+1}})\right\} + 1.$$

By (1)–(3) this procedure is in $\mathcal{C}$. Hence, $\delta \circ \widetilde{\gamma}^{-1} \in \mathcal{C}$, which concludes the proof of (ii).

By (i) and (ii) have $\delta \sim_{\mathcal{C}} \widetilde{\gamma}$. Using Lemma 4.21 and the fact that $\widetilde{\gamma}$ is a $\mathcal{C}$-numbering of $\mathbf{D}_\iota$, we conclude that also δ is a $\mathcal{C}$-numbering of $\mathbf{D}_\iota$. □

We now prove Lemma 7.2 by using a self-referential numbering.

Proof of Lemma *7.2* Let $\langle \mathbf{D}, \iota \rangle$ be an $\mathcal{N}$-definable notation system and let α be an $\mathcal{N}$-definable numbering of $\mathbf{D}$. By Lemma 6.4, the restriction $\alpha\restriction$ of α to $\mathrm{Var}_\iota \cup \mathrm{Ter}_\iota \cup \mathrm{Fml}_\iota$ is an $\mathcal{N}$-definable numbering of $\mathbf{D}_\iota$. By Lemma 7.3 there is a self-referential $\mathcal{N}$-definable numbering δ of $\mathbf{D}_\iota$. Hence, $\delta \sim_{\mathbb{DEF}(\mathcal{N})} \alpha\restriction$ by Theorem 4.20. Therefore, the function $\delta \circ \alpha\restriction^{-1}$ is $\mathcal{N}$-definable. So there is a ι-formula $f_\iota(x_\iota, y_\iota)$ that defines (the graph of) the function $\delta \circ \alpha\restriction^{-1}$.

Let now $\varphi_\iota(x_\iota)$ be a given ι-formula with free variable x_ι. Define

$$\psi_\iota(y_\iota) := \dot{\exists} z_\iota(f_\iota(z_\iota, y_\iota) \dot{\wedge} \varphi_\iota(z_\iota)).$$

Since δ is self-referential, there is a number m such that $\delta(\psi_\iota(\dot{\overline{m}})) = m$.

Hence, $\ulcorner \psi_\iota(\dot{\overline{m}}) \urcorner^\delta = \dot{\overline{m}}$, and therefore

$$\mathcal{N} \vDash \psi_\iota\left(\ulcorner \psi_\iota(\dot{\overline{m}} w) \urcorner^\delta\right) \dot{\leftrightarrow} \psi_\iota(\dot{\overline{m}}).$$

Since

$$\delta \circ \alpha\restriction^{-1}(\alpha(\psi_\iota(\dot{\overline{m}}))) = \delta(\psi_\iota(\dot{\overline{m}})),$$

we have

$$\mathcal{N} \vDash f_\iota\left(\ulcorner \psi_\iota(\dot{\overline{m}}) \urcorner^\alpha, \ulcorner \psi_\iota(\dot{\overline{m}}) \urcorner^\delta\right).$$

Hence,

$$\mathcal{N} \vDash \varphi_\iota\left(\ulcorner \psi_\iota(\dot{\overline{m}}) \urcorner^\alpha\right) \dot{\leftrightarrow} \psi_\iota\left(\ulcorner \psi_\iota(\dot{\overline{m}}) \urcorner^\delta\right).$$

Setting $\lambda_\iota = \psi_\iota(\dot{\overline{m}})$, then yields

$$\mathcal{N} \vDash \varphi_\iota(\ulcorner \lambda_\iota \urcorner^\alpha) \dot{\leftrightarrow} \lambda_\iota.$$

□

The proof of Lemma 7.1 proceeds similarly:

Proof of Lemma 7.1 Let $\langle \mathbf{D}, \iota \rangle$ be a recursive notation system and let α be a recursive numbering of $\mathbf{D}$. By Lemma 6.4, the restriction $\alpha\restriction$ of α to $\mathrm{Var}_\iota \cup \mathrm{Ter}_\iota \cup \mathrm{Fml}_\iota$ is a recursive numbering of $\mathbf{D}_\iota$. By Lemma 7.3 there is a self-referential recursive numbering δ of $\mathbf{D}_\iota$. Hence, $\delta \sim_{\mathbb{REC}} \alpha\restriction$ by Theorem 4.20. Therefore, the function $\delta \circ \alpha\restriction^{-1}$ is recursive. Let now φ_ι be a ι-formula with one free variable. Since $\delta \circ \alpha\restriction^{-1}$ is injective, by Lemma 4.10 & 4.11 of [22] there exists a ι-formula ψ_ι with one free variable and a ι-formula f_ι with two free variables which binumerates (the graph of) $\delta \circ \alpha\restriction^{-1}$ such that

$$\mathsf{R} \vdash f_\iota(\dot{n}, \dot{m}) \dot{\to} (\varphi_\iota(\dot{n}) \dot{\leftrightarrow} \psi_\iota(\dot{m})); \tag{5}$$

for all $n \in \alpha(\mathrm{Var}_\iota \cup \mathrm{Ter}_\iota \cup \mathrm{Fml}_\iota)$ and $m \in \delta(\mathrm{Var}_\iota \cup \mathrm{Ter}_\iota \cup \mathrm{Fml}_\iota)$. Since δ is self-referential, there is a number m such that $\delta(\psi_\iota(\dot{m})) = m$. Hence, $\ulcorner \psi_\iota(\dot{m}) \urcorner^\delta = \dot{m}$, and by Leibniz's Law $\vdash \psi_\iota(\ulcorner \psi_\iota(\dot{m}) \urcorner^\delta) \dot{\leftrightarrow} \psi_\iota(\dot{m})$. Now we have

$$\delta \circ \alpha\restriction^{-1}(\alpha(\psi_\iota(\dot{m}))) = \delta(\psi_\iota(\dot{m})).$$

and therefore

$$\mathsf{R}_\iota \vdash f_\iota\left(\ulcorner \psi_\iota(\dot{m}) \urcorner^\alpha, \ulcorner \psi_\iota(\dot{m}) \urcorner^\delta\right).$$

An application of Eq. 5 yields

$$\mathsf{R}_\iota \vdash \varphi_\iota\left(\ulcorner \psi_\iota(\dot{m}) \urcorner^\alpha\right) \dot{\leftrightarrow} \psi_\iota\left(\ulcorner \psi_\iota(\dot{m}) \urcorner^\delta\right).$$

Hence, $\mathsf{R}_\iota \vdash \varphi_\iota\left(\ulcorner \psi_\iota(\dot{m}) \urcorner^\alpha\right) \dot{\leftrightarrow} \psi_\iota(\dot{m})$. Setting $\lambda_\iota = \psi_\iota(\dot{m})$, then yields

$$T_\iota \vdash \varphi_\iota\left(\ulcorner \lambda_\iota \urcorner^\alpha\right) \dot{\leftrightarrow} \lambda_\iota.$$

□

Remark 7.4 The proofs given above provide a new method for constructing syntactic and semantic fixed points. An interesting feature of this method is that it entirely sidesteps the tedious arithmetisation of the numeral function and the substitution function. Thus, self-referential numberings provide a fruitful proof-technique, even if the proved result is formulated with respect to a standard numbering.

7.2 Formalisation Independence

We now introduce precise invariant versions of Gödel's, Tarski's and Church's theorems, thus answering the questions of Section 4.5.

Using Lemma 7.1 and the usual schematic "modal reasoning" we can derive the invariance of Gödel's Second Theorem regarding admissible notation systems and numberings.

Theorem 7.5 *Let $\mathcal{L} \supseteq \mathcal{L}_0$ and let $T \supseteq \mathsf{R}$ be a consistent $\mathcal{L}$-theory. We have $T_\iota \nvdash \dot{\neg}\mathsf{Pr}_\iota^\alpha(\ulcorner \bot_\iota \urcorner^\alpha)$, for every recursive notation system $\langle \mathbf{D}, \iota \rangle$ for $\mathcal{L}$, for every recursive numbering α of $\mathbf{D}$ and for every $\mathcal{L}$-formula $\mathsf{Pr}_\iota^\alpha(x)$ satisfying* $\mathrm{Löb}(T, \iota, \alpha)$.

We immediately obtain from Lemma 7.2 the invariance of Tarski's Theorem regarding admissible notation systems and numberings.

Theorem 7.6 *Let $\langle \mathcal{L}, \mathcal{N} \rangle$ be an arithmetically interpreted language. For every $\mathcal{N}$-definable notation system $\langle \mathbf{D}, \iota \rangle$ for $\mathcal{L}$ and every $\mathcal{N}$-definable numbering α of $\mathbf{D}$, the set $\{\varphi_\iota \in \mathrm{Sent}_\iota \mid \mathcal{N} \models \varphi_\iota\}$ is not definable in $\mathcal{N}$ relative to α.*

Finally, we prove the invariance of Church's Theorem.

Theorem 7.7 *Let $\mathcal{L} \supseteq \mathcal{L}_0$. For every recursive notation system $\langle \mathbf{D}, \iota \rangle$ for $\mathcal{L}$ and every recursive numbering α of $\mathbf{D}$, the function $f : \mathrm{Sent}_\iota \to \omega$, given by*

$$f(\varphi_\iota) := \begin{cases} 0 & \text{if } \models \varphi_\iota; \\ 1 & \text{if } \not\models \varphi_\iota; \end{cases}$$

is not recursive relative to $\langle \alpha, \mathsf{id}_\omega \rangle$.

Proof Let $\mathcal{L} \supseteq \mathcal{L}_0$. Assume that $\{\varphi_\iota \in \mathrm{Sent}_\iota \mid \models \varphi_\iota\}$ is decidable relative to α. Then there is a ι-formula $\psi_\iota(x_\iota)$ which binumerates $\{\varphi_\iota \in \mathrm{Sent}_\iota \mid \models \varphi_\iota\}$ in EA_ι relative to α. Let $\chi_\iota(x_\iota, y_\iota, z_\iota)$ be a ι-formula which binumerates the graph of the constructor operation $\dot{\to}_\iota$ relative to α. We define

$$\mathsf{Pr}_\iota(x_\iota) = \dot{\exists} z_\iota \left(\chi_\iota \left(\ulcorner \dot{\bigwedge} \mathsf{EA}_\iota \urcorner^\alpha, x_\iota, z_\iota \right) \mathbin{\dot{\wedge}} \psi_\iota(z_\iota) \right),$$

where $\dot{\bigwedge}\mathsf{EA}_\iota$, denotes a finite axiomatisation of EA_ι. We now show that $\mathsf{Pr}_\iota(x_\iota)$ binumerates the set of theorems of EA_ι in EA_ι relative to α. If $\mathsf{EA}_\iota \vdash \varphi_\iota$, then $\vdash \dot{\bigwedge}\mathsf{EA}_\iota \dot{\to} \varphi_\iota$ by the deduction theorem. Hence, $\mathsf{EA}_\iota \vdash \psi_\iota(\ulcorner \mathsf{EA}_\iota \dot{\to} \varphi_\iota \urcorner^\alpha)$ and therefore $\mathsf{EA}_\iota \vdash \mathsf{Pr}_\iota(\ulcorner \varphi_\iota \urcorner^\alpha)$. If $\mathsf{EA}_\iota \nvdash \varphi_\iota$, then $\nvdash \dot{\bigwedge}\mathsf{EA}_\iota \dot{\to} \varphi_\iota$ by the deduction theorem. Hence, $\mathsf{EA}_\iota \vdash \dot{\neg}\psi_\iota(\ulcorner \mathsf{EA}_\iota \dot{\to} \varphi_\iota \urcorner^\alpha)$ and therefore $\mathsf{EA}_\iota \vdash \dot{\neg}\mathsf{Pr}_\iota(\ulcorner \varphi_\iota \urcorner^\alpha)$. By Lemma 7.1 there is a ι-sentence λ such that

$$\mathsf{EA}_\iota \vdash \dot{\neg}\mathsf{Pr}_\iota(\ulcorner \lambda_\iota \urcorner^\alpha) \dot{\leftrightarrow} \lambda_\iota. \tag{6}$$

We derive the desired contradiction by showing that neither $\mathsf{EA}_\iota \vdash \lambda_\iota$ nor $\mathsf{EA}_\iota \nvdash \dot{\neg}\lambda_\iota$. If $\mathsf{EA}_\iota \vdash \lambda_\iota$, then $\mathsf{EA}_\iota \vdash \mathsf{Pr}_\iota(\ulcorner \lambda_\iota \urcorner^\alpha)$. Hence by Eq. 6 $\mathsf{EA}_\iota \vdash \dot{\neg}\lambda_\iota$, in contradiction to the consistency of EA_ι. If $\mathsf{EA}_\iota \nvdash \lambda_\iota$, then $\mathsf{EA}_\iota \vdash \dot{\neg}\mathsf{Pr}_\iota(\ulcorner \lambda_\iota \urcorner^\alpha)$. Hence by Eq. 6 $\mathsf{EA}_\iota \vdash \lambda_\iota$, in contradiction to $\mathsf{EA}_\iota \nvdash \lambda_\iota$. □

7.3 Conclusion

It is a well-known phenomenon in metamathematics that the fundamental theorems due to Gödel, Church and Tarski are understood and interpreted in a formalisation

independent way. Yet, their precise formulations rely on several specific formalisation choices. This gap is aptly described by Visser [67], who observes that "[n]o mathematical treatment of G2 correctly reflects our intuitive insight which is independent of the long sequence of design choices usually associated with the formulation of G2" (p. 81). The main purpose of this paper has been to bridge this gap by turning our intuitive insights into precise metamathematical invariance claims. According to my approach, the insight of formalisation independence is thus captured by quantifying over all admissible notation systems and numberings.

Appendix A. Many-Sorted Algebras

Let $S = \{s_0, \dots, s_k\}$ be a set (of sorts), for some $k \in \omega$. The family $A := \langle A^s \rangle_{s \in S}$ is called an *S-sorted set*. We sometimes write $\langle A_0, \dots, A_k \rangle$ for the family A. For technical convenience, I will assume throughout that for any S-sorted set $\langle A^s \rangle_{s \in S}$, we have $A^s \cap A^t = \varnothing$ for all $s \neq t \in S$. We sometimes want to consider the elements of a family A, without distinguishing their sorts. In that case we take the union $\bigcup_{s \in S} A^s$ of all sets A^s, which we also write as $\bigcup A$.

The basic set-theoretic notions can be defined for S-sorted sets in the usual sortwise fashion. More specifically, let A and B be S-sorted sets. We say that A is finite, if A_s is finite for each $s \in S$. We write $A \subseteq B$, and call A an *S-sorted subset of B*, if $A^s \subseteq B^s$ for every $s \in S$. We moreover set $A \cap B := \langle A^s \cap B^s \rangle_{s \in S}$ and define general S-sorted intersections in a similar way. Moreover, we call the S-sorted set $\langle f^s \colon A^s \to B^s \rangle_{s \in S}$ an *S-sorted function*, which we denote by $f \colon A \to B$. We call f *injective* (or *surjective*), if f^s is injective (or surjective) for each $s \in S$. Finally, we write $\varnothing$ for the S-sorted set $\langle \varnothing \rangle_{s \in S}$.

Let $S^\star$ denote the set of finite sequences over S, including the empty sequence λ. In the context of signatures, we exclusively write $s_0 \times \cdots \times s_{k-1}$ for the sequence $(s_0, \dots, s_{k-1})$. An *S-sorted algebraic signature* Ω is a finite $S^\star \times S$-sorted set $\langle \Omega^{w,s} \rangle_{w \in S^\star, s \in S}$ of pairwise disjoint, non-empty finite sets. We say that σ is a symbol of Ω and write $\sigma \in \Omega$, if $\sigma \in \bigcup_{w \in S^\star, s \in S} \Omega^{w,s}$. The elements of $\Omega^{w,s}$ are called *function symbols of type $w \to s$ and of sort s*. If a function symbol has type $\lambda \to s$, we also call it a *constant symbol of sort s*.

Many-sorted algebras consist of a many-sorted *domain*, as well as designated elements and fundamental operations defined on that set. The following definition makes this precise.

Definition A.1 An *Ω-algebra* $\mathbf{A}$ is an ordered pair $\langle \langle |\mathbf{A}|^s \rangle_{s \in S}, \langle \sigma_{\mathbf{A}} \rangle_{\sigma \in \Omega} \rangle$, such that

- $|\mathbf{A}|^s$ are non-empty sets, for all $s \in S$;
- $\sigma_{\mathbf{A}} \in |\mathbf{A}|^s$ if σ is a constant symbol of sort s;
- $\sigma_{\mathbf{A}} \colon |\mathbf{A}|^{w(1)} \times \cdots \times |\mathbf{A}|^{w(\mathrm{lh}(w))} \to |\mathbf{A}|^s$ if σ is a function symbol of type $w \to s$, where $\mathrm{lh}(w)$ denotes the length of w.

We call $\langle |\mathbf{A}|^s \rangle_{s \in S}$ the *domain* of $\mathbf{A}$, which we sometimes write as $|\mathbf{A}|$. If we do not want to distinguish between different sorts, we consider the union $\bigcup_{s \in S} |\mathbf{A}|^s$, which we sometimes write as $\bigcup |\mathbf{A}|$. Finally, we call the functions $\sigma_{\mathbf{A}}$, with $\sigma \in \Omega$, *the fundamental operations of* $\mathbf{A}$.

Let $\mathbf{A}$, $\mathbf{B}$ be Ω-algebras. An Ω-*homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ is an S-sorted function $\alpha\colon \mathbf{A} \to \mathbf{B}$, such that

- $\alpha^s(\sigma_{\mathbf{A}}) = \sigma_{\mathbf{B}}$, for all constant symbols $\sigma \in \Omega$; and
- $\alpha^s(\sigma_{\mathbf{A}}(a_1, \ldots, a_n)) = \sigma_{\mathbf{B}}(\alpha^{s_1}(a_1), \ldots, \alpha^{s_n}(a_n))$, for all $a_i \in |\mathbf{A}|^{s_i}$ (with $1 \leq i \leq n$) and every function symbol $\sigma \in \Omega$ of type $s_1 \times \cdots \times s_n \to s$.

We say that an S-sorted subset B of the domain $|\mathbf{A}|$ is closed under the fundamental operations of $\mathbf{A}$, if for every function symbol $\sigma \in \Omega^{w,s}$ and $b_1 \in B^{w(1)}, \ldots, b_{\mathrm{lh}(w)} \in B^{w(\mathrm{lh}(w))}$ we have $\sigma_{\mathbf{A}}(b_1, \ldots, b_{\mathrm{lh}(w)}) \in B^s$. Let now X be an S-sorted set with $X \subseteq |A|$. We call

$$\mathrm{Cl}_{\mathbf{A}}(X) := \bigcap \{B \mid X \subseteq B \text{ and } B \text{ is closed under the fundamental operations of } \mathbf{A}\}$$

the *algebraic closure* of X. We say that a set U is generated by the fundamental operations of $\mathbf{A}$ from X, if $U = \mathrm{Cl}_{\mathbf{A}}(X)$. We call $\mathbf{A}$ *minimal*, if $\mathrm{Cl}_{\mathbf{A}}(\varnothing) = |\mathbf{A}|$. We call $\mathbf{A}$ *finitely generated*, if there is a finite S-sorted set X such that $\mathrm{Cl}_{\mathbf{A}}(X) = |\mathbf{A}|$.

We now provide an example of a finitely generated algebra that is not single-sorted.

Example A.2 Let S contain the two sorts label and tree. A labelled binary tree is a tree where each node has at most two subtrees, called the left and right subtree respectively. Moreover, each node of the tree is labelled by exactly one element of some non-empty set L of labels. For a graphical representation of a labelled binary tree see Fig. 4.

Let B_L be the set of all labelled binary trees with labels from L. Let $[a]$ be the tree in B_L which only consists of a root with label $a \in L$. Let Ω be a S-sorted signature containing the following function symbols:

- $\mathsf{leaf}\colon \mathrm{label} \to \mathrm{tree}$
- $\mathsf{right}\colon \mathrm{label} \times \mathrm{tree} \to \mathrm{tree}$
- $\mathsf{left}\colon \mathrm{tree} \times \mathrm{label} \to \mathrm{tree}$
- $\mathsf{both}\colon \mathrm{tree} \times \mathrm{label} \times \mathrm{tree} \to \mathrm{tree}$

We now turn B_L into an Ω-algebra $\mathbf{B}_L$ by setting

- $|\mathbf{B}_L|^{\mathrm{label}} = L$ and $|\mathbf{B}_L|^{\mathrm{tree}} = \mathrm{B}_L$;
- $\mathsf{leaf}_{\mathbf{B}_L}(a) = [a]$

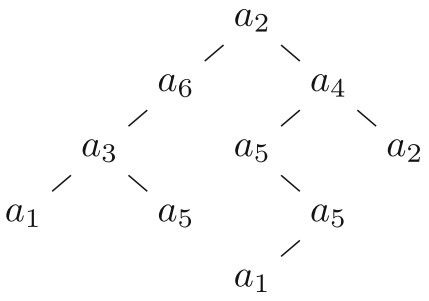


Fig. 4 Labelled binary tree

- $\mathsf{right}_{\mathbf{B}_L}(a, T)$ adds to $[a]$ the right subtree T.
- $\mathsf{left}_{\mathbf{B}_L}(T, a)$ adds to $[a]$ the left subtree T.
- $\mathsf{both}_{\mathbf{B}_L}(T, a, U)$ adds to $[a]$ the left subtree T and the right subtree U.

The algebra $\mathbf{B}_L$ is finitely generated, as long as the set L of labels is finite.

We now introduce the notion of initiality, which is of central importance in abstract algebra.

Definition A.3 Let $\mathfrak{A}$ be a class of Ω-algebras and let $\mathbf{D} \in \mathfrak{A}$. If for every $\mathbf{E} \in \mathfrak{A}$ there is a unique Ω-homomorphism $\alpha^* \colon \mathbf{D} \to \mathbf{E}$, then we say that $\mathbf{D}$ has the *universal property for* $\mathfrak{A}$ and we also call $\mathbf{D}$ *initial in* $\mathfrak{A}$.

If $\mathbf{D}$ is initial in the class $\mathrm{Mod}(\Omega)$ of all Ω-algebras, we also call $\mathbf{D}$ an *absolutely free* Ω*-algebra*.

An important example of an absolutely free algebra is the term algebra $\mathbf{T}_\Omega$, defined as follows.

Definition A.4 For each $s \in S$, the set T_Ω^s of Ω*-terms of sort* s is the smallest set such that

(1) $\Omega^{\lambda, s} \subseteq T_\Omega^s$;
(2) If $\sigma \in \Omega$ is of type $s_1 \times \cdots \times s_n \to s$ and $t_1 \in T_\Omega^{s_1}, \ldots, t_n \in T_\Omega^{s_n}$, then the string $\sigma(t_1, \ldots, t_n)$ is in T_Ω^s.

The S-sorted set T_Ω is called the *Herbrand universe of* Ω. Each element of $\bigcup T_\Omega$ is called an Ω*-term*.

We can naturally transform the Herbrand universe T_Ω of Ω into an Ω-algebra $\mathbf{T}_\Omega$ as follows:

Definition A.5 Let T_Ω be the Herbrand universe of Ω. We turn T_Ω into an Ω-algebra $\mathbf{T}_\Omega$ by setting

- $\sigma_{\mathbf{T}_\Omega} := \sigma$, for each constant symbol $\sigma \in \Omega$;
- $\sigma_{\mathbf{T}_\Omega}(t_1, \ldots, t_n) := \sigma(t_1, \ldots, t_n)$, for each function symbol $\sigma \in \Omega$ of type $s_1 \times \cdots \times s_n \to s$ and terms $t_1 \in T_\Omega^{s_1}, \ldots, t_n \in T_\Omega^{s_n}$.

We call the resulting Ω-algebra $\mathbf{T}_\Omega$ the *term algebra* or *Herbrand algebra* of Ω.

Appendix B. Quine-Bourbaki notation

In what follows, I show how the Quine-Bourbaki notation system (see Section 3.4) can be accommodated in the framework developed in Section 4.

Let A be the finite alphabet which contains the symbols of $\mathcal{L}_0$ together with the auxiliary symbols 'v', '$\prime$', '$\square$', '(' and ')'. Let $\mathbb{HF}_A$ be the set of hereditarily finite sets with the elements of A as *urelements*. $\mathbb{HF}_A$ together with the adjunction operation $\mathcal{A}$ given by $\mathcal{A}(x, y) := x \cup \{y\}$ forms a finitely generated algebra $\mathbf{HF}_A$. Assume that finite sequences of elements of $\mathbb{HF}_A$ and the concatenation operation $*$ on finite sequences are defined in $\mathbb{HF}_A$ in a fixed and standard way.

It will be convenient to conceive of A-strings as sequences $s_0 * \cdots * s_n$, where each sequence entry s_i is either a symbol of A or the result of appending finitely many strokes $\prime$ to v such that $\mathsf{v} * \prime$ is not a subsequence of $s_0 * \cdots * s_n$. Call such a sequence *simplified*. In simplified sequences, we treat A-strings of the form $\mathsf{v}\prime \ldots \prime$ as "alphabetical symbols" of length 1.

Consider now an ordered pair $\langle s_0 * \cdots * s_n, C\rangle$, such that the sequence $s_0 * \cdots * s_n$ is simplified and such that C is a (possibly empty) set of unordered pairs $\{i, j\}$ with $i \neq j \leq n$. In what follows, the ordered pair $\langle s_0 * \cdots * s_n, C\rangle$ will represent the result of adding curved lines to the A-string $s_0 * \cdots * s_n$ as follows: there is a curved line between s_i and s_j iff $\{i, j\} \in C$.

We now define an implementation ι of p-expressions of $\mathcal{L}_0$ into $\mathbf{HF}_A$. We write the projection to the i-th component as $\langle \cdot, \cdot\rangle_i$, for $i = 1, 2$. For example, $\langle s, C\rangle_1 = s$. Moreover, for any unordered pair $\{i, j\}$ of numbers, we write $\{i, j\}+n$ for $\{i+n, j+n\}$, for $n \in \omega$.

We first define the implementation ι^{var} of p-variables into $\mathbf{HF}_A$ recursively as follows.

$$\begin{aligned}\iota^{\mathrm{var}}(\mathtt{v}) &:= \langle \mathsf{v}, \varnothing\rangle\\ \iota^{\mathrm{var}}(\mathtt{nvar}(x)) &:= \langle \iota(x)_1\, \prime, \varnothing\rangle\end{aligned}$$

We now define the implementation ι^{ter} of p-terms recursively as follows:

$$\begin{aligned}\iota^{\mathrm{ter}}(\mathtt{e}(x)) &:= \iota^{\mathrm{var}}(x)\\ \iota^{\mathrm{ter}}(\mathtt{fct_0}) &:= \langle \mathsf{0}, \varnothing\rangle\\ \iota^{\mathrm{ter}}(\mathtt{fct_S}(t)) &:= \langle \mathsf{S} * \iota^{\mathrm{ter}}(t)_1, \varnothing\rangle\\ \iota^{\mathrm{ter}}(\mathtt{fct}_+(s, t)) &:= \langle (*\iota^{\mathrm{ter}}(s)_1 * + * \iota^{\mathrm{ter}}(t)_1*), \varnothing\rangle\\ \iota^{\mathrm{ter}}(\mathtt{fct}_\times(s, t)) &:= \langle (*\iota^{\mathrm{ter}}(s)_1 * \times * \iota^{\mathrm{ter}}(t)_1*), \varnothing\rangle\end{aligned}$$

We finally define the implementation ι^{fml} of p-formulas:

$$\begin{aligned}\iota^{\mathrm{fml}}(\mathtt{equ}(s, t)) &:= \langle (*\iota^{\mathrm{ter}}(s)_1 * = * \iota^{\mathrm{ter}}(t)_1*), \varnothing\rangle\\ \iota^{\mathrm{fml}}(\mathtt{neg}(\varphi)) &:= \langle \neg * \iota^{\mathrm{fml}}(\varphi)_1, \{P + 1 \mid P \in \iota^{\mathrm{fml}}(\varphi)_2\}\rangle\\ \iota^{\mathrm{fml}}(\mathtt{conj}(\varphi, \psi)) &:= \langle (*\iota^{\mathrm{fml}}(\varphi)_1 * \wedge * \iota^{\mathrm{fml}}(\psi)_1*), \{P + 1 \mid P \in \iota^{\mathrm{fml}}(\varphi)_2\}\cup\\ &\qquad \{P + \mathrm{lh}(\iota^{\mathrm{fml}}(\varphi)_1) + 2 \mid P \in \iota^{\mathrm{fml}}(\psi)_2\}\rangle\\ &\ \ \vdots\\ \iota^{\mathrm{fml}}(\mathtt{univ}(x, \varphi)) &:= \langle \forall * \iota^{\mathrm{fml}}(\varphi)_1[\square/\iota^{\mathtt{var}}(x)], \{P + 1 \mid P \in \iota^{\mathrm{fml}}(\varphi)_2\}\cup\\ &\qquad \{\{1, n\} \mid x \text{ occurs in the } n\text{-th entry of } \iota^{\mathrm{fml}}(\varphi)_1\}\rangle\end{aligned}$$

$$\iota^{\mathrm{fml}}(\texttt{exists}(x,\varphi)) := \langle \exists * \iota^{\mathrm{fml}}(\varphi)_1[\Box/\iota^{\mathtt{var}}(x)], \{P+1 \mid P \in \iota^{\mathrm{fml}}(\varphi)_2\} \cup \{\{1,n\} \mid x \text{ occurs in the } n\text{-th entry of } \iota^{\mathrm{fml}}(\varphi)_1\}\rangle$$

where $\iota^{\mathrm{fml}}(\varphi)_1[\Box/\iota^{\mathtt{var}}(x)]$ is the result of substituting each occurrence of $\iota^{\mathtt{var}}(x)$ in $\iota^{\mathrm{fml}}(\varphi)_1$ by $\Box$. We observe that $\iota^{\mathrm{fml}}(\varphi)_1$ and $\iota^{\mathrm{fml}}(\varphi)_1[\Box/\iota^{\mathtt{var}}(x)]$ are simplified sequences of the same length.

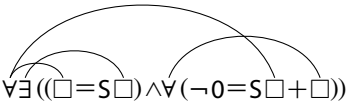
For any (distinct) p-variables x, y, z, the p-expression

$$\texttt{univ}(x, \texttt{exists}(y, \texttt{conj}(\texttt{equ}(x, \texttt{fct}_{\mathsf{S}} y), \texttt{univ}(z, \texttt{neg}(\texttt{equ}(\texttt{fct}_0, \texttt{fct}_+(\texttt{fct}_{\mathsf{S}} x, y)\texttt{fct}_{\mathsf{S}} y))))))$$

is implemented by ι as the same object, viz.

$$\langle\, \forall\exists((\Box = \mathsf{S}\Box) \wedge \forall(\neg(0 = (\mathsf{S}\Box + \Box)))), \{\{1,5\},\{1,19\},\{2,7\},\{11,21\}\}\,\rangle$$

which corresponds to the following Quine-Bourbaki notation (with the omission of some parentheses):



More generally, ι-equivalence coincides with α-equivalence, i.e., syntactic identity up to renaming of bounded variables.

Acknowledgements I am indebted to Arnon Avron, Volker Halbach, Léon Probst, Gil Sagi, Albert Visser and two anonymous referees for their detailed and helpful comments. I would also like to thank Nachum Dershowitz, Rea Golan, Michael Goldboim, David Kashtan, Carlo Nicolai, Lavinia Picollo, Carl Posy, Johannes Stern, Dan Waxman and Lingyuan Ye for helpful conversations on topics presented in this paper. Finally, I am grateful to the audiences of my talks on this material for their valuable feedback.

Author Contributions Not applicable.

Funding Open Access funding enabled and organized by Projekt DEAL. Open Access funding enabled and organized by Projekt DEAL.

Data Availability Not applicable.

Declarations

Ethical Approval Not applicable.

Competing interests No competing interests.

References

1. Avigad, J. (2022). *Mathematical Logic and Computation*. Cambridge University Press.
2. Bergstra, J., & Tucker, J. (1987). Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science, 50*(2), 137–181.
3. Béziau, J.-Y. (1999). The mathematical structure of logical syntax. In W. Carnielli & I. M. L. D'Ottaviano (Eds.), *Advances in contemporary logic and computer science* (pp. 1–17). American Mathematical Society.
4. Boolos, G. S. (1994). *The Logic of Provability*. Cambridge University Press.
5. Bourbaki, N. (1954). *Théorie des Ensembles*. Paris: Hermann.
6. Button, T., & Walsh, S. (2018). *Philosophy and Model Theory*. Oxford University Press.
7. Cheng, Y. (2021). Current research on Gödel's incompleteness theorems. *The Bulletin of Symbolic Logic*, pp. 1–52
8. Corcoran, J., Frank, W., & Maloney, M. (1974). String theory. *Journal of Symbolic Logic, 39*(4), 625–637.
9. Corry, L., & Schappacher, N. (2010). Zionist internationalism through number theory: Edmund Landau at the opening of the Hebrew University in 1925. *Science in Context, 23*(4), 427–471.
10. De Bruijn, N. G. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae, 75*(5), 381–392.
11. Dummett, M. (1973). *Frege: Philosophy of Language*. Number 1. Harper & Row.
12. Evans, G. (1977). Pronouns, quantifiers, and relative clauses (i). *Canadian Journal of Philosophy, 7*(3), 467–536.
13. Feferman, S. (1960). Arithmetization of metamathematics in a general setting. *Fundamenta Mathematicae, 49*, 35–92.
14. Feferman, S. (1984). Toward useful type-free theories. i. *J. Symbolic Logic*, 49(1), 75–111
15. Feferman, S. (1994). Finitary inductively presented logics. In D. M. Gabbay (Ed.), *What is a Logical System?* (pp. 297–328). Oxford University Press.
16. Fitting, M. (2007). *Incompleteness in the Land of Sets*. College Publications.
17. Fitting, M. (2017). Russell's paradox, Gödel's theorem. In *Raymond Smullyan on Self Reference*, pp. 47–66. Springer.
18. Frege, G. (1879). *Begriffsschrift*. Halle: Louis Nebert.
19. Frege, G. (1919). Die Verneinung. Eine logische Untersuchung. *Beiträge zur Philosophie des deutschen Idealismus*, 1, 143–157. Reprinted as 'Negation' in Frege, G. (1984). Collected papers on mathematics, logic, and philosophy. Blackwell, Oxford, UK. Page references are given for the translation.
20. Frege, G. (1923). Logische Untersuchungen. Dritter Teil: Gedankengefüge. *Beiträge zur Philosophie des deutschen Idealismus*, 3, 36–51. Reprinted as 'Compound Thoughts' in Frege, G. (1984). Collected papers on mathematics, logic, and philosophy. Blackwell, Oxford, UK. Page references are given for the translation.
21. Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik*, 38(1), 173–198. Reprinted and translated in Gödel, K. (1986). Collected Works. Vol. 1: Publications 1929–1936. Oxford University Press, Oxford. pp. 144–195
22. Grabmayr, B. (2021). On the invariance of Gödel's second theorem with regard to numberings. *Review of Symbolic Logic, 14*(1), 51–84.
23. Grabmayr, B. (2021b). *On the Philosophical Interpretation of Metamathematical Results*. PhD thesis, Humboldt University of Berlin.
24. Grabmayr, B., Halbach, V., & Ye, L. (2023). Varieties of self-reference in metamathematics. *Journal of Philosophical Logic, 52*(4), 1005–1052.
25. Grabmayr, B., & Visser, A. (2023). Self-reference upfront: A study of self-referential Gödel numberings. *Review of Symbolic Logic, 16*(2), 385–424.
26. Hájek, P., & Pudlák, P. (1998). *Metamathematics of First-Order Arithmetic*. Berlin, Heidelberg: Springer.

27. Halbach, V., & Visser, A. (2014). Self-reference in arithmetic I. *Review of Symbolic Logic, 7*(4), 671–691.
28. Halbach, V., & Visser, A. (2014). Self-reference in arithmetic II. *Review of Symbolic Logic, 7*(4), 692–712.
29. Hausdorff, F. (1914). *Grundzüge der Mengenlehre*. Leipzig: Veit and Company.
30. Heck, R. K. (2007). Self-reference and the languages of arithmetic. *Philosophia Mathematica, 15*(1), 1–29.
31. Horsten, L. (2011). *The Tarskian Turn: Deflationism and Axiomatic Truth*. MIT Press.
32. Humberstone, L. (2011). *The Connectives*. MIT Press.
33. Kaplan, D. (1986). Opacity. In L. E. Hahn & P. A. Schilpp (Eds.), *The Philosophy of W* (Vol. Quine, pp. 229–289). Open Court.
34. Kleene, S. C. (1952). *Introduction to Metamathematics*. North Holland.
35. Kleene, S. C. (1967). *Mathematical Logic*. Wiley.
36. Kracht, M. (2003). *The Mathematics of Language*. Walter de Gruyter.
37. Kripke, S. A. (1975). Outline of a theory of truth. *Journal of Philosophy, 72*(19), 690–716.
38. Kripke, S. A. (2021). *Gödel's theorem and direct self-reference*. Online First: Review of Symbolic Logic.
39. Kunen, K. (2009). *The Foundations of Mathematics*. College Publications.
40. Kuratowski, C. (1921). Sur la notion de l'ordre dans la théorie des ensembles. *Fundamenta Mathematicae, 2*, 161–171.
41. Löb, M. H. (1955). Solution of a problem of Leon Henkin. *Journal of Symbolic Logic, 20*(2), 115–118.
42. Mal'cev, A. I. (1961). Constructive Algebras I. *Russian Mathematical Surveys*, 16, 77–129. Reprinted and translated in Wells, B. F. (1971). The Metamathematics of Algebraic Systems. North-Holland, Amsterdam. pp. 148–214
43. Marker, D. (2002). *Model Theory: An Introduction (Graduate Texts in Mathematics, Vol. 217)*. Springer, 2002 edition.
44. McGee, V. (1991). *Truth*. Hackett: Vagueness and Paradox.
45. Oberschelp, A. (1990). Order sorted predicate logic. In K. H. Bläsius, U. Hedtstück, & C. -R. Rollinger (Eds.), *Sorts and Types in Artificial Intelligence*, pp. 7–17, Berlin, Heidelberg. Springer Berlin Heidelberg.
46. Paulson, L. (2015). A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. *Journal of Automated Reasoning, 55*(1), 1–37.
47. Peano, G. (1889). *Arithmetices Principia Novo Methodo Exposita*. Turin: Fratelli Bocca.
48. Peregrin, J. (2020). *Philosophy of Logical Systems*. Routledge.
49. Picollo, L. (2018). Reference in arithmetic. *Review of Symbolic Logic, 11*(3), 573–603.
50. Quine, W. V. (1940). *Mathematical Logic*. W. W. Norton, New York, 1 edition.
51. Quine, W. V. (1966). The ways of paradox. In W. V. Quine (Ed.), *The Ways of Paradox and Other Essays* (pp. 3–20). New York: Random House.
52. Ramsey, F. P. (1927). Facts and propositions. *Proceedings of the Aristotelian Society, 7*, 153–170.
53. Rasiowa, H., & Sikorski, R. (1963). *The Mathematics of Metamathematics*. Drukarnia Uniwersytetu Jagiellonskiego W Krakowie: Monografie matematyczne.
54. Sato, M. (2018). The data structures of the lambda terms. In K. Mainzer, P. Schuster, & H. Schwichtenberg (Eds.), *Proof And Computation: Digitization In Mathematics, Computer Science And Philosophy* (pp. 191–208). World Scientific Publishing.
55. Smith, P. (2003). *An Introduction to Formal Logic*. Cambridge University Press.
56. Smullyan, R. M. (1957). Languages in which self reference is possible. *Journal of Symbolic Logic, 22*(1), 55–67.
57. Smullyan, R. M. (1961). *Theory of Formal Systems*. Princeton, NJ: Princeton University Press.
58. Smullyan, R. M. (1992). *Gödel's Incompleteness Theorems*. Oxford: Oxford University Press.
59. Smullyan, R. M. (1994). *Diagonalization and Self-Reference*. Clarendon Press.
60. Świerczkowski, S. (2003). Finite sets and Gödel's incompleteness theorems. *Dissertationes Mathematicae, 422*, 1–58.
61. Tarski, A., Mostowski, A., & Robinson, R. (1953). *Undecidable Theories*. North-Holland, Amsterdam: Studies in logic and the foundations of mathematics.
62. Tennant, N. (1978). *Natural Logic*. Edinburgh University Press.
63. Vermeulen, C. F. M. (2000). Text structure and proof structure. *Journal of Logic, Language, and Information, 9*(3), 273–311.

64. Visser, A. (1989). Semantics and the liar paradox. In D. Gabbay & F. Guenthner (Eds.), *Handbook of Philosophical Logic* (Vol. IV, pp. 617–706). Dordrecht: Reidel.
65. Visser, A. (2006). Categories of theories and interpretations. Lecture Notes in LogicIn A. Enayat, I. Kalantari, & M. Moniri (Eds.), *Logic in Tehran* (pp. 284–341). Cambridge University Press.
66. Visser, A. (2011). Can we make the Second Incompleteness Theorem coordinate free? *Journal of Logic and Computation, 21*(4), 543–560.
67. Visser, A. (2016). The second incompleteness theorem: Reflections and ruminations. In L. Horsten & P. Welch (Eds.), *Gödel's Disjunction: The Scope and Limits of Mathematical Knowledge* (pp. 67–91). Oxford University Press.
68. Wehmeier, K. (2018). The proper treatment of variables in predicate logic. *Linguistics and Philosophy, 41*(2), 209–249.