

Secondary Publication



Vestrucci, Andrea

Diagonalization as computation : Gödel sentence construction and a type-disciplined blueprint

Date of secondary publication: 05.10.2026

Version of Record (Published Version), Article

Persistent identifier: urn:nbn:de:bvb:473-irb-117550x

Primary publication

Vestrucci, Andrea (2026): Diagonalization as computation : Gödel sentence construction and a type-disciplined blueprint, in: Archive for mathematical logic, Berlin ; Heidelberg: Springer, Vol. 65, No. 6, pp. 761–799, doi: 10.1007/s00153-026-01021-7.

Legal Notice

This work is protected by copyright and/or the indication of a licence. You are free to use this work in any way permitted by the copyright and/or the licence that applies to your usage. For other uses, you must obtain permission from the rights-holders.

This document is made available under a Creative Commons license.



The license information is available online:

<https://creativecommons.org/licenses/by/4.0/legalcode>



Diagonalization as computation: Gödel sentence construction and a type-disciplined blueprint

Andrea Vestrucci^{1,2}

Received: 14 March 2025 / Accepted: 4 May 2026 / Published online: 8 June 2026
© The Author(s) 2026

Abstract

We present a type-disciplined, compositional reconstruction of the diagonal strategy in Gödel’s first incompleteness theorem. The goal is not a new or “more direct” proof of incompleteness, but an explicit computational unpacking of the mechanism. The paper exhibits a concrete pipeline of code-transformers between primitive recursive syntactic domains, thereby avoiding well-definedness issues arising in Lindenbaum-quotient formulations. A central feature is the separation between a graph-mediated witness form of the Gödel sentence and a canonical normal form, connected by an internal normalization lemma formalizable in the base theory. The paper also presents the construction categorically in a category of primitive recursive code domains and morphisms, and extracts a general diagonal-flipping blueprint clarifying how type stratification distinguishes paradoxical from limitative diagonal arguments.

Keywords Gödel’s first incompleteness theorem · Diagonalization · Gödel numbering · Primitive recursive functions · Lawvere fixed-point theorem · Diagonal arguments

Mathematics Subject Classification 03F40 · 03F45 · 03D20 · 03G30

1 Introduction

The interrelation between diagonal constructions and Gödel’s first incompleteness theorem (henceforth “G1”) is a foundational theme of mathematical logic. Gödel’s 1931 argument [10, 11] proceeds by arithmetizing syntax and performing a diagonal substitution on class-signs (one-free-variable wffs), thereby constructing a sentence that asserts the unprovability of its own Gödel number (“GN” henceforth). See [29] for historical material, [2, 5] for systematic survey discussions of the scope and limits

✉ Andrea Vestrucci
andrea.vestrucci@uni-bamberg.de; andrea.vestrucci@gmail.com

¹ Department of Computer Science, University of Bamberg, Kapuzinerstraße 16, Bamberg 96047, Germany

² ILT College, 910 4th St, Brookings 57006, SD, USA

of applicability of G1 and comparisons of modern formulations and proofs, and [33] for an overview.

Later presentations often reorganize the same mechanism through a general fixed-point/diagonal lemma - frequently attributed, in its abstract form, to Carnap [7] - and then obtain the Gödel sentence as an instance [2, pp. 886–887]. This route is conceptually clean, but it can also make the Gödel sentence appear to be obtained by an existence theorem or even to arise by *fiat*, with an “aura of magician’s trick” [9, p. 710], hence leaving unresolved how to *compute* a concrete representative of the independent sentence [5, p. 517].

This paper addresses this issue by exhibiting a type-disciplined construction on codes. The novelty is not theorem-level but mechanism-level. Instead of presenting a new or “more direct” proof of G1, our contribution recasts Gödel’s strategy as an explicit compositional primitive recursive (p.r.) computation on GNs: it decomposes the construction of the Gödel sentence into explicit p.r. code-transformers whose typing and normalization behavior are tracked separately.¹ This pipeline supports (i) a categorical reformulation of the construction in which diagonalization is a specific computable morphism rather than an ambient principle, (ii) a modular scheme for variants (e.g. Rosser-style provability), and (iii) a general diagonal-flipping blueprint clarifying how type stratification distinguishes paradoxical from limitative diagonal arguments.

This perspective is influenced by Yanofsky’s function-composition reading of diagonal arguments [42], itself inspired by categorical fixed-point phenomena studied by Lawvere [21, 22]. However, the present development differs in both emphasis and technical level. First, it moves from Lindenbaum-style quotienting to codes, since operations like substitution depend on syntactic form and are in general not well-defined on provability-equivalence classes. Second, it separates two logically distinct outputs: a *witness form* of the Gödel sentence, where self-reference is mediated through a graph predicate for diagonalization, and a canonical *normal form* in the familiar “unprovability” shape. The two forms are connected by an internal normalization step; this step is itself provable in the base theory T (a p.r.-axiomatized extension of Q): this makes explicit how the canonical unprovability normal form is recovered from the witness. This code-transformer pipeline makes the diagonal equivalence an explicit computation, i.e., an “unwinding” of the composite.

The categorical reformulation proceeds as follows. We introduce a category **PRCodes** whose objects are p.r. syntactic domains (e.g. codes of one-free-variable formulas and codes of sentences) and whose morphisms are total p.r. code-transformers preserving these domains. In **PRCodes**, the witness and normal forms of the Gödel sentence are obtained as the value of a composite of morphisms applied to a distinguished global element. This categorical reading also clarifies its relationship to other categorical accounts of diagonalization.

Based on this, we extract a general *diagonal-flipping blueprint* shared by a wide family of classical diagonal arguments (Cantor, Russell, Grelling–Nelson, Turing,

¹ As Beklemishev emphasizes [2, p. 867], from a modern computability-theoretic viewpoint Gödel’s p.r. effectiveness assumptions are not essential: one can generalize to computably enumerable axiomatizations without changing the scope of incompleteness. Yet the p.r. setting is well suited to our type-disciplined code-transformer reconstruction.

Gödel). The blueprint separates the construction of a diagonal object (or diagonal data) from the internal representability of its “anti-diagonal” behavior, and it highlights the role of type discipline in explaining why some diagonal-flipping arguments yield paradoxes while others yield limitation theorems. This blueprint provides a unifying methodological perspective that, we suggest, is of independent interest: it clarifies what is being refuted in each setting (an additional closure/representation principle) and how stratification between “objects” and “classifiers” controls the outcome of the diagonal move.

In summary, the paper proposes a clarifying reorganization of the syntactic core of G1: (a) a type-correct, code-level decomposition of the construction into explicit p.r. components; (b) an internal normalization step that cleanly separates witness and canonical normal forms; (c) a modular view of variants via replacement of the flip map; and (d) a general blueprint and type analysis that isolates the shared compositional skeleton of diagonal-flipping arguments across mathematics and logic.

The paper is organized as follows. Section 2 briefly recalls Yanofsky’s compositional perspective. Section 3 motivates the move to code-level typing and presents the p.r. pipeline and the witness/normal-form distinction, including the Normalization Lemma. Section 4 develops the categorical reading in **PRCodes** and relates it to existing categorical accounts. Section 5 illustrates the modularity of the pipeline by varying the “flip” component (including Rosser’s strengthening) and by alternative matrix presentations. Finally, Section 6 formulates the diagonal-flipping blueprint and its type-discipline analysis across Cantor, Russell, Grelling–Nelson, Turing, and Gödel.

2 Yanofsky: Fixed point construction

This section summarizes the results in [42] relative to G1. The GN of a formula ϕ is denoted $\ulcorner \phi \urcorner$, as customary. Yanofsky works in a mixed GN/numeral language; we follow this in this section, but from the next section on we distinguish GN from numerals. Also, for clarity, we identify the functions introduced by Yanofsky with the superscript ^{y} .

Let $Lind^n$ denote the Lindenbaum algebra of wffs with n free variables in the language of theory T modulo provable logical equivalence in T , such that $\varphi \sim \psi \Leftrightarrow \vdash_T (\varphi \leftrightarrow \psi)$. We consider the function $\Delta^y : Lind^1 \rightarrow Lind^1 \times Lind^1$, which sends every $A \in Lind^1$ to $(A, A) \in Lind^1 \times Lind^1$.

Then Yanofsky introduces function $f^y : Lind^1 \times Lind^1 \rightarrow Lind^0$, as follows:

$$f^y(A(x), B(y)) = B(\ulcorner A(x) \urcorner)$$

This application maps each pair of one-free-variable wffs into the closed sentence built by substituting the GN of the former wff to the free variable of the latter. Strictly speaking, this function is not well-defined on Lindenbaum classes, but we follow Yanofsky’s informal presentation here and return to this issue in the next section.

Assume two more functions: first, diagonalization function $D^y : \mathbb{N} \rightarrow \mathbb{N}$ defined as $D^y(\ulcorner A(x) \urcorner) = \ulcorner \ulcorner A(x) \urcorner \urcorner$ for all wffs $A(x)$ with one free variable; second,

function $\alpha_{\mathcal{E}}^y : Lind^0 \rightarrow Lind^0$, which depends on some predicate $\mathcal{E}$: the function applies the predicate to the GN of a sentence H (with no free variable): $\alpha_{\mathcal{E}}^y(H) = \mathcal{E}(\ulcorner H \urcorner)$.

From this, Yanofsky is able to build a new function g^y resulting from the composition of f^y and $\alpha_{\mathcal{E}}^y$:

$$g^y(A(x)) = \alpha_{\mathcal{E}}^y(f^y(A(x), A(x))) = \mathcal{E}(D^y(\ulcorner A(x) \urcorner)) = \mathcal{E}(\ulcorner A(\ulcorner A(x) \urcorner) \urcorner)$$

Yanofsky establishes g^y to be representable by the one-free-variable wff $G(x)$, such that $\mathcal{E}(D^y(x)) = G(x)$, where x ranges over GNs of a $Lind^1$ wff. Thus, for an arbitrary $Lind^1$ wff $H(x)$,

$$g^y(H(x)) = G(\ulcorner H(x) \urcorner) = f^y(H(x), G(y)).$$

It follows that $G(\ulcorner G(x) \urcorner)$ is the fixed point for the function $\alpha_{\mathcal{E}}^y$:

$$\begin{aligned} \mathcal{E}(\ulcorner G(\ulcorner G(x) \urcorner) \urcorner) &= \alpha_{\mathcal{E}}^y(\ulcorner G(\ulcorner G(x) \urcorner) \urcorner) \\ &= \alpha_{\mathcal{E}}^y(f^y(G(x), G(x))) \\ &= g^y(G(x)) \\ &= f^y(G(x), G(x)) \\ &= G(\ulcorner G(x) \urcorner) \end{aligned}$$

Yanofsky concludes that $\vdash_T G(\ulcorner G(x) \urcorner) \leftrightarrow \mathcal{E}(\ulcorner G(\ulcorner G(x) \urcorner) \urcorner)$. As is customary in the literature, from the diagonal lemma the demonstration of G1 follows.

3 Direct construction of the independent sentence

Building on Yanofsky's function-composition approach, we construct the independent sentence as the output of an explicit p.r. code-transformer. The construction is direct in the sense that we do not first invoke a general fixed-point lemma and only then extract the Gödel sentence as an instance. Instead, it is obtained by unwinding a concrete composite of p.r. maps on GNs.

3.1 From classes to codes

From here on, we work with *codes* rather than provability-equivalence classes. This move avoids a well-definedness problem: operations such as substitution depend on the syntactic form of formulas, so in general they do not induce functions on Lindenbaum quotients.

For example, Yanofsky defines $f^y : Lind^1 \times Lind^1 \rightarrow Lind^0$ as:

$$f^y(A(x), B(y)) := B(\ulcorner A(x) \urcorner)$$

For f^y to be well-defined on Lindenbaum classes, it would have to respect provable equivalence, i.e.:

$$A \sim A' \implies B(\ulcorner A \urcorner) \sim B(\ulcorner A' \urcorner) \text{ for all } B$$

But this invariance fails in general: even if $A \sim A'$, the GNs $\ulcorner A \urcorner$ and $\ulcorner A' \urcorner$ may code different numbers, and nothing guarantees that $f^y(A, B) \sim f^y(A', B)$, i.e. that the corresponding substitutions yield provably equivalent outputs. Thus, the Lindenbaum class of $B(\ulcorner A \urcorner)$ can depend on the chosen representative of $[A]$, so f^y does not descend to the quotient.

In sum, the operations involved in diagonalization are syntax-driven and are not invariant under provable equivalence. This motivates redefining the relevant maps directly on codes (i.e., GNs). As we will see, type discipline - here as tracking which code-transformers preserve which syntactic domains - is a *Leitmotiv* of this paper's approach.

Fix a consistent, p.r. axiomatized theory T that extends Q —so that p.r. functions and relations are representable in T . Let $\text{Wff}_1 \subseteq \mathbb{N}$ be the set of GN of wffs whose set of free variables is exactly $\{v_0\}$ (a “class sign” in Gödel's terminology [11, p. 147]). We work up to α -equivalence and adopt v_0 as the distinguished free variable. Accordingly, there is a p.r. normalization map sending any one-free-variable wff code to the code of an α -variant in Wff_1 .

Next, let $\text{Sent} \subseteq \mathbb{N}$ be the set of GN of sentences. Let finite products be implemented via a fixed p.r. pairing function $\langle \cdot, \cdot \rangle$ with p.r. projections π_1 and π_2 , in particular $\text{Wff}_1 \times \text{Wff}_1$.

On these, we now redefine, at the level of codes, the functions corresponding to the maps Δ^y and f^y used by Yanofsky. We distinguish the code functions by the superscript c .

Duplication. We define the duplication function

$$\Delta^c : \mathbb{N} \rightarrow \mathbb{N}, \quad \Delta^c(a) = \langle a, a \rangle.$$

This is the code-level version of Yanofsky's $\Delta^y : \text{Lind}^1 \rightarrow \text{Lind}^1 \times \text{Lind}^1$.

Substitution. We define a total substitution function $\text{Sub} : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that given the code $a = \ulcorner A(v_0) \urcorner$ and a natural number b ,

$$\text{Sub}(a, b) = \ulcorner A(\bar{b}) \urcorner \text{ and if } a \in \text{Wff}_1 \text{ then } \text{Sub}(a, b) \in \text{Sent},$$

where $\bar{b}$ is the standard numeral for b in the language of T . As we will see in section 6, b can easily be, if needed, the GN of a one-free-variable wff, i.e., $b \in \text{Wff}_1$. For $a \notin \text{Wff}_1$ we fix an arbitrary default value, e.g. $\text{Sub}(a, b) = 0$, hence ensuring totality.

Hence, Yanofsky's substitution function $f^y(A(v_0), B(v_0))$ corresponds, at the code level, to function $f^c : \mathbb{N} \rightarrow \mathbb{N}$ on paired product codes such that,

$$f^c(p) = \begin{cases} \text{Sub}(\pi_2(p), \pi_1(p)) & \text{if } \pi_1(p), \pi_2(p) \in \text{Wff}_1, \\ 0 & \text{otherwise.} \end{cases}$$

With $a = \ulcorner A(v_0) \urcorner \in \text{Wff}_1$ and $b = \ulcorner B(v_0) \urcorner \in \text{Wff}_1$, we have the following equality with Sub :

$$f^c(\langle a, b \rangle) = \text{Sub}(b, a) = \ulcorner B(\bar{a}) \urcorner$$

since per construction, Yanofsky's f^y substitutes the GN of A into the free variable of B .

The function Sub is p.r. since parsing Gödel codes of Wff_1 , forming numerals $\bar{b}$, and performing capture-avoiding substitution are p.r. in any fixed coding. Hence, the graph of Sub is representable in T by a wff $\text{Sub}_T(x, y, z)$ such that

- Sub_T preserves correctness on numerals: for all $a, b, c \in \mathbb{N}$, if $\text{Sub}(a, b) = c$ then $T \vdash \text{Sub}_T(\bar{a}, \bar{b}, \bar{c})$.
- T proves Sub_T to be functional (i.e., it generates a unique output); formally: $T \vdash \forall x y z z' (\text{Sub}_T(x, y, z) \wedge \text{Sub}_T(x, y, z') \rightarrow z = z')$.

Diagonalization. Composing Δ^c with Sub gives the meta-level diagonalization function $D^c : \mathbb{N} \rightarrow \mathbb{N}$ by

$$D^c(x) = \text{Sub}(x, x).$$

Hence, for all $a \in \text{Wff}_1$ we have

$$D^c(a) = f^c(\Delta^c(a)) \quad \text{and} \quad D^c(a) \in \text{Sent}.$$

Being a special case of Sub , D^c is also p.r. Accordingly, D^c is representable in T by the formula

$$\text{Diag}_T(x, y) := \text{Sub}_T(x, x, y).$$

3.2 Building the witness and normalized forms of the independent sentence

In what follows we show how code computation can lead to the construction of the canonical normal form of the independent sentence, as well as the witness of that normal form.

We consider the usual relation $\text{Prf}_T(m, n)$ to stand for “ m is the GN of a T -proof of the formula with GN n .” The relation Prf_T is p.r.: checking that a coded finite sequence is a correct T -derivation and that its last line has code n are all operations that involve finitely many bounded arithmetical checks. Hence Prf_T is representable in T by the corresponding object-level wff $\text{Prf}_T(m, n)$.

From this we define Prov_T , the wff representing the provability predicate in T , such that:

$$\text{Prov}_T(n) := \exists m. \text{Prf}_T(m, n).$$

Next, we define the meta-level p.r. code-builder $g^c : \text{Sent} \rightarrow \text{Sent}$ by

$$g^c(n) := \ulcorner \neg \text{Prov}_T(\bar{n}) \urcorner = \ulcorner \neg \exists m \text{Prf}_T(m, \bar{n}) \urcorner,$$

i.e. g^c maps a sentence code n to the code of the sentence asserting the unprovability (in T) of the sentence with code n . Hence, g^c can be considered a meta-level “provability flipping” code-builder. This is p.r. since it is a purely syntactic operation on GN.

We now define the composite transformer $c : \text{Wff}_1 \rightarrow \text{Sent}$ by combining g^c with the result of the diagonalization function:

$$c := g^c \circ D^c = g^c \circ f^c \circ \Delta^c.$$

Thus, for each $a \in \text{Wff}_1$, c returns the code of the sentence asserting the unprovability of the diagonal instance of a :

$$c(a) = \ulcorner \neg \text{Prov}_T(\overline{D^c(a)}) \urcorner \in \text{Sent}.$$

To obtain an independent sentence by direct code computation, we apply c to the following formula:

$$E(x) := \forall y (\text{Diag}_T(x, y) \rightarrow \neg \text{Prov}_T(y)),$$

which, when instantiated with a code x , asserts that any y such that $\text{Diag}_T(x, y)$ holds is unprovable.

We fix the code $e := \ulcorner E(v_0) \urcorner \in \text{Wff}_1$, and isolate the sentence produced by diagonal evaluation of e :

$$G_{\text{wf}} := E(\bar{e}).$$

This sentence is the *witness* form of the independent sentence: it expresses, in terms of the graph predicate $\text{Diag}_T(x, y)$, that every value y obtained by diagonalizing the input code $\bar{e}$ is unprovable in T .

Let

$$d := D^c(e) = \text{Sub}(e, e) = \ulcorner E(\bar{e}) \urcorner = \ulcorner G_{\text{wf}} \urcorner \in \text{Sent}.$$

Thus d is the GN of the witness sentence itself.

The canonical *normal* form of the independent sentence results from applying the code-transformer c to the code e :

$$c(e) = (g^c \circ D^c)(e) = g^c(d) = \ulcorner \neg \text{Prov}_T(\bar{d}) \urcorner.$$

Therefore, $c(e)$ is the GN of the sentence

$$G_{\text{nf}} := \neg \text{Prov}_T(\bar{d}),$$

asserting the unprovability of the sentence with GN $d = \ulcorner E(\bar{e}) \urcorner$.

Intuitively, G_{nf} is obtained from G_{wf} by internally evaluating the diagonal graph: once T proves that $\text{Diag}_T(\bar{e}, y)$ picks out exactly $\bar{d}$, the witness form $\forall y (\text{Diag}_T(\bar{e}, y) \rightarrow \neg \text{Prov}_T(y))$ collapses to $\neg \text{Prov}_T(\bar{d})$.

This is what we prove in the next lemma, together with the fact that the normal form G_{nf} is T -provably equivalent to the witness form G_{wf} .

Lemma 1 (Normalization Lemma) *Let $d := D^c(e)$, so that $d = \ulcorner E(\bar{e}) \urcorner$. Then*

$$T \vdash G_{\text{wf}} \leftrightarrow G_{\text{nf}},$$

where $G_{\text{wf}} := E(\bar{e})$ and $G_{\text{nf}} := \neg \text{Prov}_T(\bar{d})$. In particular,

$$T \vdash G_{\text{wf}} \leftrightarrow \neg \text{Prov}_T(\overline{\neg G_{\text{wf}}}).$$

Proof By representability of $\text{Sub}(x, y) = z$ by $\text{Sub}_T(x, y, z)$, and since $d = D^c(e) = \text{Sub}(e, e)$, we obtain

$$T \vdash \text{Diag}_T(\bar{e}, \bar{d}),$$

hence

$$T \vdash \exists y. \text{Diag}_T(\bar{e}, y).$$

Since Sub_T is functional, the derived relation Diag_T is functional as well:

$$T \vdash \forall x \forall y \forall y'. ((\text{Diag}_T(x, y) \wedge \text{Diag}_T(x, y')) \rightarrow y = y').$$

Instantiating $x := \bar{e}$ yields

$$T \vdash \forall y \forall y'. ((\text{Diag}_T(\bar{e}, y) \wedge \text{Diag}_T(\bar{e}, y')) \rightarrow y = y').$$

Together with $T \vdash \exists y \text{Diag}_T(\bar{e}, y)$, this gives

$$T \vdash \exists! y \text{Diag}_T(\bar{e}, y).$$

We now show that $\text{Diag}_T(\bar{e}, y)$ picks out exactly $\bar{d}$.

First, from functionality and $T \vdash \text{Diag}_T(\bar{e}, \bar{d})$, we obtain

$$T \vdash \forall y. (\text{Diag}_T(\bar{e}, y) \rightarrow y = \bar{d}).$$

Conversely, from $y = \bar{d}$ and $T \vdash \text{Diag}_T(\bar{e}, \bar{d})$, by substitutivity of equality we infer $\text{Diag}_T(\bar{e}, y)$, hence

$$T \vdash \forall y. (y = \bar{d} \rightarrow \text{Diag}_T(\bar{e}, y)).$$

Therefore,

$$T \vdash \forall y. (\text{Diag}_T(\bar{e}, y) \leftrightarrow y = \bar{d}).$$

Now, unfolding $G_{\text{wf}} := E(\bar{e})$, we have

$$G_{\text{wf}} \equiv \forall y (\text{Diag}_T(\bar{e}, y) \rightarrow \neg \text{Prov}_T(y)).$$

Using $\text{Diag}_T(\bar{e}, y) \leftrightarrow y = \bar{d}$, we obtain

$$T \vdash G_{\text{wf}} \leftrightarrow \forall y. (y = \bar{d} \rightarrow \neg \text{Prov}_T(y)).$$

By instantiating $y := \bar{d}$ and using reflexivity $\bar{d} = \bar{d}$, we obtain

$$T \vdash \forall y. (y = \bar{d} \rightarrow \neg \text{Prov}_T(y)) \leftrightarrow \neg \text{Prov}_T(\bar{d}).$$

Hence,

$$T \vdash G_{\text{wf}} \leftrightarrow \neg \text{Prov}_T(\bar{d}).$$

By definition $G_{\text{nf}} := \neg \text{Prov}_T(\bar{d})$, this is

$$T \vdash G_{\text{wf}} \leftrightarrow G_{\text{nf}}.$$

Finally, since $\ulcorner G_{\text{wf}} \urcorner = d$, we obtain

$$T \vdash G_{\text{wf}} \leftrightarrow \neg \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner),$$

as claimed. $\square$

Thanks to this explicit composition of p.r. maps on GN, we both resolve the typing/well-definedness issue in Yanofsky's Lindenbaum-class presentation and separate two logically distinct steps: first, diagonalization produces a witness $G_{\text{wf}} = E(\bar{e})$, whose self-reference is mediated by the functional relation $\text{Diag}_T(x, y)$. A second, internal step normalizes this witness: because T proves $\text{Diag}_T(\bar{e}, y)$ has a unique output $y = \bar{d}$, the universal graph condition in $E(\bar{e})$ collapses to the canonical sentence $G_{\text{nf}} = \neg \text{Prov}_T(\bar{d}) = \neg \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner)$.

In this construction, the independent sentence emerges not by appealing to the fixed-point lemma as a black box, but by explicitly computing a p.r. pipeline on GN - first producing a witness form and then normalizing it inside T . In particular, this gives one response to Buldt's concern that fixed-point equivalences merely assert existence without "calculating" a sentence [5, p. 517]: here, on the other hand, the relevant codes - and thus a concrete representative of the Gödel sentence - are computed directly, and the equivalence between witness and canonical form is itself formalizable. In this way, the construction goes beyond the purely extensional content of a fixed-point equivalence: rather than a new proof of G1, it makes explicit the code-level mechanism that yields both the witness form and the normalized form of the independent sentence.

This process is illustrated by the following diagram:

$$\begin{array}{ccc} \text{Wff}_1 \times \text{Wff}_1 & \xrightarrow{f^c} & \text{Sent} \ni \ulcorner G_{\text{wf}} \urcorner \\ \Delta^c \uparrow & D^c \nearrow & \downarrow g^c \\ \text{Wff}_1 & \xrightarrow{c} & \text{Sent} \ni \ulcorner G_{\text{nf}} \urcorner \end{array}$$

3.3 Matrix representation

A matrix representation is natural in a code-based setting, since the p.r. substitution map f^c already fills a two-dimensional array of sentence codes, whose diagonal is then selected by D^c and transformed entrywise by g^c into the composite $c = g^c \circ D^c$.

f^c	a_1	a_2	$\cdots$	$a_k = e$	$\cdots$
a_1	$D^c(a_1)$	$Sub(a_2, a_1)$	$\cdots$	$Sub(a_k, a_1)$	$\cdots$
a_2	$Sub(a_1, a_2)$	$D^c(a_2)$	$\cdots$	$Sub(a_k, a_2)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$
$a_k = e$	$Sub(a_1, a_k)$	$Sub(a_2, a_k)$	$\cdots$	$D^c(e) = \ulcorner G_{\text{wf}} \urcorner$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$

Fig. 1 The substitution matrix: $M_{ij} = f^c(\langle a_i, a_j \rangle) = Sub(a_j, a_i)$. The diagonal consists of $D^c(a_i) = Sub(a_i, a_i)$

diagonal $D^c(a_i)$	flipped diagonal $g^c(D^c(a_i)) = c(a_i)$
$D^c(a_1)$	$g^c(D^c(a_1))$
$D^c(a_2)$	$g^c(D^c(a_2))$
$\vdots$	$\vdots$
$D^c(a_k) = D^c(e)$	$g^c(D^c(e)) = \ulcorner \neg \text{Prov}_T(\ulcorner E(\bar{e}) \urcorner) \urcorner = \ulcorner G_{\text{nf}} \urcorner$
$\vdots$	$\vdots$

Fig. 2 Applying g^c to the diagonal produces $c = g^c \circ D^c = g^c \circ f^c \circ \Delta^c$. The normal form of the Gödel sentence is the one coded by $c(e)$

To matrix-visualize this composite $c = g^c \circ f^c \circ \Delta^c$, fix an effective enumeration of Wff_1 as $a_1, a_2, a_3, \dots$. Define a 2-dimensional array by

$$M_{ij} := f^c(\langle a_i, a_j \rangle) = Sub(a_j, a_i).$$

The diagonal entries are $M_{ii} = D^c(a_i) = Sub(a_i, a_i)$.

Applying g^c to the diagonal yields the 1-dimensional “flipped diagonal” array

$$V_i := g^c(M_{ii}) = g^c(D^c(a_i)) = c(a_i).$$

Since $a_i \in \text{Wff}_1$, then $D^c(a_i) \in \text{Sent}$, so $g^c(D^c(a_i))$ is defined.

If $e = a_k$ in the enumeration, then the normalized form of the independent sentence is the sentence coded by $V_k = c(e)$ (See Fig. 1 and 2).

4 A categorical reading

The function-composition strategy developed in Section 3 admits a categorical reformulation. The section isolates the type-disciplined compositional skeleton of the construction in categorical terms. This allows one to compare how the building of the independent sentence as a concrete composite of p.r. code-transformations differs from other categorical readings of G1, and it prepares the ground for the “diagonal-flipping” blueprint of Section 6.

4.1 The category **PRCodes**

Fix a theory T satisfying the assumptions of Section 3, strong enough to represent p.r. functions and basic syntactic coding.

4.1.1 Objects.

An object of **PRCodes** is a p.r. predicate $\chi_A : \mathbb{N} \rightarrow \{0, 1\}$ deciding membership in set $|A| \subseteq \mathbb{N}$ such that $|A| = \{n : \chi_A(n) = 1\}$. Intuitively, objects are p.r. syntactic domains of GNs.

As seen in Section 3, the main objects are the code types:

$$\text{Wff}_1 = \{a \in \mathbb{N} : \chi_{\text{Wff}_1}(a) = 1\}, \quad \text{Sent} = \{a \in \mathbb{N} : \chi_{\text{Sent}}(a) = 1\},$$

each equipped with its p.r. membership test induced by the fixed Gödel coding, and working up to α -equivalence with distinguished free variable v_0 .

4.1.2 Morphisms.

A morphism $h : A \rightarrow B$ in **PRCodes** is a total p.r. function $h : \mathbb{N} \rightarrow \mathbb{N}$ such that $h[|A|] \subseteq |B|$, where two such functions h, l are the same morphism iff $\forall n \in |A|, h(n) = l(n)$.

We note that any morphism h has a representable graph in T (since it is p.r.), but **PRCodes** itself is defined at the meta-level: it is a category of code-transformations, not an internal category of T . Hence, although every morphism has a representable graph in T , the categorical structure lives externally; we are not formalizing category theory inside T .

4.1.3 Composition and identities.

Composition is given by ordinary function composition on representatives: if $h : A \rightarrow B$ and $l : B \rightarrow C$, then $l \circ h$ is represented by the total p.r. function $n \mapsto l(h(n))$. This is well-defined on equivalence classes of representatives: if $h \sim h'$ on $|A|$ and $l \sim l'$ on $|B|$, then for every $a \in |A|, h(a) = h'(a) \in |B|$, so $l(h(a)) = l'(h(a)) = l'(h'(a))$. Thus $l \circ h \sim l' \circ h'$ on $|A|$. The identity on A is represented by the ordinary identity function $\text{id}_A : \mathbb{N} \rightarrow \mathbb{N}$ with $\text{id}_A(n) = n$, which trivially maps $|A|$ into $|A|$.

4.2 Products and p.r. syntactic operations

To express duplication and the other maps categorically, we equip **PRCodes** with finite products. Fix a Gödel numbering $\ulcorner \cdot \urcorner$ for expressions of the language of T such that there exist p.r. functions (on $\mathbb{N}$) for the basic syntactic constructors listed below.

4.2.1 Pairing and projections.

Fix a p.r. pairing function $\langle \cdot, \cdot \rangle : \mathbb{N}^2 \rightarrow \mathbb{N}$ with p.r. projections $\pi_1, \pi_2 : \mathbb{N} \rightarrow \mathbb{N}$. Given objects A, B , define $A \times B$ by

$$|A \times B| = \{\langle a, b \rangle : a \in |A|, b \in |B|\},$$

with membership decided by $\chi_{A \times B}(p) = \chi_A(\pi_1(p)) \cdot \chi_B(\pi_2(p))$.

With projections π_1, π_2 and pairing $\langle \cdot, \cdot \rangle$, $A \times B$ behaves as a categorical product: every pair of arrows into A and B uniquely factors through $A \times B$ (where uniqueness is understood as equality of morphisms in **PRCodes**).

4.2.2 Terminal object and global elements.

Let 1 be the object with $|1| = \{0\}$, i.e. $\chi_1(n) = 1 \Leftrightarrow (n = 0)$. Then 1 is terminal in **PRCodes**: for all A there exists a unique morphism $A \rightarrow 1$, namely the constant function with value 0 .

A morphism $a : 1 \rightarrow A$ is called a global element of A and corresponds to the element $a(0) \in |A|$. In particular, the code $e = \ulcorner E(v_0) \urcorner \in \text{Wff}_1$ can be regarded as a global element $\hat{e} : 1 \rightarrow \text{Wff}_1$ by $\hat{e}(0) = e \in \text{Wff}_1$.

4.2.3 Code constructors for terms.

- $\text{Var}(i)$ is a p.r. function returning the code of the variable v_i as a term;
- $\text{num}(n)$ is a p.r. function returning the code of the numeral $\bar{n}$ as a term;
- $\text{PrfAtom}(u, v) : \mathbb{N}^2 \rightarrow \mathbb{N}$ returns the code of the atomic formula $\text{Prf}_T(t_u, t_v)$, where $u = \ulcorner t_u \urcorner$ and $v = \ulcorner t_v \urcorner$.

4.2.4 Formula-binding operators on codes.

- $\text{Neg}(\ulcorner \varphi \urcorner) = \ulcorner \neg \varphi \urcorner$,
- $\text{Ex}(v, \ulcorner \varphi \urcorner) = \ulcorner \exists v. \varphi \urcorner$ for a fixed variable code v .

4.2.5 Substitution.

We use the substitution function from Section 3: a p.r. function $\text{Sub} : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that when $b = \ulcorner B(v_0) \urcorner \in \text{Wff}_1$,

$$\text{Sub}(b, a) = \ulcorner B(\bar{a}) \urcorner \in \text{Sent}.$$

As mentioned, outside the intended domain we stipulate a default value to ensure totality, as in Section 3.

4.3 The diagonal-flipping pipeline as morphism composition

We now package the three key code-transformers from Section 3 as morphisms in **PRCodes**.

We use the code-level operations from Section 3: duplication $\Delta^c(a) = \langle a, a \rangle$, the substitution-on-pairs $f^c(p) = \text{Sub}(\pi_2(p), \pi_1(p))$ if $\pi_1(p), \pi_2(p) \in \text{Wff}_1$ and 0 otherwise, and the provability-flip code-builder $g^c(n) = \text{Neg}(\text{Ex}(v_m, \text{PrfAtom}(\text{Var}(m), \text{num}(n))))$, for a fixed variable index m coding the quantified proof variable.

Lemma 2 (Type-correctness in **PRCodes**) *The maps*

$$\Delta^c : \text{Wff}_1 \rightarrow \text{Wff}_1 \times \text{Wff}_1, \quad f^c : \text{Wff}_1 \times \text{Wff}_1 \rightarrow \text{Sent}, \quad g^c : \text{Sent} \rightarrow \text{Sent}$$

*are morphisms in **PRCodes** and hence compose.*

Proof P.r.-ness (and totality) of these maps is established by the closure of p.r. functions under composition, together with the syntactic constructors listed above. Domain preservation is by construction: if $a \in \text{Wff}_1$ then $\Delta^c(a) = \langle a, a \rangle \in \text{Wff}_1 \times \text{Wff}_1$; if $p = \langle a, b \rangle \in \text{Wff}_1 \times \text{Wff}_1$ then $b \in \text{Wff}_1$, hence $\text{Sub}(b, a) \in \text{Sent}$, so $f^c(p) \in \text{Sent}$; and if $n \in \text{Sent}$ then $g^c(n)$ is the code of a sentence, hence lies in Sent . Therefore Δ^c, f^c, g^c are morphisms in **PRCodes**. $\square$

Remark 1 (Factorization) By Lemma 2, the composites below are well-typed in **PRCodes**. Define the diagonal morphism

$$D^c := f^c \circ \Delta^c : \text{Wff}_1 \rightarrow \text{Sent}.$$

Then the overall transformer c factors as

$$c = g^c \circ f^c \circ \Delta^c = g^c \circ D^c : \text{Wff}_1 \rightarrow \text{Sent}.$$

Equivalently, the diagram

$$\text{Wff}_1 \xrightarrow{\Delta^c} \text{Wff}_1 \times \text{Wff}_1 \xrightarrow{f^c} \text{Sent} \xrightarrow{g^c} \text{Sent}$$

commutes with the induced arrows D^c and c .

4.4 Witness and normal form as categorical composites

We now state the two outputs of Section 3 - the witness independent sentence and its unprovability normal form - as composites in **PRCodes**.

Lemma 3 (Witness and normal form in **PRCodes**) *Let $\hat{e} : 1 \rightarrow \text{Wff}_1$ be the global element with $\hat{e}(0) = \ulcorner E(v_0) \urcorner$. Define the composites*

$$\gamma_{\text{wf}} := D^c \circ \hat{e} : 1 \rightarrow \text{Sent}$$

and

$$\gamma_{\text{nf}} := c \circ \hat{e} : 1 \rightarrow \text{Sent}.$$

By definition of $D^c(x) = \text{Sub}(x, x)$,

$$(D^c \circ \hat{e})(0) = D^c(e) = \text{Sub}(e, e) = \ulcorner E(\bar{e}) \urcorner.$$

Set

$$d := (D^c \circ \hat{e})(0) = \ulcorner E(\bar{e}) \urcorner.$$

Thus

$$\gamma_{\text{wf}}(0) = d = \ulcorner G_{\text{wf}} \urcorner.$$

and

$$\gamma_{\text{nf}}(0) = \ulcorner \neg \text{Prov}_T(\bar{d}) \urcorner = \ulcorner G_{\text{nf}} \urcorner.$$

Moreover,

$$T \vdash G_{\text{wf}} \leftrightarrow G_{\text{nf}},$$

and in particular

$$T \vdash G_{\text{wf}} \leftrightarrow \neg \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner).$$

Proof Let $E(x)$ be the one-free-variable formula from Section 3 and let $e = \ulcorner E(v_0) \urcorner \in \text{Wff}_1$. Consider e as a global element $\hat{e} : 1 \rightarrow \text{Wff}_1$ by $\hat{e}(0) = e$. By definition of $D^c(x) = \text{Sub}(x, x)$,

$$(D^c \circ \hat{e})(0) = D^c(e) = \text{Sub}(e, e) = \ulcorner E(\bar{e}) \urcorner,$$

so γ_{wf} codes the witness sentence $E(\bar{e})$.

Using Remark 1, we have $c = g^c \circ D^c$, hence

$$(\gamma_{\text{nf}})(0) = (c \circ \hat{e})(0) = (g^c \circ D^c \circ \hat{e})(0) = g^c(\ulcorner E(\bar{e}) \urcorner) = \ulcorner \neg \text{Prov}_T(\ulcorner E(\bar{e}) \urcorner) \urcorner,$$

so γ_{nf} codes the unprovability normal form.

Finally, the equivalence $T \vdash G_{\text{wf}} \leftrightarrow G_{\text{nf}}$ is exactly the Normalization Lemma proved in Section 3. Thus the categorical composites $D^c \circ \hat{e}$ and $g^c \circ D^c \circ \hat{e}$ yield codes of sentences that are provably equivalent in T . Hence, the categorical diagram captures the compositional structure of the arithmetical construction. $\square$

4.5 Comparison with Lawvere's and Roberts' categorical approaches

The categorical reformulation in this Section is deliberately not presented as a new “categorical proof” of G1, but, rather it isolates the compositional skeleton already implicit in the p.r. code pipeline of Section 3. It is therefore useful to position **PRCodes** relative to two influential categorical accounts of diagonalization: on one hand, Lawvere's cartesian closed formulation of diagonal arguments [22]; on the other hand, Roberts' subsequent substructural weakening of Lawvere's framework [34].

Lawvere's fixed-point theorem is formulated in the setting of cartesian closed categories (CCCs), and its contrapositive yields a universal diagonal argument [22,

Thm. 1.1]. Lawvere also remarks that the argument can work with finite products alone [22, p. 6]; in fact, the applications to Gödel and Tarski are obtained by passing to a Lindenbaum-style category with finite products, where morphisms are provability-equivalence classes of terms and formulas and composition is induced by substitution [22, Section 3]. In that setting, the basic semantic primitives are represented by equivalence classes, e.g. a negation endomorphism $\text{not} : 2 \rightarrow 2$ and a substitution operation $\text{subst} : A \times A \rightarrow A$ [22, p. 10–12]. In particular, Lawvere’s discussion of incompleteness is presentation-free: it emphasizes representability conditions for satisfaction, truth, and provability [22, Cor. 3.1, Thm. 3.2, Thm. 3.3].²

By contrast, the category **PRCodes** is not a Lindenbaum quotient: objects are p.r. code domains (e.g. Wff_1 , Sent), and morphisms are total p.r. functions on GNs preserving those domains. As mentioned in Section 3, the point of this design choice is to avoid a typing problem that arises when one treats substitution-like operations on provability equivalence classes. Hence, **PRCodes** makes the coding layer the primary level of discourse.

Moreover, Lawvere’s fixed-point theorem isolates diagonalization as the categorical diagonal $\delta_A : A \rightarrow A \times A$ together with evaluation in the exponential Y^A and a weak point-surjectivity hypothesis $g : A \rightarrow Y^A$ [22, p. 5]. Informally, diagonalization comes from feeding an index into the family it indexes (via the diagonal). On the other hand, in **PRCodes**, diagonalization is not an ambient categorical principle but a specific morphism D^c induced by p.r. syntactic coding, representable in T by a wff, and factoring as $D^c = f^c \circ \Delta^c$ (Remark 1). Thus, while Lawvere treats diagonalization as a structural map present in any cartesian context, our setting treats diagonalization as a computable syntactic operation on codes whose correctness is tracked by representability inside T .

Also, Lawvere’s diagonal argument requires a nontrivial endomorphism $t : Y \rightarrow Y$ (e.g. negation on 2), and the fixed-point theorem shows that under the weak surjectivity hypothesis one can construct a point fixed by t [22, p. 5–6]. In the logical applications, the “flip” is supplied by a fixed-point-free endomorphism such as classical negation (or an analogous operation arising from consistency) [22, p. 9]. In **PRCodes**, the flip is again syntactic and explicit as the p.r. code-builder g^c whose effect is to “provability-negate” its input sentence code. Hence, the construction does not proceed by assuming a point-surjective map and deriving a fixed point for arbitrary endomorphisms; instead, it computes a *particular* self-referential sentence by unwinding the p.r. pipeline.

Following this, a central output of Section 3 is the separation between a witness sentence G_{wf} and its normalized unprovability form G_{nf} : this witness-vs-normal-form is a computational distinction which records whether diagonalization has been left as a graph condition inside the sentence or has been evaluated to the unique diagonal output. This “internal normalization step” has no direct analogue in Lawvere’s presentation, since his goal is to exhibit fixed-point phenomena as consequences of abstract principles.

Concerning Roberts’ paper: it re-examines Lawvere’s theorem and weakens its hypotheses by replacing the cartesian setting by a pointed magmoidal category with

² Lawvere, in Section 3, does introduce an auxiliary “metamathematical” relation $\Gamma \subseteq \mathbf{C}(1, A) \times \mathbf{C}(1, 2)$, which functions as a minimal trace of Gödel-style coding, but later it is observed to be superfluous [22, Appendix].

diagonals $(\mathcal{C}, \#, \delta, t)$ [34, Def. 3] - where $\#$ is a binary tensor-like operation and δ supplies “diagonals” in the sense of a natural transformation $\delta : \text{id}_{\mathcal{C}} \Rightarrow \# \circ \Delta_{\mathcal{C}}$, while the role of global elements is generalized to t -points $t \rightarrow A$ [34, Sections 2 and 4]. In this general setting Roberts proves diagonal arguments and fixed-point theorems under hypotheses tailored to substructural logics (e.g. using t -free endomorphisms and “incomplete parametrization”) [34, Def. 10, Thm. 11].

Our categorical reading seems to go in the opposite direction: rather than minimizing the structural axioms needed for diagonalization, we maximize syntactic control by choosing a concrete category of p.r. code-transformers. In fact, **PRCodes** can be seen as a cartesian instance of Roberts’ framework if one takes $\# = \times, t = 1$, and $\delta_A(a) = \langle a, a \rangle$, where $\times$ is the p.r. implemented pairing product and δ is the diagonal natural transformation induced by duplication. From this perspective, Roberts’ emphasis on the locality of the argument resonates with our code-based construction which is driven by a particular global element $\hat{e} : 1 \rightarrow \text{Wff}_1$ and the specific syntactic maps Δ^c, f^c, g^c .

However, the aims differ. Roberts is concerned with the scope of diagonal methods across substructural settings and shows that diagonalization survives under strong weakening of classical structure. We instead remain in an explicitly arithmetic environment where the relevant operations are required to be p.r. and representable in T ; this forces a computability discipline that is orthogonal to Roberts’ structural weakening program.

5 Variants

5.1 Rosser’s “trick”

To obtain Rosser’s strengthening [35], we keep the same diagonal map $D^c(x) = \text{Sub}(x, x)$ from Section 3, and we only replace the provability-flip code-builder g^c by its Rosser analogue g^R .

We reuse the p.r. negation constructor $\text{Neg} : \text{Sent} \rightarrow \text{Sent}$ from Section 4, and we define the p.r. Rosser proof relation:

$$\text{Prf}_T^R(u, x) := \text{Prf}_T(u, x) \wedge \forall z < u. \neg \text{Prf}_T(z, \text{Neg}(x)).$$

Since the second conjunct is bounded, Prf_T^R is p.r. whenever Prf_T is, hence representable in T by a wff $\text{Prf}_T^R(u, x)$. We set

$$\text{Prov}_T^R(x) := \exists u. \text{Prf}_T^R(u, x),$$

so Prov_T^R is Σ_1^0 .

On this, we define the p.r. code-builder

$$g^R(n) := \ulcorner \neg \text{Prov}_T^R(\bar{n}) \urcorner, \quad g^R : \text{Sent} \rightarrow \text{Sent},$$

together with the composite code-transformer

$$c^R := g^R \circ D^c : \text{Wff}_1 \rightarrow \text{Sent}.$$

Thus $c^R(a)$ returns the code of the sentence asserting that the diagonalization of a is not Rosser-provable.

Then we proceed analogously to Section 3: we define the one-free-variable formula

$$E^R(x) := \forall y. (\text{Diag}_T(x, y) \rightarrow \neg \text{Prov}_T^R(y)),$$

and let $e^R := \ulcorner E^R(v_0) \urcorner \in \text{Wff}_1$ and $d^R := D^c(e^R) = \ulcorner E^R(\overline{e^R}) \urcorner$. Then the witness form and its Rosser normal form are

$$G_{\text{wf}}^R := E^R(\overline{e^R}), \quad G_{\text{nf}}^R := \neg \text{Prov}_T^R(\overline{d^R}).$$

The same internal normalization argument as Lemma 1 yields

$$T \vdash G_{\text{wf}}^R \leftrightarrow G_{\text{nf}}^R,$$

since Diag_T is functional in T as a consequence of the functionality of Sub_T .

5.2 Alternative matrix representation: One matrix

It is possible to improve the previous two-matrix representation (Subsection 3.3) by collapsing the substitution matrix and the flipped-diagonal vector into a single matrix whose entries already incorporate the provability-flip.

As before, fix an effective enumeration $(a_i)_{i \in \mathbb{N}}$ of Wff_1 and set

$$N_{ij} := (g^c \circ f^c)(\langle a_i, a_j \rangle) = g^c(\text{Sub}(a_j, a_i)) \in \text{Sent}.$$

The diagonal entries are

$$N_{ii} = g^c(D^c(a_i)) = c(a_i).$$

In particular, if $e = a_k$, then the (k, k) -entry is

$$N_{kk} = c(e) = \ulcorner G_{\text{nf}}^R \urcorner,$$

the code of the normal form of the independent sentence. For the Rosser variant one replaces g^c by g^R throughout (See Fig. 3).

5.3 A hypothetical (0,1)-matrix for a computable decider

We present a final matrix visualization that shows the impossibility of a total computable negation-completeness decider for T . This is not a morphism on **PRCodes**, but a *reductio* assumption. Assume that T is 1-consistent and negation complete and

$g^c \circ f^c$	a_1	a_2	$\dots$	$a_k = e = \ulcorner E(v_0) \urcorner$	$\dots$
a_1	$g^c(f^c(\langle a_1, a_1 \rangle))$			$g^c(f^c(\langle a_1, e \rangle))$	$\dots$
a_2		$g^c(f^c(\langle a_2, a_2 \rangle))$	$\dots$	$g^c(f^c(\langle a_2, e \rangle))$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\dots$
$a_k = e$				$g^c(f^c(\langle e, e \rangle))$ $= c(e) = \ulcorner G_{\text{nf}} \urcorner$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Fig. 3 Single-matrix visualization: $N_{ij} = g^c(f^c(\langle a_i, a_j \rangle))$. The diagonal is $N_{ii} = g^c(D^c(a_i)) = c(a_i)$. The highlighted column corresponds to $e = \ulcorner E(v_0) \urcorner$

that there exists a total computable function $s_{\pm} : \text{Sent} \rightarrow \{0, 1\}$ such that for every sentence code $n \in \text{Sent}$, writing φ_n for the sentence with GN n ,

$$s_{\pm}(n) = \begin{cases} 0 & \text{if } T \vdash \varphi_n, \\ 1 & \text{if } T \vdash \neg\varphi_n. \end{cases}$$

Totality of $s_{\pm}$ follows from the assumption that T is p.r.-axiomatized and negation complete.

As in Section 3, we fix an effective enumeration of Wff_1 as $a_1, a_2, \dots$, and define the substitution matrix $M_{ij} := f^c(\langle a_i, a_j \rangle) = \text{Sub}(a_j, a_i) \in \text{Sent}$. Applying $s_{\pm}$ entrywise yields a $(0, 1)$ -matrix

$$B_{ij} := s_{\pm}(M_{ij}) \in \{0, 1\}.$$

Now we let $e = \ulcorner E(v_0) \urcorner \in \text{Wff}_1$ be the specific code from Section 3, and k be its index in the enumeration, i.e. $a_k = e$. Then the diagonal entry at (k, k) is

$$M_{kk} = D^c(e) = \ulcorner E(\bar{e}) \urcorner = \ulcorner G_{\text{wf}} \urcorner,$$

and hence

$$B_{kk} = s_{\pm}(M_{kk}) = s_{\pm}(\ulcorner G_{\text{wf}} \urcorner) \in \{0, 1\}.$$

Since $s_{\pm}$ is assumed to be total, the entry at $B_{k,k}$ must be either 0 or 1. However, both options yield a contradiction.

If $B_{k,k} = 0$, then $T \vdash G_{\text{wf}}$, and hence $T \vdash \neg \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner)$ by the Normalization Lemma (Section 3). But by assuming the first derivability condition (or, alternatively, since $T \vdash \varphi$, then φ has some code p of a T -proof; hence $T \vdash \text{Prf}_T(\bar{p}, \ulcorner \varphi \urcorner)$, so $T \vdash \text{Prov}_T(\ulcorner \varphi \urcorner)$) from $T \vdash G_{\text{wf}}$ we have $T \vdash \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner)$, contradicting consistency.

If $B_{k,k} = 1$, then $T \vdash \neg G_{\text{wf}}$; by the Normalization Lemma, $T \vdash \neg G_{\text{wf}} \rightarrow \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner)$, hence $T \vdash \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner)$. Since Prov_T is Σ_1 , under 1-consistency

this implies that there exists an actual proof for G_{wf} , contradicting $T \vdash \neg G_{wf}$. For the argument to rely on mere consistency of T , it is enough to replace Prov_T with Rosser's provability predicate Prov_T^R .

We can summarize the contradiction by introducing a Cantor-like boolean diagonal-flip map

$$\text{flip} : \{0, 1\} \rightarrow \{0, 1\}, \quad \text{flip}(b) = 1 - b.$$

Total two-valued classification would then force the diagonal entry B_{kk} to disagree with itself, in the sense that it would have to satisfy

$$B_{kk} = \text{flip}(B_{kk}),$$

which is impossible in $\{0, 1\}$. Hence no such total computable two-valued classifier $s_{\pm}$ can exist, and the matrix B cannot be total in the sense required by negation-completeness.

6 A type-disciplined blueprint for diagonal arguments

Sections 3–4 exhibit the canonical normal form G_{nf} of the independent sentence as the output of an explicit p.r. code-transformer

$$c = g^c \circ D^c = g^c \circ f^c \circ \Delta^c : \text{Wff}_1 \rightarrow \text{Sent},$$

obtained by composing duplication, diagonal substitution, and a provability-flip code builder.

This explicit compositional structure suggests a broader application. As noted by [9, 42] among others, many classical diagonal arguments - e.g. Cantor, Russell, Turing, Grelling–Nelson - share the same combinatorial structure. The present section abstracts this shared structure into a general blueprint that matches the code-transformer pipeline presented so far. The blueprint does not aim to replace Gödel's arithmetical argument, nor does it aim to collapse or reduce Gödel's argument to Cantor's or Russell's. Rather, the blueprint isolates the compositional skeleton common to these diagonal arguments once one distinguishes the relevant types/levels (objects vs. codes vs. classifiers). This type analysis will also clarify why some instances of diagonal-flipping yield paradoxes (Russell, Grelling–Nelson) while others yield theorem-like limitations (Cantor, Turing, Gödel).

6.1 Known analogies with classical arguments

We briefly recall four canonical diagonal arguments - Cantor's [6], Russell's [36], Grelling–Nelson's [12], and Turing's [40] - emphasizing the same matrix-building and diagonal-flip ideas.

6.1.1 Cantor's theorem.

Cantor's theorem states that no function $f : K \rightarrow \mathcal{P}(K)$ is surjective. The proof defines

$$D = \{x \in K : x \notin f(x)\}.$$

If $f(z) = D$ for some $z \in K$, then asking whether $z \in D$ yields a contradiction. To visualize the argument in a matrix form (in the countable case), fix an enumeration $K = \{x_0, x_1, \dots\}$ and interpret each row of the matrix as the subset $f(x_i) = S_i \subseteq K$ and each column as an element $x_j \in K$. Define

$$a_{i,j} = \begin{cases} 1 & \text{if } x_j \in f(x_i), \\ 0 & \text{if } x_j \notin f(x_i). \end{cases}$$

The set D corresponds to the row obtained by flipping the diagonal entries: $a_{i,i} \mapsto 1 - a_{i,i}$. This process guarantees $D \neq f(x_i)$ for all i , hence f is not surjective.

6.1.2 Russell's paradox.

Russell's paradox [36] can be interpreted as a diagonal-flipping construction performed in a naive (i.e. unstratified) membership universe. Consider a matrix whose rows and columns range over the same collection of sets $\{x_i\}$, and define

$$a_{i,j} = \begin{cases} 1 & \text{if } x_j \in x_i, \\ 0 & \text{if } x_j \notin x_i. \end{cases}$$

The diagonal entries record whether a set contains itself. Define

$$S = \{x : x \notin x\}.$$

The entries of the row for S correspond to flipping the diagonal entries. The contradiction arises when one attempts to determine whether $S \in S$, i.e. what is the diagonal entry indexed by S .

6.1.3 Grelling–Nelson's paradox.

Let T be the set of adjective tokens and assume enumeration of tokens $T = \{t_0, t_1, \dots\}$. Define an "application" relation $Ap : T \times T \rightarrow \{0, 1\}$, where $Ap(x, y) = 1$ is read as " x applies to y " (or " y falls under x "). This forms a matrix

$$a_{i,j} = \begin{cases} 1 & \text{if } Ap(t_i, t_j) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The diagonal entries record whether a term applies to itself (autological).³ Define the anti-diagonal predicate (“heterological”) by

$$H(x) := \neg Ap(x, x).$$

The row determined by H is obtained by flipping the diagonal entries. Then, the paradoxical question is whether there is a token $h \in T$ corresponding to the adjective “heterological” for which $H(h)$ holds, i.e., which is itself heterological. Alternatively, letting $h = t_k$ in the enumeration, the paradox arises from evaluating the diagonal condition $H(h) = \neg Ap(h, h)$ (equivalently, $\neg a_{k,k}$ since $a_{k,k} = Ap(t_k, t_k) = Ap(h, h)$), which forces the contradictory requirement $Ap(h, h) \leftrightarrow \neg Ap(h, h)$.

6.1.4 Turing’s undecidability of the halting problem.

The proof can be outlined in two steps [39]. First, let $\{TM_i\}_{i \in \mathbb{N}}$ be an effective enumeration of all Turing machines and let $\{\langle TM_i \rangle\}_{i \in \mathbb{N}}$ be an effective enumeration of their encodings. Assume, per *reductio*, that there is an evaluator Turing machine TM_E which decides the halting problem: it correctly determines whether TM_j halts on input $\langle TM_i \rangle$. The behavior of the evaluator TM_E can be captured by a $(0, 1)$ -matrix M :

$$M(i, j) = \begin{cases} 1, & \text{if } TM_j \text{ halts on input } \langle TM_i \rangle, \\ 0, & \text{otherwise.} \end{cases}$$

The diagonal entries $M(i, i)$ record whether TM_i halts on its own encoding.

Second, define the “diagonal-flipping” machine TM_d as follows. On input $\langle TM_i \rangle$, TM_d consults TM_E about the diagonal entry $M(i, i)$ and then flips the resulting halting behavior

$$TM_d(\langle TM_i \rangle) := \begin{cases} \text{loop,} & \text{if } M(i, i) = 1, \\ \text{halt,} & \text{if } M(i, i) = 0. \end{cases}$$

Finally, evaluate TM_d on its own encoding $\langle TM_d \rangle$. Write d for the index of TM_d in the enumeration. Then:

- If $M(d, d) = 1$, then TM_d must diverge on $\langle TM_d \rangle$, contradicting $M(d, d) = 1$.
- If $M(d, d) = 0$, then TM_d must halt on $\langle TM_d \rangle$, contradicting $M(d, d) = 0$.

Thus the assumption that TM_E correctly decides all entries of M leads to contradiction. Hence, no such evaluator TM_E exists.

The common pattern is: (i) represent a system of objects together with an associated interaction map, (ii) form the self-application condition by restricting this map to the diagonal, and (iii) negate (flip) this diagonal condition to define a new object that cannot be captured by the original scheme. Enumerations and $(0, 1)$ -matrices provide a useful visualization of this mechanism in many cases, but are not essential. The next subsections make this schema explicit in a form specialized to the code-level pipeline used in this paper.

³ There are multiple criteria for “applies to itself”—e.g. based on form or spelling (syntactic self-application, like “pentasyllabic”), on usage (pragmatic self-application, like the adjective “common”), or on meaning (semantic self-application, like “noun”). These distinctions are not considered here.

6.2 Methodological steps

In light of the similarities - and differences - between the arguments just recalled (including Gödel's construction), we can isolate a recurring *diagonal-flipping* blueprint, which can be instantiated in set theory, computability theory, and arithmetic. The key aspect is the distinction between the diagonal flip process and the *representability* of the output of this process by an element of the domain. The blueprint can be formulated at a high level (independent of any specific background theory) as follows:

6.2.1 Step 1: Domains.

Fix two domains X and Y . Depending on the context, X and Y may coincide or be distinct. Also, X , Y can play the role of resp. row objects and column objects in matrix contexts.

- Distinct domains:
 - Cantor: $X = \mathcal{P}(K)$, $Y = K$ a base set.
 - Turing: $X = \{TM\}$ and $Y = \{\langle TM \rangle\}$ (i.e., machines vs. codes).
- Same domain:
 - Russell: $X = Y$ ranges over “all sets” in naive set theory.
 - Grelling–Nelson: $X = Y$ ranges over terms in a given natural language.
 - Gödel: $X = Y$ ranges over one-free-variable formulas (or their codes) in a sufficiently strong arithmetic theory.

When convenient (e.g. for matrix presentations), assume enumerations for these two collections (effective when the context requires/allows it) $e_X : \mathbb{N} \rightarrow X$ and $e_Y : \mathbb{N} \rightarrow Y$, written $x_i := e_X(i)$ and $y_j := e_Y(j)$.

6.2.2 Step 2: Interaction map and diagonal operator.

Fix an interaction map

$$\Phi : X \times Y \rightarrow Z.$$

Typical choices of Z include booleans (e.g. for membership/denoting/halting as in Cantor's, Russell's, Grelling–Nelson's, and Turing's) or a syntactic code domain (e.g. $Z \subseteq \mathbb{N}$ for codes of formulas, as in Gödel's).

Together with the chosen enumerations $(x_i)_{i \in \mathbb{N}}$ of X and $(y_j)_{j \in \mathbb{N}}$ of Y , Φ can be visualized as the matrix-filling map determining the entries

$$M(i, j) := \Phi(x_i, y_j) \in Z,$$

where each entry $M(i, j)$ records how the i -th row object interacts with the j -th column object via Φ .

The diagonal entries

$$M(i, i) = \Phi(x_i, y_i)$$

encode the “self” instances permitted by the setup, i.e., the interaction of an object with itself - e.g. membership in itself in Russell, a machine run on its own code in Turing, a formula instantiated with its own GN in Gödel, etc.

The diagonal behavior can be captured by a diagonal operator in two ways. First, relative to the chosen enumerations, one has the enumeration-diagonal map

$$D^e : \mathbb{N} \rightarrow Z, \quad D^e(i) := \Phi(x_i, y_i).$$

In contexts where $X = Y$ and the same enumeration is used for both rows and columns, this is the familiar $D^e(i) = M(i, i)$.

Alternatively, when the framework supplies a canonical self-input map

$$s : X \rightarrow Y,$$

one can define an object-level diagonalization map

$$D^m : X \rightarrow Z, \quad D^m(x) := \Phi(x, s(x)).$$

In the special case $X = Y$ one may take $s = \text{id}$ and recover the raw diagonal instance $D^m(x) = \Phi(x, x)$. The role of s is deepened in Step 5.

6.2.3 Step 3: Diagonal flipping.

Next, a diagonal-flip operation (typically some form of negation or complement, e.g. the provability-negating constructor) is applied to the diagonal outputs $D^e(i)$. Fix a map

$$\text{flip} : Z \rightarrow Z$$

which transforms diagonal entries into their “complements” in the relevant sense. In boolean settings, where $Z = \{0, 1\}$, flip is boolean negation $\text{flip}(z) = 1 - z$. In Gödel’s setting (as in this paper), Z is a space of sentence-codes and flip is g^c producing an unprovability statement.

Applying flip to the diagonal outputs yields the diagonal-flipped sequence

$$V : \mathbb{N} \rightarrow Z \quad V(i) := \text{flip}(D^e(i)) = \text{flip}(\Phi(x_i, y_i)) \quad (i \in \mathbb{N}).$$

Thus $V(i)$ records, at each index i , the diagonal values $\Phi(x_i, y_i)$ transformed by the relevant complement/negation operation flip . Equivalently, $V(i)$ is obtained by flipping the diagonal entry of the i -th row function at its i -th column argument.

In many settings, one can package the same information as an *anti-diagonal row function* on column objects. Concretely, when the diagonal flip is definable uniformly over Y , i.e. as a function on all column objects and not merely along the enumerated diagonal, we can fix a function $\delta_{\text{anti}} : Y \rightarrow Z$ such that

$$\delta_{\text{anti}}(y_i) = V(i) \quad (i \in \mathbb{N}).$$

In boolean settings, δ_{anti} is an anti-diagonal predicate on Y ; in syntactic settings, δ_{anti} is a Z -valued anti-diagonal operation on the column objects. In either case, δ_{anti} represents the *externally* (i.e., metatheoretically) defined row function obtained by flipping the diagonal pattern determined by Φ relative to the chosen enumerations.

6.2.4 Step 4: Representability of the anti-diagonal behavior.

The next step is to see whether the externally defined anti-diagonal row function constructed in Step 3 is *internally realized* by an element of the row domain X .

In fact, the matrix map $\Phi : X \times Y \rightarrow Z$ induces, by currying, a canonical row-function assignment

$$R : X \rightarrow (Y \rightarrow Z), \quad R(x)(y) := \Phi(x, y).$$

Hence each row object $x \in X$ determines a row function $R(x) : Y \rightarrow Z$ describing how x interacts with the column objects.

In light of this, we can distinguish three notions of internal representability of the anti-diagonal row function.

The first is *global* representability: if $\delta_{\text{anti}} : Y \rightarrow Z$ is available as a genuine function on Y , the representability question is whether

$$\exists x^* \in X. \forall y \in Y. R(x^*)(y) = \delta_{\text{anti}}(y).$$

This means that there is a single row object x^* whose induced row function agrees with the anti-diagonal function δ_{anti} on all inputs $y \in Y$. In this case, δ_{anti} itself may already be an element of X (e.g. in Cantor-style settings where X is a space of predicates on Y).

The second is a *weaker* internal representability: it only requires representability along the chosen enumeration:

$$\exists x^* \in X. \forall i \in \mathbb{N}. R(x^*)(y_i) = V(i),$$

i.e.

$$\exists x^* \in X. \forall i \in \mathbb{N}. \Phi(x^*, y_i) = V(i) = \text{flip}(\Phi(x_i, y_i)),$$

This means that there is a row object x^* whose interaction with the column objects matches the externally specified flipped-diagonal data only on the particular sequence of inputs $y_0, y_1, \dots$, without requiring that x^* agree with a globally defined anti-diagonal function on all elements of Y . Thus, this second notion of representability is weaker than global representability, and it coincides with the latter under additional assumptions, e.g. when $e_Y : \mathbb{N} \rightarrow Y$ is surjective.

The third mode of representability is *local*: instead of requiring a single $x^* \in X$ to realize the flipped diagonal data either globally or just along the entire enumeration, one targets only a *single chosen input* and internalizes the diagonal flip at that point. As we will see in a moment, Gödel's construction proceeds precisely in this local way.

When such a realizer x^* exists (in either sense), we say that it *represents* the diagonal flip (relative to the chosen enumerations). This formulation separates two notions: the *externally defined* anti-diagonal data (either as a global row function $\delta_{\text{anti}} : Y \rightarrow Z$, when available, or as the diagonal-flipped sequence $V : \mathbb{N} \rightarrow Z$ relative to an enumeration), and its *internal* realizability by some $x^* \in X$. In some settings (e.g. Cantor), representability is immediate because the anti-diagonal row function is itself an element of X .

6.2.5 Step 5: Diagonal self-evaluation.

When a realizer $x^* \in X$ exists as in Step 4, the diagonal-flip construction yields a contradiction only once one feeds the realizer back into the diagonal. Concretely, one must evaluate the interaction map Φ at a column-input canonically associated with x^* . This requires some *diagonal-access* or *self-input* operation

$$s : X \rightarrow Y,$$

so that the critical instance becomes

$$\Phi(x^*, s(x^*)) \text{ and its flip } \text{flip}(\Phi(x^*, s(x^*))).$$

As we will see in the next Subsection 6.3, the availability of this self-input operation for the realizer depends on the stratification between the domains involved. In non-stratified settings, where $X = Y$ and self-input is immediate, one may take $s = \text{id}_X$, and the self-evaluation $\Phi(x^*, x^*)$ is well-formed. In that case the diagonal-flip can force a contradiction *internally* in the system (as in Russell and Grelling–Nelson), so the inconsistency targets the ambient naive framework itself.

By contrast, in stratified settings the term $\Phi(x^*, x^*)$ is ill-typed: x^* lives in the row domain X but cannot be supplied as a column argument since the column domain is of different type. Hence, diagonal self-evaluation is not automatic and becomes possible only via additional structure supplying s : this can be e.g. a surjectivity assumption (as in Cantor), a coding map together with an evaluator (Turing), or a coding map together with an internal provability representation (Gödel).

Consequently, any contradiction obtained via the diagonal evaluation of the realizer targets such extra structure and/or additional closure principles associated with the framework (e.g. negation completeness in Gödel), which is then refuted by the diagonal argument. This is precisely what yields a limitation theorem: no surjection in Cantor, no decider in Turing, and no negation completeness under standard consistency assumptions in Gödel.

6.2.6 Gödel's case as an explicit code-level instance.

We now show how the previous abstract blueprint specializes to the concrete construction presented in Section 3.

Step 1. The abstract data are identified as $X = Y = \text{Wff}_1 \subseteq \mathbb{N}$, i.e. the set of codes of one-free-variable formulas (class signs). Then, fix an enumeration $(a_i)_{i \in \mathbb{N}}$ of Wff_1 .

Step 2. The interaction map $\Phi(a, b)$ corresponds to $\text{Sub}(b, a)$, and the codomain is $Z = \text{Sent} \subseteq \mathbb{N}$, the set of sentence codes. The associated substitution matrix is

$$M_{ij} := \Phi(a_i, a_j) = \text{Sub}(a_j, a_i) \in \text{Sent}.$$

Its diagonal is

$$M_{ii} = \text{Sub}(a_i, a_i) = D^c(a_i),$$

where $D^c(a) = \text{Sub}(a, a)$ is the code-level diagonal map.

Step 3. In Gödel's case, the flipping map is not boolean negation but the p.r. code-builder

$$g^c : \text{Sent} \rightarrow \text{Sent}, \quad g^c(n) = \ulcorner \neg \text{Prov}_T(\bar{n}) \urcorner,$$

which maps a sentence-code to the code of the sentence asserting its unprovability. Applying g^c to the diagonal yields the flipped diagonal vector

$$V_i := g^c(M_{ii}) = g^c(D^c(a_i)) = (g^c \circ D^c)(a_i) = c(a_i),$$

where $c = g^c \circ D^c$ is the composite transformer of Section 3.

Step 4. The internalization step proceeds via representability of the diagonal evaluation. First, D^c is represented in T by the graph formula $\text{Diag}_T(x, y)$. Next, let $E(x)$ be the formula introduced in Section 3, and let $e := \ulcorner E(v_0) \urcorner \in \text{Wff}_1$. Define the diagonal output code

$$d := D^c(e) = \text{Sub}(e, e) = \ulcorner E(\bar{e}) \urcorner \in \text{Sent}.$$

This is the code of a sentence that expresses that every y satisfying $\text{Diag}_T(\bar{e}, y)$ is unprovable. Sections 3 and 4 identified this sentence as the witness sentence:

$$G_{\text{wf}} := E(\bar{e}), \quad \text{with} \quad \ulcorner G_{\text{wf}} \urcorner = d.$$

Step 5. The diagonal self-evaluation of the realizer comes via the application of the flip map g^c to the diagonal output d .

This operation yields the code of a sentence that internalizes the unprovability of the sentence with code d : this sentence corresponds to the normalized self-targeting instance obtained once the evaluation of the graph formula $\text{Diag}_T(\bar{e}, y)$ is internally resolved. In other words, this sentence is the evaluation of the witness sentence at its own GN mediated through the code boundary, and not as a mere self-evaluation as a syntactic object.

Sections 3 and 4 identified this sentence as the canonical “anti-diagonal” sentence

$$G_{\text{nf}} := \neg \text{Prov}_T(\bar{d}) = \neg \text{Prov}_T(\ulcorner G_{\text{wf}} \urcorner),$$

whose code is $g^c(d) = c(e)$.

Then, by the Normalization Lemma,

$$T \vdash G_{\text{wf}} \leftrightarrow G_{\text{nf}}.$$

Together with standard metatheoretic assumptions (e.g. consistency for unprovability, and ω -consistency or Σ_1 -soundness for unrefutability), this yields that G_{nf} is independent of T . Hence any additional closure property such as negation completeness is refuted. In this case, the machinery used for Steps 4 and 5 are the arithmetization of syntax and the internal representation of provability: these are parts of the background setup, and hence they are not the items that are touched by the contradiction. What is refuted is the extra closure property of negation completeness.

In Gödel's case, the compositional code-operation pipeline clarifies the passage from Step 4 to Step 5. The diagonal-flipping construction is performed externally on codes via D^c and g^c , and it is internalized in T by representing the diagonal map via Diag_T and proving that $\text{Diag}_T(\bar{e}, y)$ picks out a unique output $y = \bar{d}$ (Step 4). This leads to collapsing the graph-style witness sentence $G_{\text{wf}} = E(\bar{e})$ to its canonical unprovability normal form $G_{\text{nf}} = \neg \text{Prov}_T(\bar{d})$ (Step 5).

This confirms that, in Gödel's arithmetical setting, the internal representation of the flipped diagonal is neither global nor enumeration-wise but *local*: it is realized not by an object representing the *entire* flipped-diagonal sequence $(V_i)_{i \in \mathbb{N}}$, but by producing the witness sentence (and its normal form) for the diagonal flip at a *specific diagonal input*, namely $E(\bar{e})$.

6.3 Type discipline

It is well known that diagonal-flipping arguments can lead to two broad kinds of outcomes: paradoxical (Russell, Grelling–Nelson) and structural-limitative (Cantor, Turing, Gödel). The contrast can be diagnosed in terms of type discipline. In untyped settings, where levels collapse and self-application is unrestricted, diagonalization produces an internal inconsistency. On the other hand, in stratified settings (Cantor: set vs. element; Turing: machine vs. code; Gödel: formula vs. number/numeral), diagonal self-evaluation is mediated by an additional interface (surjectivity, coding, representability of provability, etc.), and the contradiction refutes an extra closure principle rather than the ambient typed framework.

To make precise the connection between typing discipline and paradoxical/limitative outcomes, we adopt a simple typed metalanguage in the style of HOL. We assume:

- Type assignment for a term t is written $t :: \tau$.
- Base types include $\alpha, \beta, \gamma, \dots, \text{nat}$ for natural numbers, and bool for Booleans.
- Function types are written $\sigma \Rightarrow \tau$.
- Sets of α -objects are represented extensionally as predicates

$$A :: \alpha \Rightarrow \text{bool},$$

and membership is application: $x \in A$ abbreviates $A(x)$.

- Complementation/negation is boolean negation: if $p :: \text{bool}$ then $\neg p :: \text{bool}$.
- α list denotes finite lists over type α .
- α option denotes partiality (via None/Some).

In this discipline, diagonal paradoxes correspond to ill-typed self-application of the form $u(u)$, i.e. cases where a term is simultaneously treated as an input and as

a predicate/function on the same domain. By contrast, in stratified settings, typing prevents this collapse: non-paradoxical diagonal arguments arise when the system enforces a separation of types between objects and their classifiers (i.e. predicates or functions acting on them).

We now revisit each classical instance under this typed perspective, hence clarifying the connection between typing and outcome.

6.3.1 Cantor.

Let α be the type of elements of a base set K , with K being the whole domain of type α . Represent subsets of K extensionally as predicates $S :: \alpha \Rightarrow \text{bool}$, so that $U := \mathcal{P}(K)$ is identified with the function space $\alpha \Rightarrow \text{bool}$.

Membership is application, and the interaction map has type

$$\Phi :: (\alpha \Rightarrow \text{bool}) \Rightarrow \alpha \Rightarrow \text{bool}, \quad \Phi(A, k) := A(k).$$

Assume for contradiction that some map $f :: \alpha \Rightarrow (\alpha \Rightarrow \text{bool})$ is surjective onto $\alpha \Rightarrow \text{bool}$. For matrix illustration, fix an enumeration $(k_i)_{i \in \mathbb{N}}$ of the elements of K , and define the matrix

$$M(i, j) := \Phi(f(k_i), k_j).$$

The corresponding enumeration-diagonal operator is

$$D^e(i) := M(i, i) = \Phi(f(k_i), k_i) = f(k_i)(k_i).$$

Let $flip :: \text{bool} \Rightarrow \text{bool}$ be boolean negation $flip(p) := \neg p$. The diagonal flip produces the anti-diagonal predicate

$$\delta_{\text{anti}} :: \alpha \Rightarrow \text{bool}, \quad \delta_{\text{anti}}(k) := flip(\Phi(f(k), k)) = \neg f(k)(k).$$

By surjectivity, there exists $k_0 \in K$ with $f(k_0) = \delta_{\text{anti}}$, hence

$$\delta_{\text{anti}}(k_0) = \neg f(k_0)(k_0) = \neg \delta_{\text{anti}}(k_0),$$

a contradiction. Crucially, no ill-typed self-application occurs: δ_{anti} has type α -objects and is only applied to elements $k :: \alpha$, never to itself. The contradiction therefore refutes the additional closure principle (surjectivity into the full function classifier space $\alpha \Rightarrow \text{bool}$), yielding a structural limitation rather than an internal paradox. This is reinforced by the fact that Cantor's argument admits a proof without *reductio* [3].

6.3.2 Russell.

Fix types so that individuals have type β and “sets of individuals” have type $\beta \Rightarrow \text{bool}$.

Membership is again application via

$$\Phi :: (\beta \Rightarrow \text{bool}) \Rightarrow \beta \Rightarrow \text{bool}, \quad \Phi(A, a) = A(a).$$

Let *flip* be boolean negation. Russell's paradox aims to define the anti-diagonal predicate

$$R(x) := \text{flip}(\Phi(x, x)) = \neg\Phi(x, x),$$

i.e. “the set of all sets that do not contain themselves”. However, the diagonal instance $\Phi(x, x)$ is already ill-typed, since in the naive/untyped setting one implicitly identifies the element-type β with the predicate-type $\beta \Rightarrow \text{bool}$: the first argument of Φ must have type $\beta \Rightarrow \text{bool}$, while the second argument must have type β , so an $x :: \beta \Rightarrow \text{bool}$ cannot be supplied as the second argument of type β . Thus Russell's paradox requires a type collapse identifying

$$\beta \cong (\beta \Rightarrow \text{bool}),$$

so that predicates could be fed to themselves as arguments. Simple typing discipline blocks precisely this identification. Thus, on a typed reading, Russell's paradox is not a theorem but a diagnosis: untyped comprehension implicitly forces a type collapse enabling terms of the form $u(u)$. In other words, the paradox is precisely the symptom of refusing the stratification between “individuals” and “sets.”

6.3.3 Grelling–Nelson.

Let γ be the type of linguistic tokens (e.g. adjectives), and let application classifications of such expressions be predicates of type $\gamma \Rightarrow \text{bool}$. Let

$$\Phi :: \gamma \Rightarrow \gamma \Rightarrow \text{bool}$$

be the application relation. The diagonal entries $\Phi(t, t)$ record whether a token applies to itself (autological).

Let *flip* be boolean negation. The diagonal flip defines the anti-diagonal predicate

$$h :: \gamma \Rightarrow \text{bool}, \quad h(x) := \text{flip}(\Phi(x, x)) = \neg\Phi(x, x),$$

which classifies each expression x by flipping its diagonal status, i.e. it picks out the tokens that do not apply to themselves. We call these terms “heterological.”

The paradox arises from a naming assumption: that the externally defined predicate $h :: \gamma \Rightarrow \text{bool}$ is the application-behavior of some token $t_h :: \gamma$, i.e.

$$\forall x :: \gamma. \Phi(t_h, x) \leftrightarrow h(x).$$

This is tantamount to assuming that a predicate $h :: \gamma \Rightarrow \text{bool}$ is represented by a token $t_h :: \gamma$ in the same domain, which is a type-level identification of expressions with predicates-on-expressions. In other words, the paradox arises from asking

whether the predicate that applies to heterological tokens is itself a token, i.e. the adjective named “heterological.”

Equivalently, the paradox can be interpreted as consequence of the assumption that every predicate on tokens is expressible by some token - i.e., that the map sending a token t to its extension $x \mapsto \Phi(t, x)$ is surjective onto $\gamma \Rightarrow \text{bool}$. Thus, Grelling–Nelson does not exhibit an inconsistency of the typed framework, but rather diagnoses a level collapse implicit in untyped natural-language formulations.

6.3.4 Turing.

Let inputs be finite strings of type $\Sigma \text{ list}$ over an alphabet Σ , and let Turing machines be computable partial functions on strings with type $\Sigma \text{ list} \Rightarrow \Sigma \text{ list option}$. Set X as the set of Turing machines.

Fix an effective enumeration $e_X : \mathbb{N} \rightarrow X$ of machines, writing $x_i := e_X(i)$. Let $\Phi(m, w) :: \text{bool}$ express halting of machine m on input w :

$$\Phi :: (\Sigma \text{ list} \Rightarrow \Sigma \text{ list option}) \Rightarrow \Sigma \text{ list} \Rightarrow \text{bool}.$$

Diagonal access is mediated by a computable coding map

$$\langle \cdot \rangle :: (\Sigma \text{ list} \Rightarrow \Sigma \text{ list option}) \Rightarrow \Sigma \text{ list}$$

sending a machine m to a string $\langle m \rangle$ encoding its description. Define the diagonal predicate

$$D^e(i) := \Phi(x_i, \langle x_i \rangle),$$

and with $flip$ being boolean negation, define the flipped diagonal data

$$V(i) := flip(D^e(i)) = \neg \Phi(x_i, \langle x_i \rangle).$$

The flipped diagonal data $V(i) = \neg \Phi(x_i, \langle x_i \rangle)$ is well-defined externally as a truth-pattern, but it need not be computable from i . Hence there is no reason for V to be realized by any machine in X . To internalize the diagonal flip one therefore assumes a total evaluator/decider E which, given codes of a machine and an input, correctly decides the truth of Φ .

Under this assumption one can effectively construct a machine x^* that, on coded diagonal inputs, realizes the anti-diagonal specification:

$$\forall i \in \mathbb{N}, \quad \Phi(x^*, \langle x_i \rangle) \leftrightarrow \neg \Phi(x_i, \langle x_i \rangle).$$

Diagonal self-evaluation of x^* at $\langle x^* \rangle$ then yields the contradiction: running x^* on its own code $\langle x^* \rangle$ forces

$$\Phi(x^*, \langle x^* \rangle) \leftrightarrow \neg \Phi(x^*, \langle x^* \rangle).$$

Thus the outcome is limitative: the contradiction refutes the additional decidability assumption, rather than producing a paradox internal to the typed framework. This is precisely enforced by stratification: machines are applied to codes/inputs, not to themselves as arguments of the same type.

6.3.5 Gödel.

In Gödel's incompleteness argument, the diagonal-flip pattern is implemented in a stratified way: formulas are not applied to themselves as predicates on formulas. Instead, self-reference is mediated by a coding boundary (formulas $\rightsquigarrow$ numbers) together with type-correct syntactic constructors.

Let form be the type of formulas. Fix a Gödel numbering of type $\text{form} \Rightarrow \text{nat}$. Introduce refinement types for the intended syntactic code domains:

$$\begin{aligned}\text{wff}_1 &:= \{n :: \text{nat} \mid n \in \text{Wff}_1\}, \\ \text{sent} &:= \{n :: \text{nat} \mid n \in \text{Sent}\}, \\ \text{termcode} &:= \{t :: \text{nat} \mid t \text{ codes a term of } T\}.\end{aligned}$$

Thus wff_1 is the type of codes of one-free-variable formulas (class-signs), sent is the type of codes of sentences. This stratification is the typed analogue of the code-objects of **PRCodes** from Section 4. The extra refinement type termcode makes explicit the number/numeral boundary: there is a p.r. numeralization constructor (see Section 4)

$$\text{num} :: \text{nat} \Rightarrow \text{termcode}, \quad \text{num}(n) = \ulcorner \bar{n} \urcorner,$$

sending a meta-level number to the code of the corresponding object-language numeral.

Substitution is then a syntactic code-transformer with type

$$\text{Sub} :: \text{wff}_1 \Rightarrow \text{nat} \Rightarrow \text{sent}, \quad \text{Sub}(b, a) = \ulcorner B(\bar{a}) \urcorner,$$

where $b = \ulcorner B(v_0) \urcorner$ and the second argument a is a meta-level number (it may itself be a code, e.g. an element of type wff_1). The conversion $a \mapsto \bar{a}$ is mediated by num . This induces the matrix-filling map of Subsection 6.2:

$$\Phi :: \text{wff}_1 \Rightarrow \text{wff}_1 \Rightarrow \text{sent}, \quad \Phi(a, b) := \text{Sub}(b, a).$$

We can then isolate the typed diagonal extractor

$$D^c :: \text{wff}_1 \Rightarrow \text{sent}, \quad D^c(a) := \Phi(a, a) = \text{Sub}(a, a).$$

Notice that this “diagonal” is not an ill-typed self-application $u(u)$: no formula-code a is treated as a predicate on formula-codes. Rather, D^c crosses a type boundary from wff_1 to sent by producing a closed sentence code, and the apparent “self” component is carried only by the numeralization map $a \mapsto \text{num}(a)$ inside the substitution operation.⁴

⁴ D^c can be viewed as the object-level D^m specialized to the p.r. code setting. In fact, since row-objects and column-objects coincide ($X = Y :: \text{wff}_1$), diagonal access is given by $s = \text{id}$, such that $D^m(a) := \Phi(a, s(a)) = \Phi(a, a) = \text{Sub}(a, a)$.

The flip operator is the provability-negating code builder

$$\text{flip} :: \text{sent} \Rightarrow \text{sent}, \quad \text{flip}(n) = g^c(n) = \ulcorner \neg \text{Prov}_T(\bar{n}) \urcorner.$$

Hence the flipped diagonal data is

$$V(i) := \text{flip}(\Phi(a_i, a_i)) = g^c(\text{Sub}(a_i, a_i)).$$

Equivalently, the uniform anti-diagonal operator exists externally as

$$\delta_{\text{anti}} :: \text{wff}_1 \Rightarrow \text{sent}, \quad \delta_{\text{anti}}(a) := \text{flip}(D^c(a)) = (g^c \circ D^c)(a) = c(a).$$

However, the independent sentence is obtained not by representing the entire flipped diagonal sequence by a single object, but locally by choosing a specific one-free-variable formula $E(x)$ and letting $e = \ulcorner E(v_0) \urcorner \in \text{Wff}_1$. Then diagonal evaluation yields

$$d := D^c(e) = \text{Sub}(e, e) = \ulcorner E(\bar{e}) \urcorner :: \text{sent},$$

the code of the witness sentence $G_{\text{wf}} := E(\bar{e})$. By the Normalization Lemma, T proves that this witness collapses to the canonical unprovability normal form

$$G_{\text{nf}} := \neg \text{Prov}_T(\bar{d}),$$

so that

$$T \vdash G_{\text{wf}} \leftrightarrow G_{\text{nf}}.$$

Thus, Gödel's diagonal flip is type-disciplined: self-reference is mediated by the coding and numeralization interfaces represented by the type boundary $\text{form} \rightarrow \text{nat}$ and the number-to-numeral constructor num . The resulting contradiction - under additional closure assumptions on T such as negation completeness - yields a limitation theorem rather than an internal paradox from unrestricted self-application.

6.3.6 Proposition and synthesis.

These examples show that diagonal flipping is a uniform combinatorial procedure, but the paradoxical or limitative outcome depends on type discipline. We can then formulate the following proposition, which formalizes Step 5 of the blueprint (Subsection 6.2) in a typed metalanguage.

Proposition 1 (Stratified diagonal flip yields non-paradoxical limitation) *Work in a simply typed λ -calculus with base types ρ, σ, τ , a matrix map*

$$\Phi :: \rho \Rightarrow \sigma \Rightarrow \tau,$$

a flip operator $\text{flip} :: \tau \Rightarrow \tau$, and indexed families $(x_i)_{i \in \mathbb{N}}$ with $x_i :: \rho$ and $(y_i)_{i \in \mathbb{N}}$ with $y_i :: \sigma$. Define the diagonal-flipped data

$$V(i) := \text{flip}(\Phi(x_i, y_i)).$$

Assume the typing discipline prevents self-classification. Concretely, while it is harmless (and often intended) to have a classifier collapse

$$\rho \cong (\sigma \Rightarrow \tau),$$

so that row objects may be regarded as row-functions on σ (e.g. in Cantor's case), the type discipline excludes the self-classification collapse obtained when moreover $\rho = \sigma$, namely

$$\rho \cong (\rho \Rightarrow \tau),$$

which would permit ill-typed self-application of the form $u(u)$. (In particular, in boolean paradox settings one blocks $\rho \cong (\rho \Rightarrow \text{bool})$.)

Then the diagonal-flip pattern by itself cannot produce an internal paradox of Russell/Grelling–Nelson form. Any contradiction obtained from the diagonal-flipped data V must therefore rely on an additional closure/representation principle which (a) realizes the anti-diagonal behavior internally by some $x^* :: \rho$, and (b) supplies a typed notion of self-input (or diagonal access) for that realizer, e.g. via a map $s :: \rho \Rightarrow \sigma$. Consequently, diagonal reasoning refutes such an additional principle, yielding a structural limitation theorem rather than an inconsistency of the ambient typed framework.

Proof sketch Fix indexed families $x_i :: \rho$ and $y_i :: \sigma$. By definition, the diagonal-flipped data is an externally specified sequence

$$V(i) = \text{flip}(\Phi(x_i, y_i)).$$

By itself, the existence of V yields no contradiction: it merely specifies a pattern of diagonal-complemented outputs.

To obtain a contradiction one needs, first, an internal realizer for this pattern, i.e. some $x^* :: \rho$ satisfying a representability condition such as

$$\forall i \in \mathbb{N}. \Phi(x^*, y_i) = V(i),$$

or, in the stronger global form (when an anti-diagonal row function $\delta_{\text{anti}} :: \sigma \Rightarrow \tau$ is available),

$$\forall y :: \sigma. \Phi(x^*, y) = \delta_{\text{anti}}(y).$$

Second, even given such an x^* , a diagonal contradiction requires a well-typed instance feeding the realizer back into the diagonal, i.e. an interaction of the form

$$\Phi(x^*, s(x^*))$$

for some diagonal-access map $s :: \rho \Rightarrow \sigma$.

In an unstratified or level-collapsing framework one typically has $\rho = \sigma$ and may take $s = \text{id}$, so diagonal access is automatic. A paradox of Russell/Grelling–Nelson form arises only if, in addition, row objects can be treated as classifiers on their inputs, i.e. $\rho \cong (\sigma \Rightarrow \tau)$. In fact, when $\rho = \sigma$ this becomes the self-classification collapse

$\rho \cong (\rho \Rightarrow \tau)$ (and in boolean settings, $\rho \cong (\rho \Rightarrow \text{bool})$), which permits self-application $u(u)$ and thereby enables a direct internal contradiction of the schematic form

$$\Phi(x^*, x^*) = \text{flip}(\Phi(x^*, x^*)).$$

By contrast, in a stratified typed setting the self-classification collapse is blocked: even when diagonal access is available, there is no coercion $\rho \cong (\rho \Rightarrow \tau)$, and thus terms of the form $u(u)$ are ill-typed. Diagonal-flip reasoning can still yield contradictions, but only by combining representability (i.e., an internal realizer x^*) with an additional interface s supplying a self-input of the appropriate type. Different diagonal arguments provide such interfaces in different ways:

- Cantor: One takes $\tau = \text{bool}$, $\rho = \sigma \Rightarrow \text{bool}$, and diagonal access is mediated by an element input $s :: \rho \Rightarrow \sigma$ (evaluation at an element). The contradiction refutes an additional closure principle, namely surjectivity onto the full function space $\sigma \Rightarrow \text{bool}$.
- Turing: One has a coding map $\langle \cdot \rangle :: \rho \Rightarrow \sigma$ and assumes a total evaluator/decider for Φ . This yields an internal realizer x^* implementing the flipped diagonal behavior on coded inputs, and diagonal access is mediated by $s(m) := \langle m \rangle$. Self-evaluation at $\langle x^* \rangle$ refutes the assumed decidability/evaluation principle.
- Gödel: One internalizes diagonal access via a coding/numeralization interface together with p.r. syntactic constructors (e.g. substitution and provability predicates). The contradiction arises only under an additional closure assumption on T (e.g. negation-completeness), which is then refuted under standard consistency/soundness hypotheses.

Hence, under a type discipline that blocks self-classification collapse, diagonal-flip reasoning yields limitation results: whenever it produces a contradiction, the target is an added representability/closure principle (surjectivity in Cantor, decidability in Turing, negation completeness in Gödel), not the ambient typed framework itself. $\square$

In synthesis, the same diagonal-flipping blueprint can yield outcomes of different logical character depending on whether the underlying framework enforces type discipline. In systems lacking such discipline, paradoxical constructions (Russell, Grelling–Nelson) emerge; these arguments are methodologically significant because they expose *collapses* and ambiguities that motivate a revision or stratification of the ambient framework.

By contrast, in frameworks that do enforce type discipline (Cantor, Turing, Gödel), the diagonal-flipping blueprint highlights limitative structural *properties* of the systems to which it is applied. These properties are neither defects nor “bugs” to be eliminated, nor can they be removed by ad hoc repairs without departing from the very systems that possess them. Rather, they are *constitutive features* of the systems’ expressive and computational power, and any further study or use of such systems must acknowledge and incorporate them.

7 Takeaway and future directions

7.1 Takeaway

This paper re-reads the construction of the independent sentence for T through the lens of explicit code computation. Rather than treating the diagonal lemma as a “black box” fixed-point principle, we make the relevant syntactic self-reference emerge as the output of a concrete p.r. pipeline on GNs:

$$c = g^c \circ D^c = g^c \circ f^c \circ \Delta^c : \text{Wff}_1 \rightarrow \text{Sent}.$$

Two immediate conceptual payoffs follow.

First, moving from Lindenbaum classes to codes resolves the well-definedness issue in substitution-like maps: operations such as numeral-substitution are sensitive to syntactic shape and are not invariant under provable equivalence. The code-level setting enforces a type discipline that keeps the construction in check.

Second, the construction separates two logically distinct outputs: a *witness* sentence $G_{\text{wf}} = E(\bar{e})$ (with $\ulcorner G_{\text{wf}} \urcorner = d$) whose self-reference is mediated through the functional graph predicate $\text{Diag}_T(x, y)$, and its *normalized* form $G_{\text{nf}} = \neg \text{Prov}_T(\bar{d})$ which is obtained once diagonalization is internally evaluated. The Normalization Lemma makes this collapse explicit inside T :

$$T \vdash G_{\text{wf}} \leftrightarrow G_{\text{nf}}.$$

This witness/normal-form distinction clarifies where the “self” component resides (namely, in the code/numeral interface) and how the canonical unprovability normal form of the independent sentence is recovered.

Beyond the core construction, the paper develops two further contributions:

- **Variants as modular replacements.** Section 5 shows that the pipeline presentation is not tied to a single provability predicate. Once diagonalization is isolated as the typed map $D^c(x) = \text{Sub}(x, x)$, strengthening and variation are obtained by substituting the flip component. In particular, Rosser’s improvement is captured by retaining the same diagonal map and replacing the provability-flip code-builder g^c by its Rosser analogue g^R . The matrix visualizations further emphasize this modularity: changing the “flip” operation changes the diagonal transform while keeping the substitution matrix structure fixed, and the single-matrix presentation packages diagonalization and flipping into a single two-dimensional object.
- **The diagonal-flipping blueprint and type discipline.** Section 6 abstracts the preceding construction into a general diagonal-flipping blueprint organized into explicit methodological steps: (i) choose the relevant data types and enumerations; (ii) establish an appropriate uniform interaction map; (iii) extract and flip the diagonal; (iv) establish the internal realization of the resulting anti-diagonal behavior; and (v) analyze the form of self-evaluation permitted by the framework. This schema makes precise what is shared between Cantor-style arguments, semantic and syntactic paradoxes (Russell and Grelling–Nelson), computability-

theoretic diagonalization (Turing), and Gödel's arithmetized self-reference, while also diagnosing where they diverge.

A key outcome of this blueprint is that the logical character of the conclusion depends on *type discipline*. In unstratified settings, diagonal self-application collapses levels and enables paradoxical expressions of the form $u(u)$. In stratified settings (Cantor: element vs. set; Turing: machine vs. code; Gödel: formula vs. number/numeral), diagonal self-evaluation is mediated by an interface (surjectivity, evaluation, coding, representability of provability), and the following contradiction refutes an additional closure principle rather than producing an internal inconsistency. In this sense, the blueprint explains why diagonal arguments sometimes yield contradictions internal to a naive framework, while in typed or stratified settings they yield structural limitation theorems.

7.2 Future directions

In light of the analysis above, at least two research directions appear natural.

1. **Kleene-style incompleteness via recursive inseparability.** Kleene's classical reformulation of incompleteness proceeds "symmetrically" by exhibiting two recursively enumerable sets that are recursively inseparable, rather than by directly presenting a single self-referential sentence [18, 19]. On this view, negation-completeness would amount to an effective separation of the codes of provable sentences from those of refutable sentences, and the contradiction is obtained by showing that no such separator can exist.

A promising direction is to transport the present paper's typed composition of code-transformers into Kleene's setting. Conceptually, the diagonal-flipping pattern persists, but the roles shift: the inputs become indices (or codes) for recursively enumerable sets and Σ_1^0 predicates rather than one-free-variable formula codes, and the self-reference typically sits in the indexing apparatus (e.g. prime-power or pairing codings) rather than in literal syntactic substitution. Developing an explicit pipeline (analogous to Δ^c , f^c , g^c) for Kleene's inseparability proof would make the computational core of that argument comparably transparent, and may yield a clean bridge between "independent sentence" presentations and "undecidable separation" presentations of G1.

2. **Proof mechanization and automation.** A second direction is to investigate how the code-based construction and the blueprint of Section 6 can serve as an alternative basis for mechanized proofs of incompleteness. The present approach is particularly compatible with interactive theorem proving environments whose native metalogic is a simply typed λ -calculus (e.g. HOL-family systems), since the argument is organized around type-correct morphisms between syntactic code-domains.

There is already a substantial mechanization literature on incompleteness and its variants. Early work includes Shankar's mechanization of a Rosser-style incompleteness argument in Nqthm [4] for a first-order theory of hereditarily finite sets [37, 38]. O'Connor then produced a constructive proof of the Gödel–Rosser theorem for weak arithmetical theories in Rocq [8] (formerly Coq), hence pre-

senting the first fully computer-verified proof of the essential incompleteness of arithmetic [24]. Harrison mechanized a Gödel-style incompleteness development in HOL Light [13], emphasizing semantic definability over the standard model $\mathbb{N}$ [14, 15]. Paulson formalized for the first time *both* incompleteness theorems in Isabelle/HOL [23] via hereditarily finite set theory [25–28]. More recently, Popescu and Traytel developed a flexible and logic-agnostic Isabelle/HOL infrastructure supporting multiple variants and proof routes of both incompleteness theorems [30–32]. In a complementary direction, a further line of work by Kirst and collaborators mechanizes a synthetic approach via undecidability [16], including Kleene-style route to incompleteness [17] and the mechanization of Löb’s theorem in Rocq [1]. For a taxonomy of the mechanizations of the incompleteness theorems, see [41].

Against this background, a concrete next step is to formalize the specific pipeline of this paper (Δ^c , Sub , D^c , g^c , and the normalization argument) in a proof assistant and to investigate how far it can be automated. The normalization step is particularly attractive from this point of view: it is short, compositional, and relies on functionality of a representable p.r. graph predicate. If successful, such a mechanization could provide a modular library of diagonal-flipping components reusable across variants (Rosser-style improvement, Löb/GL-style provability arguments, and other diagonal arguments), and it may help bridge interactive formalization with increased proof automation for incompleteness results.

Acknowledgements I am grateful to an anonymous reviewer for a careful reading of earlier versions of this paper. Their comments led to a sharper positioning of the paper’s contribution relative to Gödel’s original argument and to a more explicit development of the categorical reading in Sect. 4.

Author Contributions A.V. conducted the research, developed the theoretical framework, wrote the manuscript, and prepared all figures.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Bailitis, J., Kirst, D., Forster, Y.: Löb's Theorem and Provability Predicates in Rocq. TYPES 2025 (2025)
2. Beklemishev, L.D.: Gödel incompleteness theorems and the limits of their applicability. I. Russian Mathematical Surveys **65**(5), 857–899 (2010). <https://doi.org/10.1070/RM2010v065n05ABEH004703>
3. Benz Müller, C., Fuenmayor, D.: Cantor's Theorem without Reductio Ad Absurdum. Preprint (2021). <https://doi.org/10.13140/RG.2.2.31069.95201/1>
4. Boyer, R., Kaufmann, M., Moore, J.: The Boyer-Moore theorem prover and its interactive enhancement. Comput. Math. Appl. **29**(2), 27–62 (1995). [https://doi.org/10.1016/0898-1221\(94\)00215-7](https://doi.org/10.1016/0898-1221(94)00215-7)
5. Buldt, B.: The Scope of Gödel's First Incompleteness Theorem. Log. Univers. **8**, 499–552 (2014). <https://doi.org/10.1007/s11787-014-0107-3>
6. Cantor, G.: Über eine elementare Frage der Mannigfaltigkeitslehre. Jahresber. Deutsch. Math.-Verein. **1**, 75–78 (1891)
7. Carnap, R.: Logical Syntax of Language. Routledge, Abingdon (UK) (2014)
8. The Rocq Development Team.: The Rocq Prover Reference Manual. INRIA (2026). <https://doi.org/10.5281/zenodo.1003420>
9. Gaifman, H.: Naming and diagonalization, from Cantor to Gödel to Kleene. Logic Journal of the IGPL **14**(5), 709–728 (2006). <https://doi.org/10.1093/jigpal/jzl006>
10. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik **38**(1), 173–198 (1931)
11. Gödel, K.: On formally undecidable propositions of Principia Mathematica and related systems. In: Feferman, S. (ed.), Kurt Gödel, Collected Works, Vol. I. Oxford University Press, Oxford, (1986), 144–195. First published 1931 in Monatshefte für Mathematik und Physik
12. Grelling, K., Nelson, L.: Bemerkungen zu den Paradoxien von Russell und Burali-Forti. Abhandlungen der Fries'schen Schule. Neue Folge **2**, 301–334 (1908)
13. Harrison, J.: HOL Light: an overview. In: International Conference on Theorem Proving in Higher Order Logics. Springer, Berlin, Heidelberg, 60–66 (2009). https://doi.org/10.1007/978-3-642-03359-9_4
14. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, Cambridge (2009)
15. Harrison, J.: HOL Light proof of Gödel's first incompleteness theorem. URL = <https://github.com/jrh13/hol-light/blob/master/Arithmetic/godel.ml>
16. Kirst, D., Hermes, M.: Synthetic Undecidability and Incompleteness in First-Order Axiom Systems in Coq. J. Autom. Reason. **67**(13), 1–31 (2023). <https://doi.org/10.1007/s10817-022-09647-x>
17. Kirst, D., Peters, B.: Gödel's Theorems without Tears: Essential Incompleteness in Synthetic Computability. In: 31st EACSL Annual Conference on Computer Science Logic (CSL 2023), LIPIcs (2023): 30:1–30:18. <https://doi.org/10.4230/LIPIcs.CSL.2023.30>
18. Kleene, S.C.: A Theory of Positive Integers in Formal Logic. American Journal of Mathematics **57** (1935): 153–173; 219–244. (Parts I & II)
19. Kleene, S.C.: A symmetric form of Gödel's theorem. Koninklijke Nederlandse Akademie van Wetenschappen, Proceedings of the Section of Sciences **53**, 800–802 (1950)
20. Kleene, S.C.: A symmetric form of Gödel's theorem. Koninklijke Nederlandse Akademie van Wetenschappen, Indagationes Mathematicae **12**, 244–246 (1950)
21. Lawvere, F.W.: Diagonal arguments and cartesian closed categories. In: Mitchell, B., et al. (eds.) Category Theory, Homology Theory and their Applications II: Proceedings of the Conference held at the Seattle Research Center of the Battelle Memorial Institute, June 24–July 19, 1968, Volume Two. Springer, Berlin, Heidelberg, 134–145 (1969)
22. Lawvere, F.W.: Diagonal arguments and Cartesian closed categories. Reprints in Theory and Applications of Categories **15**, 1–13 (2006). (Reprinted from Category Theory, Homology Theory and their Applications II (Battelle Institute Conference, Seattle, Wash., 1968, Vol. Two), pp. 134–145. <https://doi.org/10.1007/BFb0080769>)
23. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL-A Proof Assistant for Higher-Order Logic. Springer, Berlin, Heidelberg (2002)

24. O'Connor, R.: Essential incompleteness of arithmetic verified by Coq. In: International Conference on Theorem Proving in Higher Order Logics. Springer, Berlin, Heidelberg, 245–260 (2005). https://doi.org/10.1007/11541868_16
25. Paulson, L.C.: Gödel's Incompleteness Theorems. Archive of Formal Proofs (2013). URL = <https://isa-afp.org/entries/Incompleteness.html>
26. Paulson, L.C.: A machine-assisted proof of Gödel's incompleteness theorems for the theory of hereditarily finite sets. The Review of Symbolic Logic 7(3), 484–498 (2014). <https://doi.org/10.1017/S1755020314000112>
27. Paulson, L.C.: A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. J. Autom. Reason. 55, 1–37 (2015). <https://doi.org/10.1007/s10817-015-9322-8>
28. Paulson, L.C., Bailitis, J.: Gödel's Incompleteness Theorems. Archive of Formal Proofs (2025). URL = https://www.isa-afp.org/browser_info/current/AFP/Incompleteness/document.pdf
29. von Plato, J.: Can Mathematics be Proved Consistent?: Gödel's Shorthand Notes and Lectures on Incompleteness. Springer Nature, Cham (2020)
30. Popescu, A., Traytel, D.: A formally verified abstract account of Gödel's incompleteness theorems. In: Fontaine, P. (ed.) Automated Deduction – CADE 27. CADE 2019. Lecture Notes in Computer Science, vol. 11716. Springer, Cham, 442–461 (2019)
31. Popescu, A., Traytel, D.: Distilling the requirements of Gödel's incompleteness theorems with a proof assistant. J. Autom. Reason. 65(7), 1027–1070 (2021)
32. Popescu, A., Traytel, D.: An abstract formalization of Gödel's incompleteness theorems. Archive of Formal Proofs (2023). URL = https://www.isa-afp.org/entries/Goedel_Incompleteness.html
33. Raatikainen, P.: Gödel's Incompleteness Theorems. The Stanford Encyclopedia of Philosophy (Spring 2022 Edition), Edward N. Zalta (ed.). URL = <https://plato.stanford.edu/archives/spr2022/entries/goedel-incompleteness/>
34. Roberts, D.M.: Substructural fixed-point theorems and the diagonal argument: theme and variations. Compositionality 5 (2023): 8. Preprint: <https://arxiv.org/abs/2110.00239>
35. Rosser, B.: Extensions of Some Theorems in Gödel and Church. The Journal of Symbolic Logic 1(3), 87–91 (1936)
36. Russell, B.: The Principles of Mathematics. Cambridge University Press, Cambridge (1903)
37. Shankar, N.: Proof-checking Metamathematics. PhD Thesis, The University of Texas at Austin, (1986)
38. Shankar, N.: Metamathematics. Machines and Gödel's Proof. Cambridge University Press, Cambridge (1997)
39. Sipser, M.: Introduction to the Theory of Computation, 2nd edn. Thomson, Boston (MA) (2006)
40. Turing, A.M.: On Computable Numbers, with an Application to the Entscheidungsproblem. Proc. Lond. Math. Soc. 42(1), 230–265 (1936). <https://doi.org/10.2307/2268810>
41. Vestrucci, A.: Mechanizations of Gödel's Incompleteness Theorems. Logic and Logical Philosophy. Forthcoming
42. Yanofsky, N.S.: A universal approach to self-referential paradoxes, incompleteness and fixed points. Bulletin of Symbolic Logic 9(3), 362–386 (2003)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.