Reasoning Around Recursion

Grounded arithmetic: where it came from, and where it is

Omar

Indian Institute of Science

joint work with the Grounded Deduction group, EPFL DEDIS

September 7, 2026

Two programs

C

```
main() { printf("Hi!"); }
```

```
main() { main(); }
```

Both compile.

Coq / Lean / Agda / Isabelle

```
Fixpoint loop (n:nat) := loop n.
```

Rejected.

Why?

And again, with datatypes

Haskell

```
data Expr
  = Base
  | App Expr Expr
  | Abs (Expr -> Expr)
```

Compiles.

Coq

```
Inductive Expr :=
  | Base :  Expr
  | App :  Expr -> Expr -> Expr
  | Abs :  (Expr -> Expr) -> Expr.
```

Rejected.

And again, with datatypes

Haskell

```
data Expr
  = Base
  | App Expr Expr
  | Abs (Expr -> Expr)
```

Compiles.

Coq

```
Inductive Expr :=
  | Base :  Expr
  | App :  Expr -> Expr -> Expr
  | Abs :  (Expr -> Expr) -> Expr.
```

Rejected.

Non strictly positive occurrence of "Expr" in "(Expr -> Expr) -> Expr".

A *different* refusal — not termination, **positivity**.

Not one narrow check: a family of restrictions on what you may define.

The usual answer is wrong

"Termination checking is a conservative engineering choice."

The usual answer is wrong

"Termination checking is a conservative engineering choice."

It is not a choice.

It is the **consistency** of your proof assistant.

The usual answer is wrong

"Termination checking is a conservative engineering choice."

It is not a choice.

It is the **consistency** of your proof assistant.

Working out exactly why took fifteen years,
and ended two research programmes.

Background: why anyone wanted a way out

- **Frege** (1893): mathematics from logic, via extensions of concepts.
- **Russell** (1902): naive comprehension is inconsistent.
  $R = \{x : x \notin x\}$. Is $R \in R$?
- Two repairs, both in wide use by 1930:
  - **Russell** — *type theory*
  - **Zermelo** — *axiomatic set theory*

Background: why anyone wanted a way out

- **Frege** (1893): mathematics from logic, via extensions of concepts.
- **Russell** (1902): naive comprehension is inconsistent.
  $R = \{x : x \notin x\}$. Is $R \in R$?
- Two repairs, both in wide use by 1930:
  - **Russell** — *type theory*
  - **Zermelo** — *axiomatic set theory*

Type theory works by **stratification**: every object gets a level, and a collection may only contain objects of strictly lower level.

What stratification costs

Under stratification, $x\,x$ **is not false — it is ungrammatical.**
A thing may never be applied to itself. That is the whole mechanism.

$$|D \to D| \;\geq\; |\mathcal{P}(D)| \;>\; |D|$$

What stratification costs

Under stratification, $x\,x$ **is not false — it is ungrammatical.**
A thing may never be applied to itself. That is the whole mechanism.

$$|D \to D| \;\geq\; |\mathcal{P}(D)| \;>\; |D|$$

But self-application is exactly how you get **recursion**:
from $x\,x$ you can build a fixed-point combinator, and a fixed point solves *any* recursive equation.

What stratification costs

Under stratification, $x\,x$ **is not false — it is ungrammatical.**
A thing may never be applied to itself. That is the whole mechanism.

$$|D \to D| \;\geq\; |\mathcal{P}(D)| \;>\; |D|$$

But self-application is exactly how you get **recursion**:
from $x\,x$ you can build a fixed-point combinator, and a fixed point solves *any* recursive equation.

Ban it, and you lose general recursion.

What stratification costs

Under stratification, $x\,x$ **is not false — it is ungrammatical.**
A thing may never be applied to itself. That is the whole mechanism.

$$|D \to D| \;\geq\; |\mathcal{P}(D)| \;>\; |D|$$

But self-application is exactly how you get **recursion**:
from $x\,x$ you can build a fixed-point combinator, and a fixed point solves *any* recursive equation.

Ban it, and you lose general recursion.

> *"Rather than adopt the method of Russell . . . or that of Zermelo,* ***both of which appear somewhat artificial****, we introduce . . . a certain restriction on the law of excluded middle."*
>
> — Church 1932

Church and Curry: functions, not sets

"Both of them set out to base logic and mathematics on **functions** instead of on set theory."
— Seldin

Extensional — set theory

A function *is* its graph: the set of its input–output pairs. Two functions are the same iff they agree on every input.

So $\lambda n.\, 2n$ and $\lambda n.\, n + n$ are the *same* function.

Intensional — Church, Curry

A function is a *rule*: a finite description of how to get the output. Two functions are the same iff the descriptions are.

So $\lambda n.\, 2n$ and $\lambda n.\, n + n$ are *different* functions.

Church and Curry: functions, not sets

"Both of them set out to base logic and mathematics on **functions** instead of on set theory."
— Seldin

Extensional — set theory

A function *is* its graph: the set of its input–output pairs. Two functions are the same iff they agree on every input.

So $\lambda n.\, 2n$ and $\lambda n.\, n + n$ are the *same* function.

Intensional — Church, Curry

A function is a *rule*: a finite description of how to get the output. Two functions are the same iff the descriptions are.

So $\lambda n.\, 2n$ and $\lambda n.\, n + n$ are *different* functions.

A rule can be applied to a rule. Compilers compile compilers.
No stratification required.

Church, 1932

A Set of Postulates for the Foundation of Logic

- λ as a *logical* primitive, alongside $\Pi, \Sigma, \&, \sim, \iota, A$
- No free variables
- **Restricted excluded middle**: for some arguments $\mathbf{X}$, $\mathbf{F}(\mathbf{X})$ "is undefined and represents nothing"
- Keeps double negation; **abandons *reductio*** in certain cases
- Guarded quantifier: $\Pi(F, G)$ licenses $G(M)$ only once $F(M)$ is *known true*

Church, 1932

A Set of Postulates for the Foundation of Logic

- λ as a *logical* primitive, alongside $\Pi, \Sigma, \&, \sim, \iota, A$
- No free variables
- **Restricted excluded middle**: for some arguments $\mathbf{X}$, $\mathbf{F}(\mathbf{X})$ "is undefined and represents nothing"
- Keeps double negation; **abandons *reductio*** in certain cases
- Guarded quantifier: $\Pi(F, G)$ licenses $G(M)$ only once $F(M)$ is *known true*

Reject LEM · weaken *reductio* · let well-formed terms denote nothing
— comes up again in GA.

Church did not claim consistency

"Whether the system of logic which results from our postulates is adequate for the development of mathematics, and ***whether it is wholly free from contradiction****, are questions which we cannot now answer except by* ***conjecture****."*

— Church 1932, p. 348

Church did not claim consistency

> *"Whether the system of logic which results from our postulates is adequate for the development of mathematics, and* ***whether it is wholly free from contradiction****, are questions which we cannot now answer except by* ***conjecture****."*
>
> — Church 1932, p. 348

He proposed to find out *empirically*.

Curry, 1930

Grundlagen der kombinatorischen Logik

- Started (1922) from a different question: **substitution** in *Principia* is far more complicated than modus ponens — break it into simpler rules
- Combinators $\mathsf{I}, \mathsf{B}, \mathsf{C}, \mathsf{W}, \mathsf{K}, \mathsf{S}$; no bound variables at all
- Consistency proof for the *combinator* part — but no logical axioms given; that half was left for future work

Curry, 1930

Grundlagen der kombinatorischen Logik

- Started (1922) from a different question: **substitution** in *Principia* is far more complicated than modus ponens — break it into simpler rules
- Combinators $\mathsf{I}, \mathsf{B}, \mathsf{C}, \mathsf{W}, \mathsf{K}, \mathsf{S}$; no bound variables at all
- Consistency proof for the *combinator* part — but no logical axioms given; that half was left for future work

On the Russell term $[x](\neg(xx))$: Curry proposed to find postulates from which **it could not be proved that its self-application is a proposition** — while insisting the term *exist* in the system.

Kleene & Rosser, 1935

Both systems are strong enough to enumerate their own provably-total functions. That is enough to formalise **Richard's paradox**.

> *"We are sorry to say that we have a proof of the inconsistency of your system of formal logic."*

Kleene & Rosser, 1935

Both systems are strong enough to enumerate their own provably-total functions. That is enough to formalise **Richard's paradox**.

> *"We are sorry to say that we have a proof of the inconsistency of your system of formal logic."*
>
> *"I am afraid that I can not share your grief . . . you are not sorry but you are full of that profound satisfaction which comes from doing something worth while . . .* ***I hasten to congratulate you both.****"*
>
> — Curry to Kleene and Rosser, 19 November 1934

That — but not *why*

Kleene and Rosser showed the systems were inconsistent.
They did not isolate **which principle** was responsible.

- The proof was long, and specific to those systems
- Curry's response: a 60-page paper (1941) studying the paradox "so as to lay bare its central nerve"
- Church's response (1933): delete four postulates, the ones carrying *reductio* — **it did not save the system**

That — but not *why*

Kleene and Rosser showed the systems were inconsistent.
They did not isolate **which principle** was responsible.

- The proof was long, and specific to those systems
- Curry's response: a 60-page paper (1941) studying the paradox "so as to lay bare its central nerve"
- Church's response (1933): delete four postulates, the ones carrying *reductio* — **it did not save the system**

The answer came in 1942, in three pages.

Curry, 1942 — three pages

The Inconsistency of Certain Formal Logics, JSL 7(3):115–117

Suppose a system has:

1. λ-abstraction that computes (combinatory completeness)
2. $\vdash M \supset M$
3. **contraction**: from $\vdash M \supset (M \supset N)$ infer $\vdash M \supset N$
4. modus ponens

Curry, 1942 — three pages

The Inconsistency of Certain Formal Logics, JSL 7(3):115–117

Suppose a system has:

1. λ-abstraction that computes (combinatory completeness)
2. $\vdash M \supset M$
3. **contraction**: from $\vdash M \supset (M \supset N)$ infer $\vdash M \supset N$
4. modus ponens

Then **every term is provable.**

The derivation

Let $\mathfrak{A}$ be a term with $\mathfrak{A} \equiv (\mathfrak{A} \supset \mathfrak{B})$, obtained from a fixed point of $\lambda X.\, X \supset \mathfrak{B}$.

(1)	$\vdash \mathfrak{A} \supset \mathfrak{A}$	identity
(2)	$\vdash \mathfrak{A} \supset .\, \mathfrak{A} \supset \mathfrak{B}$	unfold the definition
(3)	$\vdash \mathfrak{A} \supset \mathfrak{B}$	contraction
(4)	$\vdash \mathfrak{A}$	fold: (3) *is* $\mathfrak{A}$
(5)	$\vdash \mathfrak{B}$	modus ponens

$\mathfrak{B}$ was arbitrary.

What is *not* in that proof

No negation.

What is *not* in that proof

No negation.

Identity, modus ponens, contraction, unfolding a definition.

Every one is valid **intuitionistically**. Every one is valid in **minimal logic**.

What is *not* in that proof

No negation.

Identity, modus ponens, contraction, unfolding a definition.

Every one is valid **intuitionistically**. Every one is valid in **minimal logic**.

Church had restricted excluded middle.
Church had abandoned *reductio*.

Neither was ever going to be enough.

Curry's own diagnosis

"Before he discovered this paradox, Curry had assumed that the contradiction was caused by his postulates for the universal quantifier, but this paradox made it clear that ***the problem was the postulates for implication alone****."*

— Seldin, *The Logic of Curry and Church*

Curry's own diagnosis

"Before he discovered this paradox, Curry had assumed that the contradiction was caused by his postulates for the universal quantifier, but this paradox made it clear that ***the problem was the postulates for implication alone****."*

— Seldin, *The Logic of Curry and Church*

Church guarded the *quantifier*. Curry blamed the *quantifier*.

It was implication.

What the proof actually needs

Curry states hypothesis 1 as **combinatory completeness**: for any term X in variables $x_1, \ldots, x_n$, there is a closed term A with

$$A\, x_1 \cdots x_n \;=\; X$$

That is: *abstraction is definable*. A is what we would write $\lambda x_1 \ldots x_n.\, X$.
He assumes it because that is what his systems offered.

What the proof actually needs

Curry states hypothesis 1 as **combinatory completeness**: for any term X in variables $x_1, \ldots, x_n$, there is a closed term A with

$$A\, x_1 \cdots x_n \;=\; X$$

That is: *abstraction is definable*. A is what we would write $\lambda x_1 \ldots x_n.\, X$.
He assumes it because that is what his systems offered.
But the proof consumes only *one* term:

$$\mathfrak{A} \qquad \text{with} \qquad \mathfrak{A} \;\equiv\; \mathfrak{A} \supset \mathfrak{B}$$

Abstraction supplies it, via a fixed-point combinator. So does a plain definitional mechanism: write $d \equiv d \to \mathfrak{B}$.

What the proof actually needs

Curry states hypothesis 1 as **combinatory completeness**: for any term X in variables $x_1, \ldots, x_n$, there is a closed term A with

$$A\, x_1 \cdots x_n \;=\; X$$

That is: *abstraction is definable*. A is what we would write $\lambda x_1 \ldots x_n.\, X$.
He assumes it because that is what his systems offered.
But the proof consumes only *one* term:

$$\mathfrak{A} \qquad \text{with} \qquad \mathfrak{A} \;\equiv\; \mathfrak{A} \supset \mathfrak{B}$$

Abstraction supplies it, via a fixed-point combinator. So does a plain definitional mechanism: write $d \equiv d \to \mathfrak{B}$.

The real killer is:

the system admits self-referential definitions.

Combinatory completeness is one route to that — not the only one, and not equivalent to it.

The shape of the result

Unrestricted recursion + ordinary implication

= proves everything.

The shape of the result

Unrestricted recursion + ordinary implication

= proves everything.

And that is why `Fixpoint loop (n:nat) := loop n.` is rejected.

The shape of the result

Unrestricted recursion + ordinary implication
= proves everything.

And that is why `Fixpoint loop (n:nat) := loop n.` is rejected.

Not conservatism. If your system admitted it, you could derive **False**.

After 1935

Church, Kleene and Rosser extracted the *pure* λ-calculus — terms and conversion, no logic — and Church retreated to simple type theory in 1940.

Curry did not retreat. He kept the untyped terms and added a predicate H for "is a proposition," using it to guard the logical rules.

After 1935

Church, Kleene and Rosser extracted the *pure* λ-calculus — terms and conversion, no logic — and Church retreated to simple type theory in 1940.

Curry did not retreat. He kept the untyped terms and added a predicate H for "is a proposition," using it to guard the logical rules.

Everything you use descends from Church 1940.

GA is closer to Curry's road.

Where everyone landed

	unrestricted recursion	ordinary rules	non-triviality
Church 1932 · Curry 1930	kept	weakened neg.	**lost**
STLC · Coq · Agda · Lean · HOL	**dropped**	kept	kept
Paraconsistent (Priest)	kept	**no explosion**	**contained**
Grounded arithmetic	**kept**	**preconditions**	**kept**

Everyone since 1942 kept implication and gave up recursion.

An aside — the same trap, from the other side

Curry's real hypothesis was not λ-abstraction. It was **self-referential definitions**:

$$\mathfrak{A} \equiv \mathfrak{A} \supset \mathfrak{B} \qquad \text{detonates through } \textbf{implication}.$$

An aside — the same trap, from the other side

Curry's real hypothesis was not λ-abstraction. It was **self-referential definitions**:

$$\mathfrak{A} \equiv \mathfrak{A} \supset \mathfrak{B} \qquad \text{detonates through } \textbf{implication}.$$

A separate tradition — theories of *truth* — had been stuck on that same hypothesis far longer, with a different fuse:

$$\lambda \equiv \text{“}\lambda \text{ is not true”} \qquad \text{detonates through } \textbf{negation}.$$

An aside — the same trap, from the other side

Curry's real hypothesis was not λ-abstraction. It was **self-referential definitions**:

$$\mathfrak{A} \equiv \mathfrak{A} \supset \mathfrak{B} \qquad \text{detonates through } \textbf{implication}.$$

A separate tradition — theories of *truth* — had been stuck on that same hypothesis far longer, with a different fuse:

$$\lambda \equiv \text{“}\lambda \text{ is not true”} \qquad \text{detonates through } \textbf{negation}.$$

Same hypothesis, different connective, **same collapse.**

And dropping LEM rescues neither — Curry's derivation contains *no negation at all*, and the Liar goes through intuitionistically too.

An aside — the same trap, from the other side

Curry's real hypothesis was not λ-abstraction. It was **self-referential definitions**:

$$\mathfrak{A} \equiv \mathfrak{A} \supset \mathfrak{B} \qquad \text{detonates through } \textbf{implication}.$$

A separate tradition — theories of *truth* — had been stuck on that same hypothesis far longer, with a different fuse:

$$\lambda \equiv \text{“}\lambda \text{ is not true”} \qquad \text{detonates through } \textbf{negation}.$$

Same hypothesis, different connective, **same collapse.**

And dropping LEM rescues neither — Curry's derivation contains *no negation at all*, and the Liar goes through intuitionistically too.

Two traditions, one wall.

The truth tradition hit it first — and made it a theorem.

The orthodox answer, both times: stratify

That theorem says the Liar is not an oddity to be stepped around. It is a proof that **the thing you wanted cannot exist**.

Tarski's undefinability theorem. No consistent theory interpreting enough arithmetic — **classical or intuitionistic** — can contain a predicate $T(x)$ of its own language satisfying

$$T(\ulcorner A \urcorner) \leftrightarrow A \qquad \text{for every sentence } A.$$

The orthodox answer, both times: stratify

That theorem says the Liar is not an oddity to be stepped around. It is a proof that **the thing you wanted cannot exist**.

Tarski's undefinability theorem. No consistent theory interpreting enough arithmetic — **classical or intuitionistic** — can contain a predicate $T(x)$ of its own language satisfying

$$T(\ulcorner A \urcorner) \;\leftrightarrow\; A \qquad \text{for every sentence } A.$$

Why. Gödel's diagonal lemma applied to $\neg T(x)$ yields λ with $\lambda \leftrightarrow \neg T(\ulcorner \lambda \urcorner)$; the schema at λ gives $T(\ulcorner \lambda \urcorner) \leftrightarrow \lambda$; chaining, $\lambda \leftrightarrow \neg\lambda$. No excluded middle is used.

The orthodox answer, both times: stratify

That theorem says the Liar is not an oddity to be stepped around. It is a proof that **the thing you wanted cannot exist**.

Tarski's undefinability theorem. No consistent theory interpreting enough arithmetic — **classical or intuitionistic** — can contain a predicate $T(x)$ of its own language satisfying

$$T(\ulcorner A \urcorner) \;\leftrightarrow\; A \qquad \text{for every sentence } A.$$

Why. Gödel's diagonal lemma applied to $\neg T(x)$ yields λ with $\lambda \leftrightarrow \neg T(\ulcorner \lambda \urcorner)$; the schema at λ gives $T(\ulcorner \lambda \urcorner) \leftrightarrow \lambda$; chaining, $\lambda \leftrightarrow \neg\lambda$. No excluded middle is used.

So truth for L is definable only in a *stronger metalanguage*: a hierarchy $L_0, L_1, L_2, \ldots$, each with true_n for the one below. Every sentence gets a **level, fixed in advance by its form**.

The orthodox answer, both times: stratify

That theorem says the Liar is not an oddity to be stepped around. It is a proof that **the thing you wanted cannot exist**.

Tarski's undefinability theorem. No consistent theory interpreting enough arithmetic — **classical or intuitionistic** — can contain a predicate $T(x)$ of its own language satisfying

$$T(\ulcorner A \urcorner) \;\leftrightarrow\; A \qquad \text{for every sentence } A.$$

Why. Gödel's diagonal lemma applied to $\neg T(x)$ yields λ with $\lambda \leftrightarrow \neg T(\ulcorner \lambda \urcorner)$; the schema at λ gives $T(\ulcorner \lambda \urcorner) \leftrightarrow \lambda$; chaining, $\lambda \leftrightarrow \neg\lambda$. No excluded middle is used.

So truth for L is definable only in a *stronger metalanguage*: a hierarchy $L_0, L_1, L_2, \ldots$, each with true_n for the one below. Every sentence gets a **level, fixed in advance by its form**.

Assign levels up front, and the paradox cannot be stated.

That is the type hierarchy's move too.

Why not just filter out the bad sentences?

Saul Kripke, *Outline of a Theory of Truth*, J. Phil. 72(19):690–716

Jones says **(1) "Most of Nixon's assertions about Watergate are false."** Ordinary, well-formed, says nothing about itself.

Why not just filter out the bad sentences?

Saul Kripke, *Outline of a Theory of Truth*, J. Phil. 72(19):690–716

Jones says **(1) "Most of Nixon's assertions about Watergate are false."** Ordinary, well-formed, says nothing about itself.

But suppose Nixon's Watergate claims split evenly true/false except for **(2) "Everything Jones says about Watergate is true"** — and (1) is Jones's only Watergate claim. **Now each is true if and only if it is false.**

Why not just filter out the bad sentences?

Saul Kripke, *Outline of a Theory of Truth*, J. Phil. 72(19):690–716

Jones says **(1) "Most of Nixon's assertions about Watergate are false."** Ordinary, well-formed, says nothing about itself.

But suppose Nixon's Watergate claims split evenly true/false except for **(2) "Everything Jones says about Watergate is true"** — and (1) is Jones's only Watergate claim. **Now each is true if and only if it is false.**

Same sentence, same syntax, same meanings. Only the *facts* moved.

Why not just filter out the bad sentences?

Saul Kripke, *Outline of a Theory of Truth*, J. Phil. 72(19):690–716

Jones says **(1) "Most of Nixon's assertions about Watergate are false."** Ordinary, well-formed, says nothing about itself.

But suppose Nixon's Watergate claims split evenly true/false except for **(2) "Everything Jones says about Watergate is true"** — and (1) is Jones's only Watergate claim. **Now each is true if and only if it is false.**

Same sentence, same syntax, same meanings. Only the *facts* moved.

A level cannot be fixed in advance: there is **no syntactic or semantic "sieve"** sorting the good from the bad (p. 692).

Kripke: forbid nothing. Let λ have no truth value.

Kripke's answer — build truth up from the ground

Add $T(x)$ to L, but let it be **partial**: true of some sentences, false of others, **undefined** on the rest. Only T is partial — the rest of L stays classical, interpreted once and for all.

Kripke's answer — build truth up from the ground

Add $T(x)$ to L, but let it be **partial**: true of some sentences, false of others, **undefined** on the rest. Only T is partial — the rest of L stays classical, interpreted once and for all.

Now climb. $\mathcal{L}_0$: T means *nothing at all*. $\mathcal{L}_{\alpha+1}$: T is true of exactly the sentences true in $\mathcal{L}_\alpha$, false of the false ones. At limits, take unions.

	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	
$A_0 \equiv$ "$0 = 0$" no T	**T**	**T**	**T**	level 0
$A_1 \equiv T(\ulcorner A_0 \urcorner)$	—	**T**	**T**	level 1
$A_2 \equiv T(\ulcorner A_1 \urcorner)$	—	—	**T**	level 2
$\tau \equiv T(\ulcorner \tau \urcorner)$ truth-teller	—	—	—	**never**
$\lambda \equiv \neg T(\ulcorner \lambda \urcorner)$ Liar	—	—	—	**never**

Kripke's answer — build truth up from the ground

Add $T(x)$ to L, but let it be **partial**: true of some sentences, false of others, **undefined** on the rest. Only T is partial — the rest of L stays classical, interpreted once and for all.

Now climb. $\mathcal{L}_0$: T means *nothing at all*. $\mathcal{L}_{\alpha+1}$: T is true of exactly the sentences true in $\mathcal{L}_\alpha$, false of the false ones. At limits, take unions.

	$\mathcal{L}_0$	$\mathcal{L}_1$	$\mathcal{L}_2$	
$A_0 \equiv$ "$0 = 0$" no T	**T**	**T**	**T**	level 0
$A_1 \equiv T(\ulcorner A_0 \urcorner)$	—	**T**	**T**	level 1
$A_2 \equiv T(\ulcorner A_1 \urcorner)$	—	—	**T**	level 2
$\tau \equiv T(\ulcorner \tau \urcorner)$ truth-teller	—	—	—	**never**
$\lambda \equiv \neg T(\ulcorner \lambda \urcorner)$ Liar	—	—	—	**never**

Deciding more never overturns a verdict, so the climb only grows — and it must stop, at a stage $\mathcal{L}_\sigma$ where nothing further changes. There T is true of exactly the truths of $\mathcal{L}_\sigma$ itself: **a language containing its own truth predicate**.

Grounded, ungrounded, paradoxical

grounded — gets a value somewhere in that climb

ungrounded — never does

paradoxical — never does, *whatever* you start T off believing

Grounded, ungrounded, paradoxical

grounded — gets a value somewhere in that climb

ungrounded — never does

paradoxical — never does, *whatever* you start T off believing

The Liar λ is **ungrounded *and* paradoxical** — stuck everywhere.

The truth-teller τ is **ungrounded but *not* paradoxical**: start instead with T true of τ, climb as before, and you land in a perfectly good stage where τ is true. Nothing breaks — the climb from nothing simply never had a *reason* to decide it.

Grounded, ungrounded, paradoxical

grounded — gets a value somewhere in that climb

ungrounded — never does

paradoxical — never does, *whatever* you start T off believing

The Liar λ is **ungrounded *and* paradoxical** — stuck everywhere.

The truth-teller τ is **ungrounded but *not* paradoxical**: start instead with T true of τ, climb as before, and you land in a perfectly good stage where τ is true. Nothing breaks — the climb from nothing simply never had a *reason* to decide it.

Ungrounded $\neq$ paradoxical.

The test is not "is it safe?" but "does it have a value?"

What GA takes from here

From Kripke

Undefined is not a third truth value — it is the **absence** of one.

Grounded = reachable from the bottom up, not "non-paradoxical".

Allowing sentences to have *no* value is what gets past Tarski.

GA adds

A proof system. Kripke gives a *model* — no inference rules, nothing to run.

Grounding by **computation** (step count), not a transfinite construction.

The condition as **preconditions on inference rules**.

What GA takes from here

From Kripke

Undefined is not a third truth value — it is the **absence** of one.

Grounded = reachable from the bottom up, not "non-paradoxical".

Allowing sentences to have *no* value is what gets past Tarski.

GA adds

A proof system. Kripke gives a *model* — no inference rules, nothing to run.

Grounding by **computation** (step count), not a transfinite construction.

The condition as **preconditions on inference rules**.

Climb from nothing until nothing changes — **this is Kleene iteration**, the same least-fixed-point recipe that gives a recursive definition its denotation.

"perhaps Scott and Kripke might well have formulated grounded deduction in the 1970s. . . " — *Have a thing?*, §6

Why bother

Classical or intuitionistic, the rule is the same: **prove a recursive definition terminates before you may use it.**

Every proposition must be true or false — so a computation that never returns has no value to *be*, and must be kept out of the logic.

Model the code

operational / denotational semantics

- reason at **arm's length**
- theorems about *encodings*, not programs
- Hoare · process calculi · separation
- each domain-specific: another language
- invariants, variants, frames: by hand

Type the code

Rocq, Lean, Agda — code *is* proof

- termination / strong normalization **first**
- strict positivity · universe stratification
- mutual recursion: bound into one construct
- **mostly no consistency proof anyway**

Why bother

Classical or intuitionistic, the rule is the same: **prove a recursive definition terminates before you may use it.**

Every proposition must be true or false — so a computation that never returns has no value to *be*, and must be kept out of the logic.

Model the code

operational / denotational semantics

- reason at **arm's length**
- theorems about *encodings*, not programs
- Hoare · process calculi · separation
- each domain-specific: another language
- invariants, variants, frames: by hand

Type the code

Rocq, Lean, Agda — code *is* proof

- termination / strong normalization **first**
- strict positivity · universe stratification
- mutual recursion: bound into one construct
- **mostly no consistency proof anyway**

Kripke's exit — **let it have no value** — is a *model*: no rules, nothing to run.

And to our knowledge no paracomplete *formal system* in that tradition has been shown strong enough to reason about Turing-complete computation — even just over $\mathbb{N}$, as in LCF.

Grounded Deduction

Habeas quid — "have a thing"

You may not reason with a term until you have shown it *has a value*.

$$\frac{\Gamma \vdash p\ \mathsf{B} \qquad \Gamma \cup \{p\} \vdash q}{\Gamma \vdash p \to q}\ (\to I) \qquad \frac{\Gamma \vdash p\ \mathsf{B} \quad \Gamma \cup \{\neg p\} \vdash q \quad \Gamma \cup \{\neg p\} \vdash \neg q}{\Gamma \vdash p}$$

$p\ \mathsf{B} \equiv p \vee \neg p$ — classically a tautology, so the preconditions are free and the rules collapse to the familiar ones.

In GA it is a **dynamic type check**: a demand that p terminate with a boolean.

In Curry's terms: GA satisfies hypotheses **1, 2 and 4**.
It breaks **3**.

Both paradoxes die of malnutrition

Liar $L \equiv \neg L$

Admitted as a definition.

Refutation by contradiction needs L B, which needs the refutation.

No value. Nothing to explode from.

Curry $C \equiv C \to P$

Admitted as a definition.

Step (3) now needs C B first, obtainable only via the implication we are trying to introduce.

Circular obligation. No proof.

The paradoxes are **expressible** and simply not **provable**.

The restriction is not on what you may write

Type theory's mechanism is **grammatical**: make $x\,x$ ill-formed, and the paradox cannot be stated.

The restriction is not on what you may write

Type theory's mechanism is **grammatical**: make $x\,x$ ill-formed, and the paradox cannot be stated.

GA does the opposite. All of these are **well-formed**:

$$f[f] \qquad L \equiv \neg L \qquad C \equiv C \to P \qquad \texttt{collatz}(n)$$

The restriction is not on what you may write

Type theory's mechanism is **grammatical**: make $x\,x$ ill-formed, and the paradox cannot be stated.

GA does the opposite. All of these are **well-formed**:

$$f[f] \qquad L \equiv \neg L \qquad C \equiv C \to P \qquad \texttt{collatz}(n)$$

The restriction is on what you may **infer**.

Which is why the paradoxes are *expressible* and merely not *provable* — and why a definition never needs a termination proof to be admitted.

Grounded versus classical inference

$$\frac{p \vdash q \qquad p \vdash \neg q}{\neg p}\,\neg I \qquad \frac{\neg\neg p}{p}\,\neg\neg E$$

$$\frac{p \vdash q}{p \to q}\,\to I \qquad \frac{p \to q \qquad p}{q}\,\to E$$

$$\frac{}{a = a}\,{=}R \qquad \frac{a = b}{b = a}\,{=}S \qquad \frac{a = b \qquad b = c}{a = c}\,{=}T$$

$$\frac{}{\mathrm{S}(a) \neq \mathbf{0}}\,\mathrm{S}{\neq}\mathbf{0}I$$

$$\frac{p\langle \mathbf{0} \rangle \qquad p\langle x \rangle \vdash p\langle \mathrm{S}(x) \rangle}{p\langle a \rangle}\,Ind$$

(a) Classical inference rules

$$\frac{p}{\neg\neg p}\,\neg\neg IE \qquad \frac{p \qquad \neg p}{q}\,\neg E$$

$$\frac{p\ \mathsf{B} \qquad p \vdash q}{p \to q}\,\to I \qquad \frac{p \to q \qquad p}{q}\,\to E$$

$$\frac{a\ \mathsf{N}}{a = a}\,{=}R \qquad \frac{a = b}{b = a}\,{=}S \qquad \frac{a = b \qquad b = c}{a = c}\,{=}T$$

$$\frac{}{\mathbf{0}\ \mathsf{N}}\,\mathbf{0}TI \qquad \frac{a\ \mathsf{N}}{\mathrm{S}(a)\ \mathsf{N}}\,\mathrm{S}TIE \qquad \frac{a\ \mathsf{N}}{\mathrm{S}(a) \neq \mathbf{0}}\,\mathrm{S}{\neq}\mathbf{0}I$$

$$\frac{p\langle \mathbf{0} \rangle \qquad x\ \mathsf{N}, p\langle x \rangle \vdash p\langle \mathrm{S}(x) \rangle \qquad a\ \mathsf{N}}{p\langle a \rangle}\,Ind$$

(b) Grounded inference rules

Same rules, same shape — with a **habeas quid** premise wherever the classical rule quietly assumed the term had a value.

Termination proofs *are* typing proofs

A static type system proves **safety**: a well-typed term never yields a value of the wrong type. It says nothing about **liveness** — whether a value ever arrives.

Termination proofs *are* typing proofs

A static type system proves **safety**: a well-typed term never yields a value of the wrong type. It says nothing about **liveness** — whether a value ever arrives.

GA's dynamic typing proves exactly the liveness property, and it does so **by running the computation backwards**.

$$\frac{}{0\ \mathrm{N}}\,0\,TI \quad \Rightarrow \quad \frac{0\ \mathrm{N}}{S(0)\ \mathrm{N}}\,STIE \quad \Rightarrow \quad \frac{S(0)\ \mathrm{N}}{S(S(0))\ \mathrm{N}}\,STIE \quad \cdots$$

Each proof says: **this term is a terminating computation yielding a number.**

Longer proofs are built from shorter ones — reverse symbolic execution, one reduction step at a time.

From pointwise to uniform: induction

Those proofs are **pointwise** — one per number. Induction makes them uniform.

$$\texttt{even}(a) \;\equiv\; a = 0 \;?\; 1 : 1 - \texttt{even}(P(a))$$

Admitted with **no obligation**. To *use* it we prove $a\,\mathsf{N} \vdash \texttt{even}(a)\,\mathsf{N}$ by induction on a:

base case, the conditional takes the first branch, reducing to $1\,\mathsf{N}$;
step, assume $\texttt{even}(x)\,\mathsf{N}$, apply the definition, $S{\neq}0I$, the predecessor rules, and the prior proofs.

From pointwise to uniform: induction

Those proofs are **pointwise** — one per number. Induction makes them uniform.

$$\texttt{even}(a) \;\equiv\; a = 0 \;?\; 1 : 1 - \texttt{even}(P(a))$$

Admitted with **no obligation**. To *use* it we prove $a\,\mathsf{N} \vdash \texttt{even}(a)\,\mathsf{N}$ by induction on a:
base case, the conditional takes the first branch, reducing to $1\,\mathsf{N}$;
step, assume $\texttt{even}(x)\,\mathsf{N}$, apply the definition, $S{\neq}0I$, the predecessor rules, and the prior proofs.

Every primitive-recursive function is handled this way — follow the composition structure, prove the simpler pieces first, use induction at each primitive recursion.

And general recursion — Turing completeness

Kleene normal form: any general-recursive $f(x)$ = a **primitive-recursive step function** + one **unbounded search**.

$T(x, i)$ returns $1 + y$ if the machine on x halts within i steps with output y, and 0 otherwise — primitive recursive.

$$F(x, i) \;\equiv\; T(x, i) \neq 0 \;?\; P(T(x, i)) \;:\; F(x, S(i))$$

Then $F(x, 0)$ is the computation's result if it has one.

And general recursion — Turing completeness

Kleene normal form: any general-recursive $f(x)$ = a **primitive-recursive step function** + one **unbounded search**.

$T(x, i)$ returns $1 + y$ if the machine on x halts within i steps with output y, and 0 otherwise — primitive recursive.

$$F(x, i) \;\equiv\; T(x, i) \neq 0 \;?\; P(T(x, i)) \;:\; F(x, S(i))$$

Then $F(x, 0)$ is the computation's result if it has one.

That second definition is the whole difference.

No classical or intuitionistic system will admit it without a proof it terminates — and it does not, in general.

And general recursion — Turing completeness

Kleene normal form: any general-recursive $f(x)$ = a **primitive-recursive step function** + one **unbounded search**.

$T(x, i)$ returns $1 + y$ if the machine on x halts within i steps with output y, and 0 otherwise — primitive recursive.

$$F(x, i) \;\equiv\; T(x, i) \neq 0 \;?\; P(T(x, i)) \;:\; F(x, S(i))$$

Then $F(x, 0)$ is the computation's result if it has one.

That second definition is the whole difference.

No classical or intuitionistic system will admit it without a proof it terminates — and it does not, in general.

GA admits it, and for every x on which it *does* terminate with result y, GA proves $F(x, 0) = y$ — by executing it in reverse.

Where it diverges, GA proves nothing. **No inconsistency; just no theorem.**

Seven papers

2024	GD	the idea, and the propositional core
2025	BGA / PGA	machine-checked metatheory
2026	Paradoxes in GA	the three paradoxes, mechanised as nontermination
2026	RGA	quantifiers, grounded reflectively
2026	Internalized truth	its own truth predicate, full language
2026	OGA	one more rule — incompleteness becomes *priced*
2026	Grounded set theory	a graded ZF
now	**Isabelle/Pure**	**can anyone actually use it?**

1 · *Reasoning Around Paradox with Grounded Deduction*

Ford, arXiv:2409.08243 (2024)

- Kripke-inspired: let ungrounded expressions denote nothing
- *habeas quid* as *preconditions on inference rules* — the central move
- An impossibility triangle:
  consistency · unrestricted LEM · unrestricted recursive definitions
- Propositional GD, predicate logic, arithmetic; a computational interpretation and a denotational semantics

Preliminary — no machine-checked metatheory.

2 · *Have a thing?* — BGA and PGA

arXiv:2510.25369

Two concrete systems, everything checked in Isabelle/HOL.

BGA — minimal kernel

syntactic consistency
semantic soundness
semantic completeness
represents every general-recursive function
the missing connectives, as proof searches
its own truth predicate (grounded sentences)

PGA — expressive layer

primitive propositional operators
higher-order functions via SKI
consistency, completeness
reduces to BGA

16 of Copi's 19 classical rules survive *unmodified*

Proof-theoretic strength: somewhere in $[\omega^\omega,\ \omega_1^{\mathrm{CK}}]$ — **open**.

Why every line of this is machine-checked

Everyone we have talked about so far was **sure** their system was consistent.

- **Frege** — a completed foundation, until Russell's letter
- **Church 1932** — consistency "we cannot now answer except by *conjecture*"; proposed to find out empirically. **Kleene–Rosser, 1935.**
- **Curry** — consistency proved for the combinator half; the logical half left for later. **Same paper.**
- **Church 1933** — deleted the four postulates he blamed. **Still inconsistent.**
- **Martin-Löf**, `Type : Type`. **Girard, 1972.**

Why every line of this is machine-checked

Everyone we have talked about so far was **sure** their system was consistent.

- **Frege** — a completed foundation, until Russell's letter
- **Church 1932** — consistency "we cannot now answer except by *conjecture*"; proposed to find out empirically. **Kleene–Rosser, 1935.**
- **Curry** — consistency proved for the combinator half; the logical half left for later. **Same paper.**
- **Church 1933** — deleted the four postulates he blamed. **Still inconsistent.**
- **Martin-Löf**, `Type : Type`. **Girard, 1972.**

In this subject, **a plausible argument on paper has a track record.**

So: **∼49,000 lines of Isabelle/HOL**. Every theorem in the rest of this section is a machine-checked one.

BGA — strip it to the kernel

$$t \equiv v \mid \mathbf{0} \mid \mathbf{S}\, t \mid \mathbf{P}\, t \mid t\ 0?\ t : t \mid d(t,t)$$

(a) Abstract term syntax of BGA

$$f \equiv t = t \mid t \neq t$$

(b) Abstract formula syntax of BGA

- **No logical connectives at all.** A formula is $t = t$ or $t \neq t$, and $\neq$ is *primitive* — not $\neg(=)$. There is no $\neg$ to define it with.
- Functions come from a background list Δ of **2-argument recursive definitions**, $d_i(v_0, v_1) \equiv b_i$. No define-before-use rule, so definitions may be singly or mutually recursive, in any tangle.
- An out-of-range d_i simply never reduces — **a nonterminating computation**, not an error. $P(0)$ likewise has no value.
- So BGA is not one system but a **family**, one per Δ.

The result is strikingly close to **PCF** — with a definition list where PCF has typed lambda terms. An earlier BGA had primitive $\neg$ and $\vee$; they turned out to be unnecessary, and the completeness proof got simpler without them.

BGA's inference rules

$$\frac{a = b \qquad p\langle a, \ldots\rangle}{p\langle b, \ldots\rangle} =E \qquad \frac{a = b}{b = a} =S \qquad \frac{a \neq b}{b \neq a} \neq S$$

$$\frac{}{\mathbf{0}\ \mathsf{N}}\,\mathbf{0}I \qquad \frac{a\ \mathsf{N}}{\mathrm{S}(a) \neq \mathbf{0}}\,\mathrm{S}\neq\mathbf{0}I \qquad \frac{a = b}{\mathrm{S}(a) = \mathrm{S}(b)}\,\mathrm{S}{=}IE \qquad \frac{a \neq b}{\mathrm{S}(a) \neq \mathrm{S}(b)}\,\mathrm{S}{\neq}IE \qquad \frac{a\ \mathsf{N}}{\mathbf{P}(\mathrm{S}(a)) = a}\,\mathbf{P}I$$

$$\frac{p\langle \mathbf{0}, \ldots\rangle \qquad x\ \mathsf{N}, p\langle x, \ldots\rangle \vdash p\langle \mathrm{S}(x), \ldots\rangle \qquad a\ \mathsf{N}}{p\langle a, \ldots\rangle}\,Ind$$

$$\frac{c = \mathbf{0} \qquad a\ \mathsf{N}}{(c\ 0?\ a : b) = a}\,?I1 \qquad \frac{c \neq \mathbf{0} \qquad b\ \mathsf{N}}{(c\ 0?\ a : b) = b}\,?I2$$

$$\frac{s(v_0, v_1) \equiv b\langle v_0, v_1\rangle \qquad a_0\ \mathsf{N} \qquad a_1\ \mathsf{N} \qquad p\langle b\langle a_0, a_1\rangle, \ldots\rangle}{p\langle s(a_0, a_1), \ldots\rangle} \equiv I$$

Almost all *introduction* rules; the only real elimination is $= E$, substitution of equals for equals.

$\equiv I$ is where computation enters: to use definition $s(v_0, v_1) \equiv b\langle v_0, v_1\rangle$ you must first have a_0 N and a_1 N — **call-by-value, as an inference rule.**

BGA's operational semantics

A **big-step structural operational semantics**: a set of inference rules in which $a \Downarrow n$ reads "term a evaluates to the value n".

Term reductions

$$\frac{A_j = n}{\mathbf{v}_j \Downarrow n} \qquad \frac{}{\mathbf{0} \Downarrow 0} \qquad \frac{a \Downarrow n}{\mathbf{S}(a) \Downarrow n+1} \qquad \frac{a \Downarrow n+1}{\mathbf{P}(a) \Downarrow n} \qquad \frac{c \Downarrow 0 \qquad a \Downarrow n}{c\ 0?\ a : b \Downarrow n} \qquad \frac{c \Downarrow 1+m \qquad b \Downarrow n}{c\ 0?\ a : b \Downarrow n}$$

$$\frac{s(v_0, v_1) \equiv b\langle v_0, v_1\rangle \qquad a_0 \Downarrow n_0 \qquad a_1 \Downarrow n_1 \qquad b\langle a_0, a_1\rangle \Downarrow m}{s(a_0, a_1) \Downarrow m}$$

Formula reductions

$$\frac{a \Downarrow n \qquad b \Downarrow n}{a = b \Downarrow 1} \qquad \frac{a \Downarrow n \qquad b \Downarrow m \qquad n \neq m}{a = b \Downarrow 0} \qquad \frac{a \Downarrow n \qquad b \Downarrow n}{a \neq b \Downarrow 0} \qquad \frac{a \Downarrow n \qquad b \Downarrow m \qquad n \neq m}{a \neq b \Downarrow 1}$$

BGA's operational semantics

A **big-step structural operational semantics**: a set of inference rules in which $a \Downarrow n$ reads "term a evaluates to the value n".

Term reductions

$$\frac{A_j = n}{\mathbf{v}_j \Downarrow n} \qquad \frac{}{\mathbf{0} \Downarrow 0} \qquad \frac{a \Downarrow n}{\mathbf{S}(a) \Downarrow n+1} \qquad \frac{a \Downarrow n+1}{\mathbf{P}(a) \Downarrow n} \qquad \frac{c \Downarrow 0 \qquad a \Downarrow n}{c\ 0?\ a : b \Downarrow n} \qquad \frac{c \Downarrow 1+m \qquad b \Downarrow n}{c\ 0?\ a : b \Downarrow n}$$

$$\frac{s(v_0, v_1) \equiv b\langle v_0, v_1\rangle \qquad a_0 \Downarrow n_0 \qquad a_1 \Downarrow n_1 \qquad b\langle a_0, a_1\rangle \Downarrow m}{s(a_0, a_1) \Downarrow m}$$

Formula reductions

$$\frac{a \Downarrow n \qquad b \Downarrow n}{a = b \Downarrow 1} \qquad \frac{a \Downarrow n \qquad b \Downarrow m \qquad n \neq m}{a = b \Downarrow 0} \qquad \frac{a \Downarrow n \qquad b \Downarrow n}{a \neq b \Downarrow 0} \qquad \frac{a \Downarrow n \qquad b \Downarrow m \qquad n \neq m}{a \neq b \Downarrow 1}$$

- $a \Downarrow n$ holds iff there is a *finite derivation tree* for it. So **nontermination is the absence of a derivation** — no $\bot$ appears in the rules, and partiality costs nothing.
- The HOL version carries a **step count**. That is what the proofs induct over.

Result 1 — soundness, and hence consistency

The classical method: show every inference rule **preserves truth**. The twist is what "truth" means here.

A **satisfies** φ iff $\varphi \Downarrow 1$ under A.

$\Gamma \vdash \varphi$ is **truth-preserving** iff every A satisfying all of Γ satisfies φ.

Result 1 — soundness, and hence consistency

The classical method: show every inference rule **preserves truth**. The twist is what "truth" means here.

A **satisfies** φ iff $\varphi \Downarrow 1$ under A.

$\Gamma \vdash \varphi$ is **truth-preserving** iff every A satisfying all of Γ satisfies φ.

- **Lemma.** Every rule in the table is truth-preserving.
- **Theorem (semantic soundness).** Every theorem of BGA is true in that model.
- **Theorem (syntactic consistency).** There are no a, b with both $\vdash a = b$ and $\vdash a \neq b$.

Consistency is normally stated with negation; BGA has none, so the equals/unequals pair does the work — and parts of the development need a metalogical notion of **refutation** dual to entailment.

Result 2 — semantic completeness

Lemma. If term a reduces to n in k steps under all assignments, then $\vdash a = \overline{n}$ is provable in BGA.

Proof: induction on step count, then case analysis over the reduction rules.

Result 2 — semantic completeness

Lemma. If term a reduces to n in k steps under all assignments, then $\vdash a = \overline{n}$ is provable in BGA.

Proof: induction on step count, then case analysis over the reduction rules.

That is: **turn the derivation tree of the semantics into a proof tree of the system.**

Longer proofs built from shorter ones — progressively longer reverse symbolic executions.

Result 2 — semantic completeness

Lemma. If term a reduces to n in k steps under all assignments, then $\vdash a = \overline{n}$ is provable in BGA.

Proof: induction on step count, then case analysis over the reduction rules.

That is: **turn the derivation tree of the semantics into a proof tree of the system.**

Longer proofs built from shorter ones — progressively longer reverse symbolic executions.

Theorem (semantic completeness). Every semantically true BGA formula is provable, *unconditionally*.

Result 2 — semantic completeness

Lemma. If term a reduces to n in k steps under all assignments, then $\vdash a = \overline{n}$ is provable in BGA.

Proof: induction on step count, then case analysis over the reduction rules.

That is: **turn the derivation tree of the semantics into a proof tree of the system.**

Longer proofs built from shorter ones — progressively longer reverse symbolic executions.

Theorem (semantic completeness). Every semantically true BGA formula is provable, *unconditionally*.

Why the model had to be an operational one: we can induct over reductions. Over HOL's own $\mathbb{N}$ there would be nothing to induct on.

Consistent *and* complete *and* arithmetic?

- Gödel I concerns **syntactic** completeness — for every f, $\vdash f$ or $\vdash \neg f$.
  BGA is emphatically not that, and cannot be: LEM is built into the definition.
- What is proved is **semantic** completeness: every formula true in the model is provable.
- Gödel's argument assumes a classical *object* logic. **GA is not one.**

Result 3 — BGA does arithmetic

BGA has no arithmetic built in. Every function lives in Δ, the background list of recursive definitions — so the question is whether **some Δ gets the answers right**.

f is **represented** when BGA proves $\hat{f}\langle \overline{a}, \overline{b} \rangle = \overline{f(a,b)}$ for every a, b — and, by consistency, never proves a *wrong* answer.

Result 3 — BGA does arithmetic

BGA has no arithmetic built in. Every function lives in Δ, the background list of recursive definitions — so the question is whether **some Δ gets the answers right**.

f is **represented** when BGA proves $\hat{f}\langle\overline{a},\overline{b}\rangle = \overline{f(a,b)}$ for every a, b — and, by consistency, never proves a *wrong* answer.

- **Every primitive-recursive function** is representable, and BGA proves it total: $a\,\mathsf{N},\ b\,\mathsf{N} \vdash f(a,b)\,\mathsf{N}$.
- **Every general-recursive function** is representable: take that Δ and **append one definition**, the unbounded search.

Result 3 — BGA does arithmetic

BGA has no arithmetic built in. Every function lives in Δ, the background list of recursive definitions — so the question is whether **some Δ gets the answers right**.

f is **represented** when BGA proves $\hat{f}\langle\overline{a},\overline{b}\rangle = \overline{f(a,b)}$ for every a, b — and, by consistency, never proves a *wrong* answer.

- **Every primitive-recursive function** is representable, and BGA proves it total: $a\,\mathrm{N},\ b\,\mathrm{N} \vdash f(a,b)\,\mathrm{N}$.
- **Every general-recursive function** is representable: take that Δ and **append one definition**, the unbounded search.

That one definition is the entire gap between the two classes — and it is exactly the one no other system will admit.

Where it terminates, BGA proves the result. Where it does not, BGA proves nothing.

One system, and it can see itself

First, one system instead of a family: write an **interpreter for BGA, in BGA** — a term $E(e, x)$ running the function coded by e on input x. A single fixed Δ_U then represents *every* computable function.

The universal-Turing-machine move: pass the function as data rather than building a new system per function.

One system, and it can see itself

First, one system instead of a family: write an **interpreter for BGA, in BGA** — a term $E(e, x)$ running the function coded by e on input x. A single fixed Δ_U then represents *every* computable function.

The universal-Turing-machine move: pass the function as data rather than building a new system per function.

Then **arithmetization**: BGA's own syntax and proof rules are represented in Δ_U too — so **a BGA term can search for BGA proofs**.

One system, and it can see itself

First, one system instead of a family: write an **interpreter for BGA, in BGA** — a term $E(e, x)$ running the function coded by e on input x. A single fixed Δ_U then represents *every* computable function.

The universal-Turing-machine move: pass the function as data rather than building a new system per function.

Then **arithmetization**: BGA's own syntax and proof rules are represented in Δ_U too — so **a BGA term can search for BGA proofs**.

Which is enough to recover the connectives BGA does not have.

Result 4 — the connectives were computations all along

- **Negation.** $\neg\varphi$ is a term that takes $\ulcorner\varphi\urcorner$ and *searches* for a BGA proof or a refutation of φ, returning the opposite. (A refutation of $a = b$ is just a proof of $a \neq b$.)
- **Disjunction.** $\varphi_1 \vee \varphi_2$ simulates a *parallel* search for: a proof of φ_1, or a proof of φ_2, or refutations of both.
  Hence $1 \vee \bot = 1$ — Kleene's strong 3-valued semantics, obtained rather than stipulated.
- Conjunction, implication, biconditional then follow by de Morgan, as usual.

Result 4 — the connectives were computations all along

- **Negation.** $\neg\varphi$ is a term that takes $\ulcorner\varphi\urcorner$ and *searches* for a BGA proof or a refutation of φ, returning the opposite. (A refutation of $a = b$ is just a proof of $a \neq b$.)
- **Disjunction.** $\varphi_1 \vee \varphi_2$ simulates a *parallel* search for: a proof of φ_1, or a proof of φ_2, or refutations of both.
  Hence $1 \vee \bot = 1$ — Kleene's strong 3-valued semantics, obtained rather than stipulated.
- Conjunction, implication, biconditional then follow by de Morgan, as usual.

Undefined is not a third truth value bolted on.

It is a proof search that never returns. Kripke's "no value", as an actual computation.

Result 5 — BGA contains its own truth predicate

There is a term $\mathcal{T}(x)$ in Δ_U such that for every grounded sentence φ with $\vdash \varphi$ B,

$$\vdash \mathcal{T}(\ulcorner\varphi\urcorner) \leftrightarrow \varphi.$$

Proof: unfold $\leftrightarrow$ into the reflective $\neg$ and $\vee$, then apply arithmetization and semantic completeness.

Result 5 — BGA contains its own truth predicate

There is a term $\mathcal{T}(x)$ in Δ_U such that for every grounded sentence φ with $\vdash \varphi$ B,

$$\vdash \mathcal{T}(\ulcorner\varphi\urcorner) \leftrightarrow \varphi.$$

Proof: unfold $\leftrightarrow$ into the reflective $\neg$ and $\vee$, then apply arithmetization and semantic completeness.

Why Tarski does not kill this. His theorem assumes a consistent *classical* system, in which every sentence has a value, so the T-schema must hold at the Liar too.

In BGA the biconditional is *itself* grounded — provable only when both sides are decided. At the Liar, the schema is simply **not provable**, and nothing follows.

Result 5 — BGA contains its own truth predicate

There is a term $\mathcal{T}(x)$ in Δ_U such that for every grounded sentence φ with $\vdash \varphi$ B,

$$\vdash \mathcal{T}(\ulcorner\varphi\urcorner) \leftrightarrow \varphi.$$

Proof: unfold $\leftrightarrow$ into the reflective $\neg$ and $\vee$, then apply arithmetization and semantic completeness.

Why Tarski does not kill this. His theorem assumes a consistent *classical* system, in which every sentence has a value, so the T-schema must hold at the Liar too.

In BGA the biconditional is *itself* grounded — provable only when both sides are decided. At the Liar, the schema is simply **not provable**, and nothing follows.

This is Kripke's fixed point — **now inside a system with arithmetic, Turing-complete computation, and inference rules.**

PGA — putting the logic back on the surface

BGA *has* the connectives, but only as opaque reflective searches. You cannot **reason** with them — there are no rules for $\vee$, only a computation.

PGA — putting the logic back on the surface

BGA *has* the connectives, but only as opaque reflective searches. You cannot **reason** with them — there are no rules for $\vee$, only a computation.
PGA makes them **primitive**, with inference rules, and adds what a user would want:

- **No term/formula distinction.** Everything is a term; a "formula" is just a term computing to 0 or 1. $\bot$ becomes a primitive constant.
- The type tests are still **not primitives**: $a\,\mathrm{N} \equiv (a = a)$, $p\,\mathrm{B} \equiv p \vee \neg p$, $a \neq b \equiv \neg(a = b)$ — all shorthands.
- **Higher-order functions** via a primitive application $f[a]$ and the SKI combinators (plus $\mathcal{R}$ from Gödel's System T, and a successor combinator s). No λ or `letrec` binders — combinator conversion instead.
- $P(0) = 0$ in PGA, unlike BGA. Deliberate: some typing rules would be unsound otherwise.

Still **no quantifiers** — only implicit top-level universal quantification over free variables, as in PRA. (Those arrive in RGA, next.)

PGA's semantics — informally

$$\frac{A_j = n}{\mathbf{v}_j \Downarrow n} \qquad \frac{}{\mathbf{0} \Downarrow 0} \qquad \frac{a \Downarrow n}{\mathbf{S}(a) \Downarrow n+1} \qquad \frac{a \Downarrow 0}{\mathbf{P}(a) \Downarrow 0} \qquad \frac{a \Downarrow n+1}{\mathbf{P}(a) \Downarrow n} \qquad \frac{c \Downarrow 1 \qquad a \Downarrow n}{c \mathbin{?} a : b \Downarrow n} \qquad \frac{c \Downarrow 0 \qquad b \Downarrow n}{c \mathbin{?} a : b \Downarrow n}$$

$$\frac{a \Downarrow n}{\mathcal{I}[a] \Downarrow n} \qquad \frac{a \Downarrow n \qquad b \Downarrow m}{\mathcal{K}[a][b] \Downarrow n} \qquad \frac{a[c][b[c]] \Downarrow n}{\mathcal{S}[a][b][c] \Downarrow n}$$

$$\frac{a \Downarrow n \qquad b \Downarrow n}{a = b \Downarrow 1} \qquad \frac{a \Downarrow n \qquad b \Downarrow m \qquad n \neq m}{a = b \Downarrow 0}$$

$$\frac{a \Downarrow 1}{\neg a \Downarrow 0} \qquad \frac{a \Downarrow 0}{\neg a \Downarrow 1} \qquad \frac{a \Downarrow 1}{a \vee b \Downarrow 1} \qquad \frac{b \Downarrow 1}{a \vee b \Downarrow 1} \qquad \frac{a \Downarrow 0 \qquad b \Downarrow 0}{a \vee b \Downarrow 0}$$

A reading aid, not the definition — the real one is the reduction to BGA, next. Evaluation is strict, with two exceptions, and both exceptions are the point:

- **Conditional**: the not-taken branch need never evaluate to anything.
- **Disjunction**: $a \Downarrow 1$ gives $a \vee b \Downarrow 1$ *with no premise about* b *at all*.

Operators fed non-boolean inputs simply have no rule, hence no value. **$\perp$ is never produced; it is what is left when no rule applies.**

Where *habeas quid* actually appears

Logical negation (NOT)

$$\frac{p}{\neg\neg p}\,\neg\neg IE \qquad \frac{p \quad \neg p}{q}\,\neg E \qquad \frac{p \vee \neg p}{p\ \mathsf{B}}\,boolIE$$

$$\frac{p\ \mathsf{B} \quad p \vdash q \quad p \vdash \neg q}{\neg p}\,boolE1 \qquad \frac{p\ \mathsf{B} \quad \neg p \vdash q \quad \neg p \vdash \neg q}{p}\,boolE2$$

Material implication

$$\frac{p\ \mathsf{B} \quad p \vdash q}{p \to q} \to I \qquad \frac{p \to q \quad p}{q} \to E \qquad \frac{\neg p \vee q}{p \to q} \to IE$$

Material biconditional (if-and-only-if)

$$\frac{p\ \mathsf{B} \quad q\ \mathsf{B} \quad p \vdash q \quad q \vdash p}{p \leftrightarrow q} \leftrightarrow I \qquad \frac{p \leftrightarrow q \quad p}{q} \leftrightarrow E1 \qquad \frac{p \leftrightarrow q \quad q}{p} \leftrightarrow E2 \qquad \frac{(p \to q) \wedge (q \to p)}{p \leftrightarrow q} \leftrightarrow IE$$

Four rules carry a B: refutation, proof by contradiction, $\to I$, and $\leftrightarrow I$ (which needs *two*).

Everything else — $\vee$, $\wedge$, modus ponens, de Morgan, double negation — is **the classical rule, unchanged**.

Many of PGA's rules are deliberately redundant: expressiveness over minimality, the opposite of BGA's brief.

PGA is BGA in better clothes

PGA's semantics is not given afresh. Every well-formed PGA term is compiled to a BGA term by a **primitive-recursive** translation $\rho(\cdot)$, interpreted in Δ_U:

$\bot$	$\mapsto$ $P(0)$ — undefined in BGA, so nonterminating
0, $S(a)$	$\mapsto$ the corresponding BGA constructs, directly
$P(a)$	$\mapsto$ a primitive-recursive *minus one*, so $P(0) = 0$
$c\,?\,a : b$	$\mapsto$ two chained if-zero tests, diverging on non-booleans
$f[a]$	$\mapsto$ the general-recursive apply: read f as a step-function code
$\neg$, $\vee$	$\mapsto$ **the reflective proof searches of Result 4**

PGA is BGA in better clothes

PGA's semantics is not given afresh. Every well-formed PGA term is compiled to a BGA term by a **primitive-recursive** translation $\rho(\cdot)$, interpreted in Δ_U:

$\bot$	$\mapsto$ $P(0)$ — undefined in BGA, so nonterminating
0, $S(a)$	$\mapsto$ the corresponding BGA constructs, directly
$P(a)$	$\mapsto$ a primitive-recursive *minus one*, so $P(0) = 0$
$c\,?\,a : b$	$\mapsto$ two chained if-zero tests, diverging on non-booleans
$f[a]$	$\mapsto$ the general-recursive apply: read f as a step-function code
$\neg$, $\vee$	$\mapsto$ **the reflective proof searches of Result 4**

So PGA is not a new system to be trusted — it is a **view of BGA**, and it inherits BGA's model.

PGA's results

- **Semantic consistency** and **semantic completeness** — both by pushing through the reduction to BGA.
- **Functional completeness.** $f[a]$ reads f as the Gödel code of an arbitrary primitive-recursive step function, so **every** general-recursive function is applicable — whatever f is, however it was computed, whether or not it halts on a.
- **Primitive-recursive totality** is provable, as in BGA, via the combinators.
- **Proof-theoretic ordinal unknown** — somewhere in $[\omega^\omega$ (PRA), $\omega_1^{\mathrm{CK}}]$.

So — is this the “type-free paradise”?

Church, Curry and Scott were all after a “Fregean paradise of *type-free* functions”.

So — is this the “type-free paradise”?

Church, Curry and Scott were all after a “Fregean paradise of *type-free* functions”.

Not literally. GA’s *habeas quid* conditions are plainly typing rules — dynamic ones.

So — is this the "type-free paradise"?

Church, Curry and Scott were all after a "Fregean paradise of *type-free* functions".

Not literally. GA's *habeas quid* conditions are plainly typing rules — dynamic ones. But the thing that word was ever really about is **Russellian stratification**: levels assigned in advance, a collection containing only things below it. That is still what keeps modern dependent type theories consistent — the universe hierarchy, and Girard's paradox waiting if you collapse it.

GA has no levels at all.

Types and their properties live in exactly the same syntactic and semantic space as everything else. One closed universe.

And the type predicates are not even primitive: $a\,\mathsf{N} \equiv (a = a)$ $\quad$ $p\,\mathsf{B} \equiv p \vee \neg p$

Unfold them, and the "type system" disappears into the term language.

So — is this the "type-free paradise"?

Church, Curry and Scott were all after a "Fregean paradise of *type-free* functions".
Not literally. GA's *habeas quid* conditions are plainly typing rules — dynamic ones. But the thing that word was ever really about is **Russellian stratification**: levels assigned in advance, a collection containing only things below it. That is still what keeps modern dependent type theories consistent — the universe hierarchy, and Girard's paradox waiting if you collapse it.

GA has no levels at all.

Types and their properties live in exactly the same syntactic and semantic space as everything else. One closed universe.

And the type predicates are not even primitive: $a\,\mathsf{N} \equiv (a = a)$ $p\,\mathsf{B} \equiv p \vee \neg p$

Unfold them, and the "type system" disappears into the term language.

Free of stratification, then. **Whether it is a *paradise*** —
"only time, experience, and further exploration will tell."

Usability of GA

What was built

Not a prover from scratch. GA is axiomatized as an **object logic on Isabelle/Pure** — the way Isabelle/HOL and Isabelle/ZF are.

Isabelle supplies the kernel, Isar, the simplifier. We supply the logic.

What was built

Not a prover from scratch. GA is axiomatized as an **object logic on Isabelle/Pure** — the way Isabelle/HOL and Isabelle/ZF are.

Isabelle supplies the kernel, Isar, the simplifier. We supply the logic.

Two Pure types carry it: `o` for GA formulas, `num` for GA terms, joined by `Trueprop :: o ⟹ prop`.

What was built

Not a prover from scratch. GA is axiomatized as an **object logic on Isabelle/Pure** — the way Isabelle/HOL and Isabelle/ZF are.

Isabelle supplies the kernel, Isar, the simplifier. We supply the logic.

Two Pure types carry it: `o` for GA formulas, `num` for GA terms, joined by `Trueprop :: o ⟹ prop`.

Strip the axioms of `GD.thy` and **Pure proves nothing.**

To Pure, the GA connectives are constants about which it knows nothing.

The object logic is Kehrli's (2025); this development builds on it.

The logic, as Isabelle sees it

```
axiomatization disj ::  "o ⟹ o ⟹ o" (infixr "∨" 30) and
  not ::  "o ⟹ o" ("¬ _") where
    disjE1:  "P ∨ Q ⟹ (P ⟹ R) ⟹ (Q ⟹ R) ⟹ R" and
    dNegE: "¬¬P ⟹ P" and exF: "P ⟹ ¬P ⟹ Q"

axiomatization eq ::  "num ⟹ num ⟹ o" (infixl "=" 45) where
    eqSubst:  "a = b ⟹ Q a ⟹ Q b" and
    eqSym:  "a = b ⟹ b = a"

definition "(p B) ≡ (p ∨ ¬p)"     definition "x N ≡ (x = x)"
```

The logic, as Isabelle sees it

```
axiomatization disj ::  "o ⟹ o ⟹ o" (infixr "∨" 30) and
  not ::  "o ⟹ o" ("¬ _") where
    disjE1:  "P ∨ Q ⟹ (P ⟹ R) ⟹ (Q ⟹ R) ⟹ R" and
    dNegE: "¬¬P ⟹ P" and exF: "P ⟹ ¬P ⟹ Q"

axiomatization eq ::  "num ⟹ num ⟹ o" (infixl "=" 45) where
    eqSubst:  "a = b ⟹ Q a ⟹ Q b" and
    eqSym:  "a = b ⟹ b = a"

definition "(p B) ≡ (p ∨ ¬p)"     definition "x N ≡ (x = x)"
```

Reflexivity is absent. That is the pivot of the whole system.

Since *a* N unfolds to $a = a$, an axiom "$a = a$" would say *every term terminates*. Transitivity is one substitution away; symmetry is an axiom; reflexivity is a *goal*.

Definitions carry no obligation

```
axiomatization def ::  "'a ⟹ 'a ⟹ o" (infix ":=" 10) where
    defE: "a := b ⟹ Q b ⟹ Q a" and
    defI: "a := b ⟹ Q a ⟹ Q b"
```

`a := b` **does not assert** `a = b`. It asserts that a and b are interchangeable in *every* context Q — whether or not either denotes a value.

Definitions carry no obligation

```
axiomatization def ::  "'a ⟹ 'a ⟹ o" (infix ":=" 10) where
    defE: "a := b ⟹ Q b ⟹ Q a" and
    defI: "a := b ⟹ Q a ⟹ Q b"
```

`a := b` **does not assert** `a = b`. It asserts that *a* and *b* are interchangeable in *every* context *Q* — whether or not either denotes a value.

```
add_def:  "add x y := if y = 0 then x else S(add x (P y))"
ack_def:  "ack x y := if x = 0 then y + 1
                      else if y = 0 then ack (P x) 1
                      else ack (P x) (ack x (P y))"
omega_def:  "omega := omega"
```

`omega` is admissible, and `omega N` is underivable.

The promise from the first half of the talk, as one Isabelle mechanism.

What has been proved in it

- **Arithmetic.** $+$, $-$, $\times$, $\div$, $\leq$, $<$ by recursive definition, with termination proved by induction.
- **Ackermann's function is proved total** — and it is *not* primitive recursive, so this is termination-proving strength beyond PRA.
- **Cantor pairing**, defined recursively rather than by the closed formula: injectivity, surjectivity, reconstruction $\langle \mathrm{cpx}\ a, \mathrm{cpy}\ a \rangle = a$, and monotonicity of both projections.
- **Strong induction** in bounded form, $y \leq x$. This is what turns structural recursion on a code into bounded numerical recursion — a component of a code is a *smaller number*.
- **Lists with no datatype package.** $\mathrm{Nil} = 0$, $\mathrm{Cons}\ n\ xs = \langle n, xs \rangle + 1$ — a bijection with $\mathbb{N}$, so no well-formedness predicate is needed at all. Head, tail, `nth`, `len`, `mem`, injectivity, and an induction principle.
- **Tooling.** `simp` and `auto` work on GA goals, with GA lemma sets; `induct` and `cases bool:` for induction and case analysis on a grounded guard; an `unfold_def` method for the `:=` rules; subgoal solvers for routine *habeas quid* obligations.

A test for the system

The existing consistency proofs for GA are in **Isabelle/HOL** —
a classical metatheory far stronger than the object logic.

A test for the system

The existing consistency proofs for GA are in **Isabelle/HOL** —
a classical metatheory far stronger than the object logic.

So: carry the argument out **inside grounded arithmetic itself.**

`GD.thy`	imports Pure	QGA, grounded arithmetic in Pure
`BGA_on_GA.thy`	imports `GD`	BGA arithmetized; its proof checker proved sound
`encode.thy`	imports `BGA_on_GA`	one concrete encoding; the theorem

A test for the system

The existing consistency proofs for GA are in **Isabelle/HOL** —
a classical metatheory far stronger than the object logic.

So: carry the argument out **inside grounded arithmetic itself.**

`GD.thy`	imports Pure	QGA, grounded arithmetic in Pure
`BGA_on_GA.thy`	imports `GD`	BGA arithmetized; its proof checker proved sound
`encode.thy`	imports `BGA_on_GA`	one concrete encoding; the theorem

QGA is stronger than BGA — a weaker system proved consistent inside a stronger one.

The point is that no classical ambient logic is available at any step.

BGA, as numbers

There is one type, `num`. Terms, formulas, judgments, proofs, assignments and the definition list are all **natural numbers**.

- A term or formula is a **tagged pair**: `pack_T tag payload`, with six term tags and two formula tags.
- A judgment is ⟨hypotheses, conclusion⟩. A **proof is a list of judgments**, read head first, each justified by the ones *after* it.

BGA, as numbers

There is one type, `num`. Terms, formulas, judgments, proofs, assignments and the definition list are all **natural numbers**.

- A term or formula is a **tagged pair**: `pack_T tag payload`, with six term tags and two formula tags.
- A judgment is ⟨hypotheses, conclusion⟩. A **proof is a list of judgments**, read head first, each justified by the ones *after* it.

The encoding is never fixed until the last file. `BGA_on_GA.thy` fixes an interface — `tag`, `load`, `pack` — and the smallest set of assumptions the argument needs:

habeas quid	`pack`, `tag`, `load` preserve `N`
injectivity	`tag (pack t L) = t`, `load (pack t L) = L`
retraction	`pack (tag f) (load f) = f`
decrease	`f` $\neq$ `0` $\Longrightarrow$ `load f < f` — payloads are strictly smaller
monotonicity	`pack` is monotone in its payload

Ill-formed codes are never excluded and never recognised. They are simply harmless.

The proof

Everything reduces to one implication — the **soundness bridge**:

> if p checks as a proof of J, and the hypotheses of J are satisfied by an assignment A, then the conclusion of J is satisfied by A.

The proof

Everything reduces to one implication — the **soundness bridge**:

> if p checks as a proof of J, and the hypotheses of J are satisfied by an assignment A, then the conclusion of J is satisfied by A.

1. The step-indexed evaluator determines **at most one value**, so `evals` is a partial function.
2. **Every branch** of the step checker preserves satisfaction, given that the tail of the proof does.
3. That lifts along the proof list **by induction**.
4. Satisfaction of an encoded equation or disequation unpacks into statements about **values**.

The proof

Everything reduces to one implication — the **soundness bridge**:

if p checks as a proof of J, and the hypotheses of J are satisfied by an assignment A, then the conclusion of J is satisfied by A.

1. The step-indexed evaluator determines **at most one value**, so `evals` is a partial function.
2. **Every branch** of the step checker preserves satisfaction, given that the tail of the proof does.
3. That lifts along the proof list **by induction**.
4. Satisfaction of an encoded equation or disequation unpacks into statements about **values**.

Consistency then takes half a page — by **grounded proof by contradiction**.

Which requires `is_valid_proof p J` to be **grounded**: *proof checking must be decidable*, and that is itself a theorem here (`proof_is_bool`).

Four things that made it work

- **Fuel.** "φ is satisfied" hides an $\exists$ over the evaluation budget, so it is not grounded — and $\to I$ needs a grounded antecedent. Fix the budget and the predicate becomes quantifier-free, hence decidable; monotonicity in the budget bridges back.
- **Fresh variables out of the numbering.** The substitution rule must invent a template with a variable occurring nowhere in the judgment. Take index $J + 1$, where J is the *code of the whole judgment*: a variable is $\mathrm{pack}(\mathrm{VAR}, i)$, and packing is monotone in i, so every variable in J has index $\leq J$. Freshness for free, out of the numbering.
- **Every search is a countdown.** GA has no bounded quantifier, so every search the checker performs — for a template, for a premise in the proof list, for the three arguments of an application — is written as a descending recursion.
- **Generalize inside the formula.** Induction instantiates a schematic Q, so the motive must be *one GA formula*. Anything that has to stay general through the induction is quantified with **GA's own $\forall$, inside the motive** — prove $\forall u.\, \forall A.$ `eval_fuel k u A N` by induction on `k`, then instantiate twice to recover what you wanted.
  Which is where the fuel comes back: a motive with an implication inside needs that antecedent grounded before $\to I$ applies.

What is left

The system works. **Using it is still expensive.**

- **Automating *habeas quid*.** Most obligations are of one shape — "this compound term terminates" — and follow by chaining typing rules over already-proved termination lemmas. That is a solver, not a proof.
- **Proof search** for the cases that are not routine — and tactics that close a rule's obligation rather than leaving it as a subgoal.
- **Conditional rewriting** where equality is not reflexive, and induction and case analysis under grounding premises.
- **Inductive datatypes** as a general definitional mechanism, rather than one hand-built encoding at a time.
- **Notation** — λ and `letrec`, types beyond $\mathbb{N}$.

What is left

The system works. **Using it is still expensive.**

- **Automating *habeas quid*.** Most obligations are of one shape — "this compound term terminates" — and follow by chaining typing rules over already-proved termination lemmas. That is a solver, not a proof.
- **Proof search** for the cases that are not routine — and tactics that close a rule's obligation rather than leaving it as a subgoal.
- **Conditional rewriting** where equality is not reflexive, and induction and case analysis under grounding premises.
- **Inductive datatypes** as a general definitional mechanism, rather than one hand-built encoding at a time.
- **Notation** — λ and `letrec`, types beyond $\mathbb{N}$.

The question the whole programme now turns on is measurable:

how often can these obligations be discharged automatically?

3 · *Computable Quantification in RGA*

Ford, arXiv:2607.25533 (2026)

Quantifiers — grounded **reflectively**:

a universal is *true* when the system's own proof search certifies its schematic instance,
false when it refutes a particular numeral instance.

- Proves totality of $+$ and $\times$ as internally quantified theorems
- Represents **exactly the recursively enumerable sets**
- Machine-checked: soundness, consistency, open completeness, $\mathbb{N}$-soundness, a Church–Turing characterisation
- **ω-incompleteness** — grounded truth is r.e., so some family has every numeric instance provable while its closure is *semantically ungrounded*

"A **Markov-flavored, substructural corner** distinct from both classical and intuitionistic arithmetic":
DNE holds · quantified excluded middle fails · refuted universals yield explicit counterexample witnesses
the deduction theorem's abstraction direction fails precisely at ungrounded hypotheses

4 · *Internalized Truth in RGA*

Ford, arXiv:2608.16140 (Aug 2026)

Tarski: no consistent *classical* system with arithmetic defines its own truth.

- A truth predicate for RGA's **full language, quantifiers included**, as an *internal RGA term*
- Compiled from a primitive-recursive decider for RGA's operational semantics, and **proven adequate in both directions**
- Four metatheorems close around it: every formula RGA proves, RGA derives the *internal truth* of; every grounded-true formula is internally provable; internal truth implies internal provability; and **the consistency of RGA follows**
- The two directions run on *disjoint* internal machines — a certified decider and a certified proof-checker, both RGA terms

Along the way: coded syntax and substitution, symbolic unfolding, internal strong induction — **evidence that RGA supports nontrivial mathematical reasoning.**

5, 6 · OGA, and grounded set theory

"ZF fails computably, but holds fictionally, and the fiction is measurable."

Idealizing Useful Fictions in OGA — arXiv:2608.17862

- **OGA is RGA plus one rule** — *ATI*, the ω-grounded universal: if every numeric instance of a universal is certified decided, so is the universal. Otherwise RGA symbol for symbol.
- Decidedness certificates become **abundant**: every totality question about a computable function is certified to *have* an answer, whether or not anyone can produce it.
- It costs no completeness; provability stays r.e., ω-truth deliberately does not. There the Gödel sentence is **a genuine fiction**: neither provable nor refutable, yet valued, with a computable pedigree of what adopting it commits one to.

Fact and Fiction in Grounded Set Theory — Aug 2026

- **Sets of numbers** whose membership a computation *decides*: a full Boolean algebra, **complement included**. The halting problem factors into it plus exactly one new set former.
- **Sets of sets**, hereditarily decided: foundation free, choice a computation — but the classical prohibitions become **theorems** (no universal set, no complement), and the **collecting axioms** — union, replacement, powerset — **leak away**.
- In OGA membership is **three-valued by proof-theoretic status**: affirmed, denied, **fictional**. The collecting axioms return totalized, their leaks now *priced*; Tarski's universe axiom is a theorem.

Reasoning Around Recursion

Curry proved you cannot have both general recursion
and ordinary implication.

Everyone since has given up recursion.

Grounded arithmetic gives up something in implication instead.

Have a thing? arXiv:2510.25369