МАШИННО-ПРОВЕРЕННЫЙ ОТЧЁТ · LEAN 4

Аксиоматика «Обелиска»: аудит, ремонт и реализация TBI-пунктов

Разбор демонстрации пруф-ассистента «Обелиск» (вариант HOL, 2026-05-05). Каждое утверждение этого отчёта о выводимости или противоречивости проверено проверщиком Lean и снабжено точным списком использованных аксиом.

объект: **obeliskproofassistant.github.io** автор системы: **Георгий Дунаев**

проверка: **Lean 4.32.0, без Mathlib** дата: **2026-07-22**

2	9	3	0
независимых вывода противоречия в напечатанной системе	пунктов демо доказано из напечатанного ядра	TBI-пункта (3.45–3.47) реализовано и верифицировано	ошибок · 0 sorry · полный аудит #print axioms

Резюме. «Обелиск» — содержательная система: теория классов уровня Морса–Келли с аксиомой Тарского, ∈-индукцией в изящной форме степени (3.14) и оригинальными парами для собственных классов. Но напечатанный список аксиом **противоречив**: в определении 3.34 (≼) пропущена инъективность, и пустая функция делает ≼ тотальным. Отсюда двумя независимыми путями — через аксиому ограничения размера 3.37 и через аксиому Тарского 3.66 — выводится ⊥. Ремонт минимален (см. §4). Попутно: пары Куратовского несовместимы с определением 3.29, в списке не хватает аксиом пары и объединения, а помеченные «TBI» пары Дунаева здесь реализованы и доказаны — они действительно работают для собственных классов, ради чего и задуманы.

1 · Что такое «Обелиск» математически

Ядро — теория *классов*, где «быть множеством» — определяемый предикат

(3.02: `X set ↔ ∃Y. X∈Y`), а выделение классов импредикативно (3.01: формула φ может квантифицировать по классам — это уровень Морса–Келли, сильнее NBG). Поверх ядра: Heredity (3.12), Exponent (3.13), «Regularity» (3.14) — на самом деле *∈-индукция* в классовой форме, Infinity (3.53), фон-неймановское ограничение размера (3.37) и аксиома Тарского (3.66) — родня Tarski–Grothendieck, на которой стоит Mizar. Метачасть (разделы 4 и 6 демо) добавляет кавычки ‹—›/⟦—⟧ и внутренний предикат выводимости с целью доказать теорему Лёба и вторую теорему Гёделя внутри системы — программа в духе рефлексивных замыканий Фефермана.

Это серьёзная работа с настоящими оригинальными идеями. Именно поэтому она заслуживает того инструмента, для которого и создаются пруф-ассистенты: машинной проверки.

2 · Находки

НАХОДКА 1 **⩽ без инъективности ⇒ противоречие через аксиому 3.37**

```
3.34) Def. “Is not strictly larger than” binary relation:

   (∀(A:Class)∀(B:Class)((A⩽B)↔(∃(f:Class)((f class-function)∧(f[A]⊆B)))
                                          ▲ нет ни инъективности, ни усл
```

Пустой класс ∅ — class-function по определениям 3.29–3.33 (все условия вакуумны: Dom(∅)=∅). Его образ пуст: eval (3.25) выдаёт default-значение **V**, а V — собственный класс (Рассел + Heredity), так что ни одно *множество* b не равно V. Значит ∅[A]=∅⊆B, то есть **A ⩽ B для всех A, B** — и «строго меньше» (3.36) не выполняется никогда. Тогда аксиома 3.37, `X∈V ↔ |X|<|V|`, говорит, что V пусто — а по аксиоме 3.53 в нём лежит ω. Противоречие.

✓ Lean: `PrintedWleq_total , printed_inconsistent_route1 : False` — только из {CCR, ext, heredity, infinity} + гипотеза 3.37. Доказательство не зависит от кодировки пар: пустая функция не строит ни одной пары.

НАХОДКА 2 **Тот же баг убивает универсумы: аксиома Тарского 3.66 противоречива, теорема 3.65 опровержима**

Предикат Tight (3.41) использует $\prec$: раз $\prec$ ложно всегда, Tight(U) означает «$\forall B \subseteq U$: $B \notin U$». Но универсум (3.64) содержит ω и Hereditary-замкнут (3.38), откуда $\varnothing \in U$; при этом $\varnothing \subseteq U$, и Tight требует $\varnothing \notin U$. Значит **универсумов не существует вовсе**: аксиома Тарского 3.66 даёт $\bot$ сама по себе, а заявленная в демо *теорема* 3.65 «V universe» — опровержима в напечатанной системе.

✓ Lean: `no_universe , printed_inconsistent_route2 : False , printed_3_65_refutable .`

НАХОДКА 3

Пары Куратовского + квантор ∃! по классам = «фантомные партнёры» в 3.29

Для собственного класса Y пара Куратовского вырождается: {x,Y}={x}, откуда **(x,Y)=(x,x)**. Поэтому если граф F содержит диагональную пару (x,x), единственность из 3.29 (`∃!(y:Class)((x,y)∈F)`) недостижима: y:=V — второй свидетель. Тождественная функция перестаёт быть class-function. Это *доказывает*, что машинерия функций «Обелиска» обязана опираться на собственные пары Дунаева (3.46), а не на 3.48 — конструкция 3.45–3.47 не украшение, а необходимость.

✓ Lean: `kpair_phantom , no_unique_partner .`

НАХОДКА 4

В списке аксиом нет Пары и Объединения — и их нельзя восстановить

Pairful — часть определения универсума (3.64), и теорема 3.65 фактически анонсирует аксиому пары для V. Теги {{{a}},n} в парах Дунаева должны быть множествами — без аксиомы пары это недоказуемо. Восстановить пару из (исправленного) ограничения размера нельзя: $\preccurlyeq$ определяется через class-functions, чьи графы *уже* состоят из упорядоченных пар — циркулярность. Аудит Lean подтверждает: `dpair_inj` требует ровно аксиому `pairing` (и, что приятно, не требует объединения).

✓ Lean: `#print axioms dpair_inj → {CCR, ext, heredity, infinity, pairing}.`

3 · Что выживает: девять пунктов демо, доказанных из напечатанного ядра

Аудит честный в обе стороны: значительная часть заявленных теорем действительно выводима из напечатанных аксиом {CCR, ext, Heredity, Exponent, Regularity, Infinity} — и теперь у каждой есть машинное доказательство с точным списком использованных аксиом.

ДЕМО	УТВЕРЖДЕНИЕ	ТЕОРЕМА LEAN	АКСИОМЫ
3.07	∅ ⊆ X	`thm_3_07`	CCR
3.11	интенсиональность	`thm_3_11`	CCR
3.18	**Ord(On)**	`thm_3_18`	**только CCR**
3.55	Ind(ω)	`thm_3_55`	CCR·ext·her·inf
3.56	Ind(On)	`thm_3_56`	CCR·her·exp·inf
3.57	Tr(ω)	`thm_3_57`	CCR·ext·her·inf
3.58	Ord(ω)	`thm_3_58`	CCR·ext·her·inf
3.59	ω ∈ On	`thm_3_59`	CCR·ext·her·inf
3.67	индукция на транзитивном классе	`thm_3_67`	**CCR + Regularity**

Три наблюдения, которых нет в демо. **(а)** Пункт 3.67 выводится из одной лишь 3.14 — подтверждение, что «Regularity» Дунаева — это в точности ∈-индукция; формулировка через $\mathscr{P}(X)\subseteq X \to V\subseteq X$ корректна и красива. **(б)** Ind(ω) доказуема из ядра классическим трюком: либо индуктивный класс существует, либо ω=V — что противоречит 3.53 + Расселу (`exists_inductive`). **(в)** Ind(On) спасает транзитивность: для транзитивного a верно $a\cup\{a\} \subseteq \mathscr{P}(a)$, и множественность следует из Heredity+Exponent — без аксиом пары и объединения (`isSet_succ_of_Tr`). Единственный пункт, которому добавленные аксиомы правда нужны, — Ind(V) (3.54), из-за нетранзитивных множеств.

4 · TBI-пункты 3.45–3.47 — реализованы и доказаны

Пары и списки Дунаева в демо помечены «TBI» (to be implemented). Здесь они реализованы, и их корректность доказана *для произвольных классов, включая собственные* — то, ради чего конструкция и задумана, и что пары Куратовского дать не могут (Находка 3):

```
dpr1_dpair : pr1(A,B) = A                  -- для ЛЮБЫХ классов A, B
dpr2_dpair : pr2(A,B) = B
dpair_inj  : (A,B) = (C,D) → A = C ∧ B = D
dpair_proper_showcase : pr1(V,On) = V ∧ pr2(V,On) = On   -- собственные кла
dsingle_ne_dnil : [A] ≠ []                 -- даже при A = ∅: маркер {0,n} р
```

Последняя строка — проверка главного дизайнерского решения конструкции: маркер {0,n} делает длину списка восстановимой всегда, включая пустые компоненты. Это аккуратное усиление стандартного тегирования $A\times\{1\} \cup B\times\{2\}$, которое проблему $[\varnothing]$ vs [] не решает.

5 · Минимальный ремонт

1. **3.34′:** $A \preccurlyeq B \longleftrightarrow \exists f$ (f class-function ∧ $A \subseteq \mathrm{Dom}\ f$ ∧ $f[A] \subseteq B$ ∧ f инъективна на A). С этим исправлением 3.37 становится классическим фон-неймановским ограничением размера, а 3.41/3.66 — осмысленными.

2. **Добавить аксиомы Пары и Объединения** — стандартные для MK/NBG, верные во всех интендированных моделях; без них недоказуемы 3.54, 3.65 и собственные TBI-конструкции демо (Находка 4).

3. **3.29 читать через пары Дунаева** (либо ограничить ∃! множествами) — иначе фантомные партнёры (Находка 3).

4. **Зафиксировать следствие:** 3.37′ влечёт глобальный выбор (von Neumann, 1925) — это стоит сделать осознанным решением системы.

5. **Раздел 6:** внешняя теория в 6.02/6.03 должна быть сильнее Q — Q не доказывает условий выводимости Гильберта–Бернайса–Лёба; стандартный выбор — EA или $I\Sigma_1$.

Согласованность отремонтированного ядра: модель — $V_{\kappa+1}$ для κ —

недостижимого предела недостижимых (множества = $V_к$, классы = $V_{к+1}$). Система калибруется в окрестности MK + Limitation of Size + универсумы Тарски–Гротендика — строго сильнее ZFC, как и задумано.

6 · Куда это развивать: программа «первоклассной математики»

Статья A (материал готов — этот отчёт + Lean-файл): «The Obelisk class theory: a machine-checked audit and repair» — находки, ремонт, верифицированная положительная часть, пары Дунаева как теоремы. Формат: ITP/CPP или arXiv с артефактом. **Статья B**: точная калибровка силы — взаимная интерпретируемость с MK+LoS+TG; кардинальная арифметика классов из 3.35 (|On|=On, Card(On), ¬Card(V) — пункты 3.60–3.63). **Статья C**: метатеория раздела 6 — типизированная кавычка/eval как модальность в духе GL, связь F с рефлексивным замыканием Фефермана и теориями истины; стратификация eval для класс-кванторов (обход теоремы Тарского); формализованные Лёб и Gödel II поверх существующих механизаций (Paulson в Isabelle; Lean-библиотека Foundation). **Инженерное**: экспорт доказательств «Обелиска» в проверяемый формат (Lean/Metamath) — тогда каждая строка демо получает независимую машинную проверку автоматически.

Что этот отчёт утверждает — и чего не утверждает

Проверен *напечатанный* список аксиом демо от 2026-05-05 — то, что опубликовано. Внутренняя реализация «Обелиска» может отличаться от печати (например, ⩽ в коде может быть инъективным); тогда находки 1–2 — о выводе демо, а не о ядре системы. Формализация моделирует CCR лин-предикатами — это надмножество синтаксической схемы; существенно, что *каждый* использованный в доказательствах инстанс comprehension записывается формулой языка самого демо (Рассел, Dom, img, Ind и т.п. — собственные определения системы). Все прочие определения перенесены дословно, с сохранением нумерации.

7 · Воспроизведение

```
# установка Lean (однократно)
curl https://elan.lean-lang.org/elan-init.sh -sSf | sh -s -- -y --default-t

# проверка (из папки obelisk-analysis)
lean Obelisk.lean        # exit 0, без sorry; в конце — аудит #print axioms
```

▼ **Полный исходник Obelisk.lean (самодостаточный, ~660 строк)**

```
/-
  ============================================================================
  A machine-checked analysis of the axiom system of the "Obelisk"
  proof assistant (higher-order-logic variant, demo dated 2026-05-05,
  https://obeliskproofassistant.github.io/), by Georgy Dunaev.

  Checked with Lean 4.32.0.  Self-contained: no Mathlib, only core Lean

  Contents.
    Part 0.  The base theory exactly as printed in the demo:
             impredicative class comprehension (3.01), extensionality (
             heredity (3.12), exponent (3.13), regularity-as-∈-inductio
             infinity (3.53).  These axioms are consistent (model: take
             uncountable stage V_α of the cumulative hierarchy with α a
             ordinal > ω, classes = subsets of V_α).
    Part 1.  THE PRINTED SYSTEM IS INCONSISTENT, in two independent way
             (a) definition 3.34 of ≼ has no injectivity requirement, s
                 empty class-function witnesses A ≼ B for ALL A, B; her
                 size-limitation axiom 3.37 forces V to have no members
                 contradicting axiom 3.53 (ω is a set);
             (b) the same ≼-bug propagates through "Tight" (3.41) into
                 universe predicate (3.64): NO class can be a universe,
                 Tarski's axiom 3.66 is itself contradictory, and the a
                 theorem 3.65 ("V is a universe") is refutable as print
           Also: with Kuratowski pairs (3.48), "concentrated" (3.29)
           phantom partners — for a proper class Y, (x,Y) = (x,x) — s
           uniqueness in 3.29 fails at every diagonal point.  This sh
           function machinery must use the Dunaev pairs (3.46), which
           turn require Pairing and Union axioms absent from the prin
           list.
    Part 2.  What survives: the announced theorems that ARE derivable f
             printed base axioms alone, machine-checked:
             3.07, 3.11, 3.18 (Ord(On)), 3.55 (Ind(ω)), 3.56 (Ind(On)),
             3.57 (Tr(ω)), 3.58 (Ord(ω)), 3.59 (ω ∈ On), 3.67 (∈-induct
             on a transitive class), and the components of "V is a univ
             that do not mention Tight.
```

```
            that do not mention right.
    Part 3.  The repair: add Pairing and Union (they are NOT recoverabl
             limitation-of-size — the ≼ machinery itself presupposes or
             pairs, a circularity — and they are required by the demo's
             TBI items).  With them we implement and verify the demo's
             items 3.44–3.47: Dunaev's tagged lists/pairs, which — unli
             Kuratowski's — are faithful on PROPER classes:
             pr1(A,B) = A, pr2(A,B) = B, and (A,B) = (C,D) → A = C ∧ B
             for arbitrary classes, and [A] ≠ [] even for A = ∅
             (the {0,n} marker device works as designed).
  ============================================================================
-/

set_option linter.unusedVariables false

namespace Obelisk
noncomputable section

/-! ### Part 0.  Language and the printed base axioms -/

axiom Class : Type
axiom Mem : Class → Class → Prop
infix:50 " ∈' "  => Mem

/-- 3.02: `X` is a set iff it is a member of some class. -/
def isSet (X : Class) : Prop := ∃ Y, X ∈' Y
/-- 3.03. -/
def proper (K : Class) : Prop := ¬ isSet K

/-- 3.01 (CCR): the class `{x | φ}`.  The demo's `φ : Fm` ranges over f
    with quantifiers over `Class` (Morse–Kelley-style impredicative
    comprehension); we model it by a Lean-level predicate. -/
axiom cls : (Class → Prop) → Class
axiom mem_cls : ∀ (φ : Class → Prop) (x : Class), x ∈' cls φ ↔ (isSet x

/-- 3.10 Extensionality. (3.08/3.09 are Lean's `Eq` congruence, for fre
axiom extAx : ∀ {X Y : Class}, (∀ z, z ∈' X ↔ z ∈' Y) → X = Y

/-- 3.06. -/
def Sub (X Y : Class) : Prop := ∀ a, a ∈' X → a ∈' Y
infix:50 " ⊆' " => Sub

/-- 3.04, 3.05. -/
def Null : Class := cls (fun _ => False)
def V    : Class := cls (fun _ => True)
/-- Powerclass (used by 3.13, 3.14). -/
def power (X : Class) : Class := cls (fun y => y ⊆' X)
```

```
def power (X : Class) : Class := cls (fun y => y ⊆' X)

/-- 3.12 Heredity. -/
axiom heredity : ∀ {A B : Class}, A ⊆' B → isSet B → isSet A
/-- 3.13 Exponent. -/
axiom exponent : ∀ {X : Class}, isSet X → isSet (power X)
/-- 3.14 "Regularity" — the powerclass form of ∈-induction. -/
axiom regularity : ∀ {X : Class}, power X ⊆' X → V ⊆' X

/-! Printed definitions 3.15–3.52, verbatim. -/

/-- 3.19, 3.20. -/
def sUnion (K : Class) : Class := cls (fun x => ∃ y, y ∈' K ∧ x ∈' y)
def sInter (K : Class) : Class := cls (fun x => ∀ y, y ∈' K → x ∈' y)
/-- 3.42, 3.43, 3.49, 3.50. -/
def sing  (a : Class)   : Class := cls (fun x => x = a)
def upair (a b : Class) : Class := cls (fun x => x = a ∨ x = b)
def bUnion (A B : Class) : Class := cls (fun x => x ∈' A ∨ x ∈' B)
def bInter (A B : Class) : Class := cls (fun x => x ∈' A ∧ x ∈' B)
/-- 3.48 Kuratowski pair (a,b) = {{a,a},{a,b}}. -/
def kpair (a b : Class) : Class := upair (upair a a) (upair a b)
/-- 3.21 Domain. -/
def Dom (F : Class) : Class := cls (fun x => ∃ b, kpair x b ∈' F)
/-- 3.22 IF-THEN-ELSE-FI (the demo's semantic definition). -/
def IfC (ψ : Prop) (A B : Class) : Class :=
  cls (fun v => (ψ ∧ v ∈' A) ∨ (¬ψ ∧ v ∈' B))
/-- 3.25 Evaluation, with the demo's `ELSE V` default. -/
def app (F a : Class) : Class :=
  IfC (∃ s, kpair a s ∈' F) (sUnion (cls (fun s => kpair a s ∈' F))) V
/-- 3.26 Image. -/
def img (f A : Class) : Class := cls (fun b => ∃ a, a ∈' A ∧ app f a =
/-- The demo's `∃!` quantifier (4.05): over ALL classes, proper include
def ExistsU (p : Class → Prop) : Prop := ∃ y, p y ∧ ∀ z, p z → z = y

/-- 3.27–3.33. -/
def isSingleton  (s : Class) : Prop := ExistsU (fun x => x ∈' s)
def setValued    (F : Class) : Prop := ∀ x, x ∈' Dom F → isSet (app F x
def concentrated (F : Class) : Prop :=
  ∀ x, x ∈' Dom F → isSet (app F x) → ExistsU (fun y => kpair x y ∈' F)
def flatProper   (F : Class) : Prop :=
  ∀ x, x ∈' Dom F → proper (app F x) → ∀ b, kpair x b ∈' F → isSingleto
def isFamily        (F : Class) : Prop := concentrated F ∧ flatProper F
def isClassFunction (F : Class) : Prop := isFamily F ∧ setValued F

/-- 3.34 AS PRINTED — note: no injectivity, no domain condition. -/
def PrintedWleq (A B : Class) : Prop :=
```

```
  ∃ f, isClassFunction f ∧ img f A ⊆' B
/-- 3.36 as printed: A ≺ B ↔ ¬(B ≼ A). -/
def PrintedWlt (A B : Class) : Prop := ¬ PrintedWleq B A

/-- 3.15, 3.16, 3.17. -/
def Tr (A : Class) : Prop := ∀ B, B ∈' A → B ⊆' A
def OrdC (A : Class) : Prop := Tr A ∧ ∀ B, B ∈' A → Tr B
def On : Class := cls OrdC
/-- 3.35 Class cardinality (as printed; uses the printed ≼). -/
def card (K : Class) : Class :=
  bInter On (sInter (cls (fun a => a ∈' On ∧ PrintedWleq K a)))

/-- 3.51, 3.52. -/
def IndC (X : Class) : Prop :=
  Null ∈' X ∧ ∀ a, a ∈' X → bUnion a (sing a) ∈' X
def omega : Class := cls (fun x => ∀ Q, IndC Q → x ∈' Q)
def succ (a : Class) : Class := bUnion a (sing a)

/-- 3.53 Infinity. -/
axiom infinity : isSet omega

/-- 3.38–3.41 and 3.64, verbatim (Tight uses the printed ≺). -/
def Hereditary (A : Class) : Prop :=
  ∀ B, B ∈' A → ∀ X, X ⊆' B → X ∈' A
def Powerful (U : Class) : Prop := ∀ B, B ∈' U → power B ∈' U
def Pairful  (U : Class) : Prop :=
  ∀ A B, A ∈' U → B ∈' U → upair A B ∈' U
def Tight (U : Class) : Prop :=
  ∀ B, B ⊆' U → (PrintedWlt B U ↔ B ∈' U)
def isUniverse (U : Class) : Prop :=
  ((omega ∈' U) ∧ ((Tr U ∧ Hereditary U) ∧ (Powerful U ∧ Tight U))) ∧ F

/-! ### Basic lemmas -/

theorem not_mem_Null (x : Class) : ¬ x ∈' Null :=
  fun h => ((mem_cls _ x).mp h).2

theorem Null_sub (X : Class) : Null ⊆' X :=
  fun a ha => absurd ha (not_mem_Null a)

theorem mem_V {x : Class} : x ∈' V ↔ isSet x :=
  ⟨fun h => ((mem_cls _ x).mp h).1, fun h => (mem_cls _ x).mpr ⟨h, triv

theorem isSet_of_mem {x Y : Class} (h : x ∈' Y) : isSet x := ⟨Y, h⟩

/-- ∅ is a set (from Heredity and Infinity). -/
```

```
theorem isSet_Null : isSet Null := heredity (Null_sub omega) infinity

/-- Russell: the universal class is proper (uses CCR and Heredity). -/
theorem V_proper : proper V := fun hV =>
  have hsub : cls (fun x => ¬ x ∈' x) ⊆' V :=
    fun a ha => mem_V.mpr ((mem_cls _ a).mp ha).1
  have hset : isSet (cls (fun x => ¬ x ∈' x)) := heredity hsub hV
  have hiff := mem_cls (fun x => ¬ x ∈' x) (cls (fun x => ¬ x ∈' x))
  (Classical.em (cls (fun x => ¬ x ∈' x) ∈' cls (fun x => ¬ x ∈' x))).e
    (fun h => (hiff.mp h).2 h)
    (fun h => h (hiff.mpr ⟨hset, h⟩))

theorem mem_upair {x a b : Class} :
    x ∈' upair a b ↔ isSet x ∧ (x = a ∨ x = b) := mem_cls _ _

theorem mem_sing {x a : Class} : x ∈' sing a ↔ isSet x ∧ x = a := mem_c

theorem self_mem_sing {a : Class} (h : isSet a) : a ∈' sing a :=
  mem_sing.mpr ⟨h, rfl⟩

theorem upair_self (a : Class) : upair a a = sing a :=
  extAx (fun z =>
    ⟨fun h => mem_sing.mpr ⟨(mem_upair.mp h).1, (mem_upair.mp h).2.elim
     fun h => mem_upair.mpr ⟨(mem_sing.mp h).1, Or.inl (mem_sing.mp h).

theorem sing_inj {a b : Class} (ha : isSet a) (h : sing a = sing b) : a
  (mem_sing.mp (h ▸ self_mem_sing ha)).2

/-! ### Part 1.  Inconsistency of the printed system -/

/-- The empty class has empty domain, hence is (vacuously) a class-func
theorem Dom_Null_empty (x : Class) : ¬ x ∈' Dom Null := fun hx =>
  match ((mem_cls _ x).mp hx).2 with
  | ⟨_, hb⟩ => not_mem_Null _ hb

theorem Null_concentrated : concentrated Null :=
  fun x hx => absurd hx (Dom_Null_empty x)
theorem Null_flatProper : flatProper Null :=
  fun x hx => absurd hx (Dom_Null_empty x)
theorem Null_setValued : setValued Null :=
  fun x hx => absurd hx (Dom_Null_empty x)

theorem Null_isClassFunction : isClassFunction Null :=
  ⟨⟨Null_concentrated, Null_flatProper⟩, Null_setValued⟩

/-- Evaluation of the empty class-function falls into the `ELSE V` bran
```

```
theorem app_Null (a : Class) : app Null a = V := by
  have hcond : ¬ ∃ s, kpair a s ∈' Null :=
    fun ⟨_, hs⟩ => not_mem_Null _ hs
  apply extAx
  intro v
  constructor
  · intro hv
    have h' := (mem_cls _ v).mp hv
    exact match h'.2 with
      | Or.inl ⟨hψ, _⟩ => absurd hψ hcond
      | Or.inr ⟨_, hvV⟩ => hvV
  · intro hv
    exact (mem_cls _ v).mpr ⟨isSet_of_mem hv, Or.inr ⟨hcond, hv⟩⟩

/-- Its image is empty: no SET can equal `V`, the default value. -/
theorem img_Null_empty (A b : Class) : ¬ b ∈' img Null A := fun hb =>
  match (mem_cls _ b).mp hb with
  | ⟨hbset, _, _, happ⟩ =>
      V_proper ((happ.symm.trans (app_Null _)) ▸ hbset)

/-- **Finding 1.** With definition 3.34 as printed, `A ≼ B` holds for A
    `A B` — the empty class-function is a witness. -/
theorem PrintedWleq_total (A B : Class) : PrintedWleq A B :=
  ⟨Null, Null_isClassFunction, fun b hb => absurd hb (img_Null_empty A

/-- Hence `≺` never holds… -/
theorem PrintedWlt_never (A B : Class) : ¬ PrintedWlt A B :=
  fun h => h (PrintedWleq_total B A)

/-- **Theorem (inconsistency, route 1).**  The printed base axioms toge
    with the printed size-limitation axiom 3.37 derive `False`:
    3.37 forces `V` to be empty, but ω ∈ V by 3.53. -/
theorem printed_inconsistent_route1
    (sizeLimitation : ∀ X : Class, X ∈' V ↔ PrintedWlt (card X) (card \
    False :=
  PrintedWlt_never (card omega) (card V)
    ((sizeLimitation omega).mp (mem_V.mpr infinity))

/-- **Theorem (inconsistency, route 2).**  The ≼-bug propagates through
    `Tight` (3.41): no class is a universe — ∅ must both belong (via
    Hereditary, since ω ∈ U) and not belong (via Tight, since ≺ never
    holds) to it. -/
theorem no_universe (U : Class) (hU : isUniverse U) : False :=
  have hω     : omega ∈' U  := hU.1.1
  have hHer   : Hereditary U := hU.1.2.1.2
  have hTight : Tight U      := hU.1.2.2.2
```

```
  have hNull  : Null ∈' U    := hHer omega hω Null (Null_sub omega)
  PrintedWlt_never Null U ((hTight Null (Null_sub U)).mpr hNull)

/-- Hence Tarski's axiom 3.66 is itself contradictory over the base… -/
theorem printed_inconsistent_route2
    (tarski : ∀ x, isSet x → ∃ u, ((isUniverse u ∧ x ∈' u) ∧ isSet u))
    False :=
  match tarski Null isSet_Null with
  | ⟨u, ⟨hu, _⟩, _⟩ => no_universe u hu

/-- …and the demo's announced THEOREM 3.65 ("V is a universe") is refut
    as printed. -/
theorem printed_3_65_refutable : ¬ isUniverse V := no_universe V

/-- **Finding (phantom partners).**  With Kuratowski pairs, a proper cl
    collapses the pair: `{x, y} = {x}` and hence `(x, y) = (x, x)`.  So
    uniqueness demanded by "concentrated" (3.29) fails at every diagona
    point, and the function machinery cannot rest on 3.48 — it needs th
    Dunaev pairs 3.46. -/
theorem upair_proper_collapse {x y : Class} (hy : proper y) :
    upair x y = sing x := by
  apply extAx
  intro z
  constructor
  · intro hz
    have h' := (mem_cls _ z).mp hz
    exact match h'.2 with
      | Or.inl h => mem_sing.mpr ⟨h'.1, h⟩
      | Or.inr h => absurd (h ▸ h'.1) hy
  · intro hz
    have h' := (mem_cls _ z).mp hz
    exact mem_upair.mpr ⟨h'.1, Or.inl h'.2⟩

theorem kpair_phantom {x y : Class} (hy : proper y) :
    kpair x y = kpair x x := by
  show upair (upair x x) (upair x y) = upair (upair x x) (upair x x)
  rw [upair_proper_collapse hy, upair_self]

/-- No `F` containing a diagonal pair `(x,x)` has a UNIQUE partner at `
    `y := V` is a phantom second witness. -/
theorem no_unique_partner {F x : Class} (hx : isSet x)
    (hxx : kpair x x ∈' F) : ¬ ExistsU (fun y => kpair x y ∈' F) :=
  fun ⟨y₀, _, huniq⟩ =>
    have h1 : x = y₀ := huniq x hxx
    have h2 : V = y₀ := huniq V (by
      show kpair x V ∈' F
```

```
      rw [kpair_phantom V_proper]; exact hxx)
    V_proper ((h1.trans h2.symm) ▸ hx)

/-! ### Part 2.  What survives: theorems derivable from the printed bas

/-- 3.07: ∅ ⊆ X. -/
theorem thm_3_07 (X : Class) : Null ⊆' X := Null_sub X

/-- 3.11 "Intensionality": sets with the same containers are equal. -/
theorem thm_3_11 {X Y : Class} (hX : isSet X) (_hY : isSet Y)
    (h : ∀ W, X ∈' W ↔ Y ∈' W) : X = Y :=
  have hXX : X ∈' sing X := self_mem_sing hX
  (((mem_cls _ Y).mp ((h (sing X)).mp hXX)).2).symm

/-- Members of `On` are exactly the ordinal sets. -/
theorem mem_On {B : Class} : B ∈' On ↔ isSet B ∧ OrdC B := mem_cls _ _

/-- 3.18: `On` is transitive… -/
theorem Tr_On : Tr On := fun B hB b hb =>
  have h := mem_On.mp hB
  mem_On.mpr ⟨isSet_of_mem hb,
    h.2.2 b hb,
    fun c hc => h.2.2 c (h.2.1 b hb c hc)⟩

/-- …and 3.18 in full: `Ord(On)` — the class of all ordinals is an ordi
    class (no Burali-Forti paradox: `On` is proper, which is fine here)
theorem thm_3_18 : OrdC On :=
  ⟨Tr_On, fun B hB => (mem_On.mp hB).2.1⟩

theorem mem_omega {x : Class} :
    x ∈' omega ↔ isSet x ∧ ∀ Q, IndC Q → x ∈' Q := mem_cls _ _

theorem Null_mem_omega : Null ∈' omega :=
  mem_omega.mpr ⟨isSet_Null, fun _ hQ => hQ.1⟩

/-- A classical detour available in the printed system: SOME inductive
    exists — otherwise ω = V, contradicting 3.53 + Russell. -/
theorem exists_inductive : ∃ Q, IndC Q :=
  (Classical.em (∃ Q, IndC Q)).elim id (fun hno => by
    have hωV : omega = V := extAx (fun x =>
      ⟨fun h => mem_V.mpr (isSet_of_mem h),
       fun h => mem_omega.mpr ⟨mem_V.mp h,
         fun Q hQ => absurd ⟨Q, hQ⟩ hno⟩⟩)
    exact absurd (hωV ▸ infinity) V_proper)

theorem isSet_succ_of_mem_omega {a : Class} (ha : a ∈' omega) :
```

```
theorem isSet_succ_of_mem_omega {a : Class} (ha : a ∈' omega) :
    isSet (succ a) :=
  match exists_inductive with
  | ⟨Q, hQ⟩ => ⟨Q, hQ.2 a ((mem_omega.mp ha).2 Q hQ)⟩

/-- 3.55: ω is inductive — provable from the printed base alone. -/
theorem thm_3_55 : IndC omega :=
  ⟨Null_mem_omega,
   fun a ha => mem_omega.mpr
     ⟨isSet_succ_of_mem_omega ha,
      fun Q hQ => hQ.2 a ((mem_omega.mp ha).2 Q hQ)⟩⟩

/-- The induction principle of ω: ω is contained in every inductive cla
theorem omega_sub {Q : Class} (hQ : IndC Q) : omega ⊆' Q :=
  fun x hx => (mem_omega.mp hx).2 Q hQ

theorem mem_succ {x a : Class} :
    x ∈' succ a ↔ isSet x ∧ (x ∈' a ∨ x = a) := by
  constructor
  · intro h
    have h' := (mem_cls _ x).mp h
    exact ⟨h'.1, h'.2.elim Or.inl (fun hs => Or.inr ((mem_cls _ x).mp h
  · intro ⟨hset, h⟩
    exact (mem_cls _ x).mpr ⟨hset,
      h.elim Or.inl (fun he => Or.inr ((mem_cls _ x).mpr ⟨hset, he⟩))⟩

theorem sub_succ {a : Class} : a ⊆' succ a :=
  fun x hx => mem_succ.mpr ⟨isSet_of_mem hx, Or.inl hx⟩

theorem mem_bInter {x A B : Class} :
    x ∈' bInter A B ↔ isSet x ∧ x ∈' A ∧ x ∈' B := mem_cls _ _

/-- 3.57: ω is transitive (by ω-induction on `{x ∈ ω | x ⊆ ω}`). -/
theorem thm_3_57 : Tr omega := by
  have hInd : IndC (bInter omega (cls (fun x => x ⊆' omega))) := by
    constructor
    · exact mem_bInter.mpr ⟨isSet_Null, Null_mem_omega,
        (mem_cls _ _).mpr ⟨isSet_Null, Null_sub omega⟩⟩
    · intro a ha
      have h := mem_bInter.mp ha
      have haω : a ∈' omega := h.2.1
      have hasub : a ⊆' omega := ((mem_cls _ a).mp h.2.2).2
      have hsω : succ a ∈' omega := thm_3_55.2 a haω
      exact mem_bInter.mpr ⟨isSet_of_mem hsω, hsω,
        (mem_cls _ _).mpr ⟨isSet_of_mem hsω,
          fun x hx => (mem_succ.mp hx).2.elim
```

```
            (fun h' => hasub x h') (fun h' => h'.symm ▸ haω)⟩⟩
  intro B hB
  exact ((mem_cls _ B).mp (mem_bInter.mp (omega_sub hInd B hB)).2.2).2

/-- Every member of ω is transitive (again by ω-induction). -/
theorem Tr_of_mem_omega {B : Class} (hB : B ∈' omega) : Tr B := by
  have hInd : IndC (bInter omega (cls Tr)) := by
    constructor
    · exact mem_bInter.mpr ⟨isSet_Null, Null_mem_omega,
        (mem_cls _ _).mpr ⟨isSet_Null,
          fun C hC => absurd hC (not_mem_Null C)⟩⟩
    · intro a ha
      have h := mem_bInter.mp ha
      have haω : a ∈' omega := h.2.1
      have haTr : Tr a := ((mem_cls _ a).mp h.2.2).2
      have hsω : succ a ∈' omega := thm_3_55.2 a haω
      refine mem_bInter.mpr ⟨isSet_of_mem hsω, hsω,
        (mem_cls _ _).mpr ⟨isSet_of_mem hsω, ?_⟩⟩
      intro C hC
      cases (mem_succ.mp hC).2 with
      | inl h' => exact fun x hx => sub_succ x (haTr C h' x hx)
      | inr h' => exact fun x hx => sub_succ x (h' ▸ hx)
  exact ((mem_cls _ B).mp (mem_bInter.mp (omega_sub hInd B hB)).2.2).2

/-- 3.58: ω is an ordinal class. -/
theorem thm_3_58 : OrdC omega :=
  ⟨thm_3_57, fun B hB => Tr_of_mem_omega hB⟩

/-- 3.59: ω ∈ On. -/
theorem thm_3_59 : omega ∈' On := mem_On.mpr ⟨infinity, thm_3_58⟩

/-- For a TRANSITIVE set, `succ a ⊆ 𝒫(a)`, so `succ a` is a set withou
    pairing/union axiom — this rescues Ind(On) in the printed base. -/
theorem isSet_succ_of_Tr {a : Class} (haTr : Tr a) (ha : isSet a) :
    isSet (succ a) :=
  heredity
    (fun x hx => (mem_cls _ x).mpr ⟨isSet_of_mem hx,
      (mem_succ.mp hx).2.elim (fun h => haTr x h)
        (fun h => fun z hz => h ▸ hz)⟩)
    (exponent ha)

/-- 3.56: On is inductive — provable from the printed base alone. -/
theorem thm_3_56 : IndC On := by
  constructor
  · exact mem_On.mpr ⟨isSet_Null,
      fun B hB => absurd hB (not_mem_Null B),
```

```
      fun B hB => absurd hB (not_mem_Null B)⟩
  · intro a ha
    have h := mem_On.mp ha
    have haTr : Tr a := h.2.1
    refine mem_On.mpr ⟨isSet_succ_of_Tr haTr h.1, ?_, ?_⟩
    · intro B hB
      cases (mem_succ.mp hB).2 with
      | inl h' => exact fun x hx => sub_succ x (haTr B h' x hx)
      | inr h' => exact fun x hx => sub_succ x (h' ▸ hx)
    · intro B hB
      cases (mem_succ.mp hB).2 with
      | inl h' => exact h.2.2 B h'
      | inr h' => exact h'.symm ▸ haTr

/-- 3.67: the general induction principle on a transitive class, derive
    from the printed "Regularity" 3.14 — which is exactly ∈-induction i
    powerclass form. -/
theorem thm_3_67 {K A : Class} (hK : Tr K) (h : bInter K (power A) ⊆' A
    K ⊆' A := by
  have step : power (cls (fun x => x ∈' K → x ∈' A)) ⊆'
      cls (fun x => x ∈' K → x ∈' A) := by
    intro y hy
    have h' := (mem_cls _ y).mp hy
    refine (mem_cls _ y).mpr ⟨h'.1, fun hyK => ?_⟩
    have hyA : y ⊆' A := fun x hx =>
      ((mem_cls _ x).mp (h'.2 x hx)).2 (hK y hyK x hx)
    exact h y (mem_bInter.mpr ⟨h'.1, hyK,
      (mem_cls _ y).mpr ⟨h'.1, hyA⟩⟩)
  intro k hk
  exact ((mem_cls _ k).mp
    (regularity step k (mem_V.mpr (isSet_of_mem hk)))).2 hk

/-- The Tight-free components of "V is a universe" (towards 3.65), prov
    from the printed base. -/
theorem V_Tr : Tr V := fun _B _hB b hb => mem_V.mpr (isSet_of_mem hb)
theorem V_Hereditary : Hereditary V :=
  fun _B hB X hX => mem_V.mpr (heredity hX (mem_V.mp hB))
theorem V_Powerful : Powerful V :=
  fun _B hB => mem_V.mpr (exponent (mem_V.mp hB))
theorem omega_mem_V : omega ∈' V := mem_V.mpr infinity

/-! ### Part 3.  The repair: Pairing and Union, and the TBI items 3.44-

    Pairing and Union are absent from the printed axiom list, yet:
    (i)   `Pairful` is part of the universe definition 3.64, and 3.65
          announces `Pairful V` — i.e. the pairing axiom itself;
```

```
    (ii)  the Dunaev lists 3.45 (marked TBI in the demo) need the tags
          `{{{a}},n}` to be sets;
    (iii) pairing cannot be recovered from a repaired limitation-of-siz
          axiom, because ≼ is defined through class functions whose gra
          already consist of ordered pairs — a circularity.
    So we add them.  (Both hold in every intended model.) -/

axiom pairing : ∀ {a b : Class}, isSet a → isSet b → isSet (upair a b)
axiom unionAx : ∀ {x : Class}, isSet x → isSet (sUnion x)

theorem isSet_sing {a : Class} (h : isSet a) : isSet (sing a) :=
  upair_self a ▸ pairing h h

theorem bUnion_eq_sUnion_upair {A B : Class} (hA : isSet A) (hB : isSet
    bUnion A B = sUnion (upair A B) :=
  extAx (fun x =>
    ⟨fun h =>
      have h' := (mem_cls _ x).mp h
      (mem_cls _ x).mpr ⟨h'.1, h'.2.elim
        (fun hx => ⟨A, mem_upair.mpr ⟨hA, Or.inl rfl⟩, hx⟩)
        (fun hx => ⟨B, mem_upair.mpr ⟨hB, Or.inr rfl⟩, hx⟩)⟩,
     fun h =>
      have h' := (mem_cls _ x).mp h
      (mem_cls _ x).mpr ⟨h'.1,
        match h'.2 with
        | ⟨_, hy, hxy⟩ => (mem_upair.mp hy).2.elim
            (fun e => Or.inl (e ▸ hxy)) (fun e => Or.inr (e ▸ hxy))⟩⟩)

theorem isSet_succ {a : Class} (ha : isSet a) : isSet (succ a) := by
  show isSet (bUnion a (sing a))
  rw [bUnion_eq_sUnion_upair ha (isSet_sing ha)]
  exact unionAx (pairing ha (isSet_sing ha))

/-- 3.54: V is inductive — NOW provable.  (This one genuinely needs the
    added axioms: for a non-transitive set `a`, the 𝒫-trick of
    `isSet_succ_of_Tr` does not apply.) -/
theorem thm_3_54 : IndC V :=
  ⟨mem_V.mpr isSet_Null, fun a ha => mem_V.mpr (isSet_succ (mem_V.mp ha

/-- The `Pairful` component of 3.65 — literally the pairing axiom. -/
theorem V_Pairful : Pairful V :=
  fun _A _B hA hB => mem_V.mpr (pairing (mem_V.mp hA) (mem_V.mp hB))

/-! #### The Dunaev tagged lists (3.44–3.47) — the demo's TBI, implemen

    3.44 numerals;  3.45  mark(A,n) = {{0,n}} ∪ { {{a}},n} | a ∈ A };
```

```
    3.46  (A,B) := [A,B] = mark(A,1) ∪ mark(B,2);
    3.47  pr1(L) = { a | {{{a}},1} ∈ L },  pr2(L) = { b | {{{b}},2} ∈ L

    The point of the construction: it is faithful on PROPER classes, wh
    Kuratowski pairs collapse (see `kpair_phantom`).  We verify this. -

def zero : Class := Null
def one  : Class := sing Null
def two  : Class := upair Null one

theorem isSet_one : isSet one := isSet_sing isSet_Null
theorem isSet_two : isSet two := pairing isSet_Null isSet_one

theorem Null_ne_one : Null ≠ one := fun h =>
  have h1 : Null ∈' one := self_mem_sing isSet_Null
  not_mem_Null Null (h.symm ▸ h1)

theorem Null_ne_two : Null ≠ two := fun h =>
  have h1 : one ∈' two := mem_upair.mpr ⟨isSet_one, Or.inr rfl⟩
  not_mem_Null one (h.symm ▸ h1)

theorem one_ne_two : one ≠ two := fun h =>
  have h2 : one ∈' two := mem_upair.mpr ⟨isSet_one, Or.inr rfl⟩
  have h3 : one ∈' one := h.symm ▸ h2
  Null_ne_one ((mem_sing.mp h3).2.symm)

/-- The tag `{{{a}}, n}`. -/
def tag (a n : Class) : Class := upair (sing (sing a)) n
/-- 3.45. -/
def markC (A n : Class) : Class :=
  cls (fun x => x = upair Null n ∨ ∃ a, a ∈' A ∧ x = tag a n)
/-- 3.46: the Dunaev ordered pair. -/
def dpair (A B : Class) : Class := bUnion (markC A one) (markC B two)
/-- 3.47. -/
def dpr1 (L : Class) : Class := cls (fun a => tag a one ∈' L)
def dpr2 (L : Class) : Class := cls (fun a => tag a two ∈' L)

theorem isSet_tag {a n : Class} (ha : isSet a) (hn : isSet n) :
    isSet (tag a n) := pairing (isSet_sing (isSet_sing ha)) hn

theorem mem_markC {x A n : Class} :
    x ∈' markC A n ↔ isSet x ∧
      (x = upair Null n ∨ ∃ a, a ∈' A ∧ x = tag a n) := mem_cls _ _

/-! Disambiguation kit: tags, markers and numerals never collide. -/
```

```
theorem sing_sing_ne_one {a : Class} (ha : isSet a) : sing (sing a) ≠ c
  fun h =>
    have h1 : sing a ∈' sing (sing a) := self_mem_sing (isSet_sing ha)
    have h2 : sing a ∈' one := h ▸ h1
    have h3 : sing a = Null := (mem_sing.mp h2).2
    not_mem_Null a (h3 ▸ self_mem_sing ha)

theorem sing_sing_ne_two {a : Class} (ha : isSet a) : sing (sing a) ≠ t
  fun h =>
    have h1 : Null ∈' two := mem_upair.mpr ⟨isSet_Null, Or.inl rfl⟩
    have h2 : Null = sing a := (mem_sing.mp (h.symm ▸ h1)).2
    not_mem_Null a (h2.symm ▸ self_mem_sing ha)

/-- A tag `{{{a}},n}` with `n ≠ ∅` is never a marker `{0,m}`. -/
theorem tag_ne_marker {a n m : Class} (ha : isSet a) (hn : n ≠ Null) :
    tag a n ≠ upair Null m :=
  fun h =>
    have h1 : Null ∈' upair Null m := mem_upair.mpr ⟨isSet_Null, Or.inl
    have h2 : Null ∈' tag a n := h.symm ▸ h1
    (mem_upair.mp h2).2.elim
      (fun e =>
        have h3 : sing a ∈' Null := e.symm ▸ self_mem_sing (isSet_sing
        not_mem_Null _ h3)
      (fun e => hn e.symm)

theorem tag_one_ne_tag_two {a c : Class} (ha : isSet a) (hc : isSet c)
    tag a one ≠ tag c two :=
  fun h =>
    have h1 : one ∈' tag a one := mem_upair.mpr ⟨isSet_one, Or.inr rfl⟩
    have h2 : one ∈' tag c two := h ▸ h1
    (mem_upair.mp h2).2.elim
      (fun e =>
        have h0 : Null ∈' one := self_mem_sing isSet_Null
        have h3 : Null ∈' sing (sing c) := e ▸ h0
        have h4 : Null = sing c := (mem_sing.mp h3).2
        not_mem_Null c (h4.symm ▸ self_mem_sing hc))
      one_ne_two

theorem tag_inj {a c n : Class} (ha : isSet a) (hc : isSet c)
    (hn : sing (sing a) ≠ n) (h : tag a n = tag c n) : a = c :=
  have h1 : sing (sing a) ∈' tag a n :=
    mem_upair.mpr ⟨isSet_sing (isSet_sing ha), Or.inl rfl⟩
  have h2 : sing (sing a) ∈' tag c n := h ▸ h1
  (mem_upair.mp h2).2.elim
    (fun e => sing_inj ha (sing_inj (isSet_sing ha) e))
    (fun e => absurd e hn)
```

```
    (fun e => absurd e hm))

theorem mem_dpair {x A B : Class} :
    x ∈' dpair A B ↔ isSet x ∧ (x ∈' markC A one ∨ x ∈' markC B two) :=
  mem_cls _ _

theorem tag1_mem_dpair {a A B : Class} (ha : a ∈' A) :
    tag a one ∈' dpair A B :=
  have ht : isSet (tag a one) := isSet_tag (isSet_of_mem ha) isSet_one
  mem_dpair.mpr ⟨ht, Or.inl (mem_markC.mpr ⟨ht, Or.inr ⟨a, ha, rfl⟩⟩)⟩

theorem tag2_mem_dpair {b A B : Class} (hb : b ∈' B) :
    tag b two ∈' dpair A B :=
  have ht : isSet (tag b two) := isSet_tag (isSet_of_mem hb) isSet_two
  mem_dpair.mpr ⟨ht, Or.inr (mem_markC.mpr ⟨ht, Or.inr ⟨b, hb, rfl⟩⟩)⟩

theorem mem_A_of_tag1 {a A B : Class} (ha : isSet a)
    (h : tag a one ∈' dpair A B) : a ∈' A :=
  (mem_dpair.mp h).2.elim
    (fun h1 => (mem_markC.mp h1).2.elim
      (fun e => absurd e (tag_ne_marker ha (Ne.symm Null_ne_one)))
      (fun ⟨a', ha', he⟩ =>
        (tag_inj ha (isSet_of_mem ha') (sing_sing_ne_one ha) he).symm ▸
    (fun h2 => (mem_markC.mp h2).2.elim
      (fun e => absurd e (tag_ne_marker ha (Ne.symm Null_ne_one)))
      (fun ⟨b, hb, he⟩ =>
        absurd he (tag_one_ne_tag_two ha (isSet_of_mem hb))))

theorem mem_B_of_tag2 {b A B : Class} (hb : isSet b)
    (h : tag b two ∈' dpair A B) : b ∈' B :=
  (mem_dpair.mp h).2.elim
    (fun h1 => (mem_markC.mp h1).2.elim
      (fun e => absurd e (tag_ne_marker hb (Ne.symm Null_ne_two)))
      (fun ⟨a, ha, he⟩ =>
        absurd he.symm (tag_one_ne_tag_two (isSet_of_mem ha) hb)))
    (fun h2 => (mem_markC.mp h2).2.elim
      (fun e => absurd e (tag_ne_marker hb (Ne.symm Null_ne_two)))
      (fun ⟨b', hb', he⟩ =>
        (tag_inj hb (isSet_of_mem hb') (sing_sing_ne_two hb) he).symm ▸

/-- **The TBI items, verified.**  First projection recovers the first
    component of a Dunaev pair — for ARBITRARY classes… -/
theorem dpr1_dpair (A B : Class) : dpr1 (dpair A B) = A :=
  extAx (fun x =>
    (fun h =>
      have h' := (mem_cls _ x).mp h
```

```
      mem_A_of_tag1 h'.1 h'.2,
     fun h => (mem_cls _ x).mpr ⟨isSet_of_mem h, tag1_mem_dpair h⟩⟩)

/-- …and the second likewise. -/
theorem dpr2_dpair (A B : Class) : dpr2 (dpair A B) = B :=
  extAx (fun x =>
    ⟨fun h =>
      have h' := (mem_cls _ x).mp h
      mem_B_of_tag2 h'.1 h'.2,
     fun h => (mem_cls _ x).mpr ⟨isSet_of_mem h, tag2_mem_dpair h⟩⟩)

/-- Hence the Dunaev pair is injective on ARBITRARY classes (proper inc
    — exactly what Kuratowski pairs cannot deliver (`kpair_phantom`). -
theorem dpair_inj {A B C D : Class} (h : dpair A B = dpair C D) :
    A = C ∧ B = D :=
  ⟨by rw [← dpr1_dpair A B, h, dpr1_dpair],
   by rw [← dpr2_dpair A B, h, dpr2_dpair]⟩

/-- Showcase: it works where Kuratowski provably fails — on proper clas
theorem dpair_proper_showcase :
    dpr1 (dpair V On) = V ∧ dpr2 (dpair V On) = On :=
  ⟨dpr1_dpair V On, dpr2_dpair V On⟩

/-- The `{0,n}` marker device works: a mark is never empty, so list len
    is always recoverable — in particular `[A] ≠ []` even for `A = ∅`.
theorem mark_ne_Null (A n : Class) (hn : isSet n) : markC A n ≠ Null :=
  fun h =>
    have hm : upair Null n ∈' markC A n :=
      mem_markC.mpr ⟨pairing isSet_Null hn, Or.inl rfl⟩
    not_mem_Null _ (h ▸ hm)

theorem dsingle_ne_dnil (A : Class) : markC A one ≠ Null :=
  mark_ne_Null A one isSet_one

end
end Obelisk

/-! ### Audit: exact axiom dependencies of the main results -/
#print axioms Obelisk.printed_inconsistent_route1
#print axioms Obelisk.printed_inconsistent_route2
#print axioms Obelisk.printed_3_65_refutable
#print axioms Obelisk.no_unique_partner
#print axioms Obelisk.thm_3_18
#print axioms Obelisk.thm_3_55
#print axioms Obelisk.thm_3_56
#print axioms Obelisk.thm_3_58
```

```
#print axioms Obelisk.thm_3_67
#print axioms Obelisk.dpair_inj
#print axioms Obelisk.dsingle_ne_dnil
```

Отчёт подготовлен Claude (Anthropic) 22 июля 2026 по просьбе Алексея Комиссарова. Все формальные утверждения проверены Lean 4.32.0; файл самодостаточен (без Mathlib) и компилируется без ошибок и без sorry. Источники: obeliskproofassistant.github.io → georgydunaev.github.io, репозиторий georgydunaev/variousreleases (output.html/output.pdf, 2026-05-05).