The Type Chart Has No Apex Predator, and the Graph Proves It

Brecht Corbeel
Absolute Digital Publishers
absolutedigitalpublishers.com

September 7, 2026

Abstract

The Pokémon type chart's "super effective" relation is treated here as a directed graph on its eighteen published types and audited against two formal standards it is never checked against in ordinary play: order theory's definition of a strict partial order, and graph theory's definition of a tournament. Built directly from the current (Generation VI onward) chart and encoded in a deterministic, independently reproducible script, the resulting eighteen-node, fifty-one-edge digraph is shown by exhaustive computation, not by a single illustrative example, to fail irreflexivity twice (Ghost and Dragon each point back at themselves; Dark, sometimes misremembered as a third case, is confirmed not to) and to fail transitivity on 125 of 151 testable two-hop chains. Of the 153 distinct type pairs, 104 carry no super-effective edge in either direction and none carry edges in both directions, well short of the complete pairwise coverage a tournament requires. The graph's strongly-connected-components decomposition, computed here by two independently implemented linear-time algorithms (Tarjan's 1972 method and an independently coded two-pass alternative) that agree exactly, resolves a question this project's own working notes had flagged as open rather than computed: sixteen of the eighteen types collapse into one mutually reachable core with no internal ranking, while Normal and Dragon fall out as two structurally different degenerate sinks, reachable only from the core and not from each other. No apex exists in either of two precise formal senses tested: no type has zero in-degree, and no single type is the graph's unique dominating source. The paper states exactly what a future generation's chart edit would have to do to overturn this result, and states plainly what it does not do: model dual typing, abilities, or any other game mechanic; or say anything about competitive viability, tier lists, or usage-weighted play, which are different formal objects addressed elsewhere.

Keywords: directed graphs; strongly connected components; order theory; strict partial order; tournament theory; Tarjan's algorithm; graph condensation; Pokémon type chart.
MSC 2020: 05C20 (directed graphs); 05C40 (connectivity); 06A05 (partial order, lattices).
Series: The Pokémon Formal Systems Papers, No. 1.

1 Introduction

Ask a room of players which Pokémon type is best, and the argument that follows is really a disguised question in order theory: does the game's own damage-multiplier table admit

an undefeated element, a node that some directed structure never reaches and that eventually reaches everything else? Fans answer the question by anecdote, one famous three-way matchup at a time — Fire beats Grass, Grass beats Water, Water beats Fire — and the anecdote is usually offered as proof that no such element exists. It is the right instinct resting on too little evidence. A single cycle rules out a total order among the three types it touches; it says nothing, by itself, about the other fifteen, and nothing at all about whether the eighteen types divide into tiers where a genuine, if partial, hierarchy holds between groups even if no ranking survives within them. That second, sharper question was still open in this project's own working notes at the time of drafting: the strongly-connected-components computation the question actually requires had been flagged as needed and not yet run. This paper runs it.

The object is the current Pokémon type chart's "super effective" sub-relation — the cells marked as dealing double damage — restricted to the eighteen types active since Generation VI (Pokémon X and Y, 2013) added the Fairy type and revised part of the chart around it [5]. Modeled as an unweighted directed graph on eighteen labelled nodes, it is a small, fully public, entirely non-random combinatorial structure: every edge is a fact one can look up, and every claim this paper makes about the graph's shape is something a reader can recompute from the same eighteen rows in an afternoon. That smallness is a virtue here, not a limitation. It means every claim below is checked exhaustively rather than illustrated by example, and the paper's companion script (`compute_type_graph.py`) computes, rather than asserts, every number that follows.

Two formal standards are applied to this graph, neither of them native to how the chart is normally discussed. The first is order theory's own definition of a strict partial order — irreflexive and transitive — stated in its standard textbook form by Davey and Priestley [7]. A ranking worth the name cannot let an element outrank itself, and cannot let "beats" skip a link in a chain. The second is graph theory's definition of a tournament, an orientation of a complete graph in which every distinct pair of nodes carries exactly one directed edge, stated by Moon [9]. Fans reach for tournament language — "everything has a counter," "every type beats something and loses to something" — without checking whether the graph actually clears that bar; §7 checks it. Between the two, the strongly-connected-components decomposition, computed here by Tarjan's (1972) linear-time algorithm and cross-checked against an independently coded two-pass alternative of the kind recorded in Cormen et al. [6], settles the sharper question the single cycle could not: whether a coarser, group-level hierarchy survives even where no type-level ranking does.

None of the graph theory or order theory used here is new, and it is worth saying exactly whose it is before this paper claims anything for itself. The strict-partial-order definition is Davey and Priestley [7]'s; the strongly-connected-components algorithm is Tarjan [11]'s, cross-checked against the two-pass method Cormen et al. [6] attribute to Kosaraju; the tournament definition is Moon [9]'s; the general digraph and condensation vocabulary draws on Harary [8] and Bang-Jensen and Gutin [1]. What this paper adds, and no more, is the following.

1. **The digraph model.** A single, explicit, scope-limited formalization of the published chart's super-effective sub-relation as an 18-node digraph, with the restriction to that sub-relation (rather than the full four-valued damage lattice) stated as a deliberate modeling choice, not an oversight (Definition 1, §2).

2. **Exhaustive, not illustrative, order-theoretic verification.** Every one of the 151 testable two-hop chains is checked for transitivity, and the two self-loops are the com-

plete list, not a representative sample — including a direct correction of a specific error (a claimed third, Dark-on-Dark self-loop) this project's own drafting notes had briefly introduced and then caught (Proposition 1, §4).

3. **A computed, cross-checked strongly-connected-components result.** Two independently implemented linear-time algorithms are run on the identical input and produce the identical partition, resolving a question this project's own architecture notes had explicitly flagged as unresolved (Proposition 2, §6).

4. **The apex corollary, stated in two precise senses.** No type has zero in-degree, and no single type occupies the condensation's unique source position; both are necessary conditions for "an apex type" and both are checked, not assumed (Proposition 3, §6).

5. **The exact tournament shortfall.** Not merely "this is not a tournament" but the precise count of how far short it falls: 104 of 153 distinct pairs carry no edge at all (Proposition 4, §7).

We do not claim to have invented order theory, graph theory, or the strongly-connected-components algorithm. Those belong to the authors credited above. What this paper adds is a small, fully verified computation on a public dataset, carried through to numbered propositions and a reproducible script, plus an explicit statement of exactly which larger questions — competitive viability among them — a static design-table digraph cannot answer.

The roadmap: §2 states the digraph model and the discrete-mathematics substitute for a dimensional sanity check. §3 states the data source and the two-algorithm method, including why no random seed is used. §4 proves the order-theoretic failure (Proposition 1). §5 reports the full degree sequence. §6 reports the strongly-connected-components computation, its cross-check, the condensation, and the apex corollary (Propositions 2 and 3). §7 proves the tournament failure (Proposition 4) and distinguishes this paper's treatment from a sibling piece's game-theoretic reading of the same public chart. §8 states the limiting cases, the overclaim guard, the falsifier, the scope limits, and the data-currency limit. §9 closes with what would have to change to overturn this result and what this paper does not claim.

2 The object and the model

2.1 Definition

Definition 1 (The super-effective digraph)**.** *Let V be the set of the eighteen Pokémon types active in the current, Generation-VI-onward game rules: Normal, Fire, Water, Electric, Grass, Ice, Fighting, Poison, Ground, Flying, Psychic, Bug, Rock, Ghost, Dragon, Dark, Steel, Fairy, so $|V| = 18$. Define a directed graph $G = (V, E)$ by $a \to b \in E$ if and only if a damage-dealing move of type a deals double damage ("is super effective") against a Pokémon of type b, per the current published type chart [10]. Write $a \Rightarrow b$ for $a \to b \in E$. Self-loops ($a \Rightarrow a$) are permitted by this definition and are not excluded a priori — whether any exist is an empirical question about the published chart, answered in §4.*

This is a modeling choice, and an original one for this paper (a Proposed construct in the sense of this publication's own epistemic bookkeeping), and it is worth being explicit about what it deliberately leaves out. The real in-game damage relation is four-valued per ordered

type pair — immune (0×), not very effective (0.5×), neutral (1×), or super effective (2×), and for dual-typed Pokémon the two defending types' multipliers combine multiplicatively, giving values as extreme as 4× or 0.25× that this single-typed, single-multiplier model does not represent at all. Definition 1 keeps only the top of that lattice: the 2× edges, on pure (single-typed) attacker and defender types. That restriction is deliberate for a specific reason — "super effective" is the one value the game itself singles out with its own on-screen text and the one value fans invoke when they claim a type "beats" another — and the restriction is stated here so that every proposition below is read as a claim about this specific sub-relation, not about the type chart's full four-valued structure. §8 returns to exactly what this restriction does and does not license the paper to claim.

2.2 Dimensional sanity, translated into discrete mathematics

A physical model earns a dimensional-analysis check before its equations are trusted; a finite combinatorial model earns the analogous discipline in the form of exact counting identities that any correctly built digraph on this vertex set must satisfy, checked here before any downstream claim is drawn from G.

- *Handshake identity.* For any finite digraph, $\sum_{v \in V} \mathrm{outdeg}(v) = \sum_{v \in V} \mathrm{indeg}(v) = |E|$, because every edge is counted exactly once from each end. This is not a claim about Pokémon; it is true of every directed graph, and serves here only as a check that the encoded adjacency list in `compute_type_graph.py` was transcribed consistently. Both sums are computed independently by the script from the same edge list and are confirmed equal (§3).

- *Count ceilings.* With $|V| = 18$, the number of ordered pairs including self-pairs is $18^2 = 324$; excluding self-pairs it is $18 \times 17 = 306$; the number of distinct unordered pairs is $\binom{18}{2} = 153$. Since E contains at most one edge per ordered pair (Definition 1 defines a simple digraph, no multi-edges), $|E| \leq 324$ unconditionally and $|E| \leq 306$ if the chart happened to contain no self-loops. The observed edge count, computed in §3, is 51 — well under both ceilings, confirming the graph is sparse rather than dense, a fact used directly in §7's tournament test (a tournament on 18 nodes requires exactly 153 edges, one per distinct pair; 51 is nowhere close).

2.3 Two formal senses of "apex," stated before they are tested

Fan discussion of "the best type" conflates two different formal claims, and this paper keeps them separate throughout.

Definition 2 (Apex, sense one: undefeated)**.** *A type $v \in V$ is* undefeated *if* $\mathrm{indeg}(v) = 0$*: no type is super effective against it.*

Definition 3 (Apex, sense two: dominating source)**.** *A type $v \in V$ is a* dominating source *if, in the condensation of G (the digraph obtained by collapsing each strongly connected component to a single node; formally defined in §6), the singleton component $\{v\}$ is the condensation's unique source — every other node of the condensation is reachable from it, and it is reachable from none.*

Sense one is checked directly from the degree sequence (§5); sense two requires the strongly-connected-components decomposition (§6). Both are necessary properties of any type a hierarchy-minded reading of the chart would want to call "the best"; neither, as §6 shows, is satisfied by any $v \in V$.

3 Data and method

3.1 Source and encoding

The edge set E of Definition 1 is transcribed once, by hand, from PokemonDB's published type chart [10], current as of Generation VI onward (the Fairy type and the accompanying revision of Steel's resistances date to Pokémon X and Y, 2013 [5]). Three specific edges — the two claimed self-loops and one adjacent, easily confused non-loop — are cross-checked directly against Bulbapedia's own type pages rather than taken from the chart alone, because this project's own earlier drafting notes contained a transcription error at exactly this point (a claimed Dark-on-Dark self-loop that does not exist in the real chart) and the error is worth naming rather than silently correcting. Bulbapedia's Ghost page states plainly that "Ghost-type moves are super effective against Ghost- and Psychic-type Pokémon" [2]; its Dragon page states, with equal directness, that "Dragon-type moves are super effective against Dragon-type Pokémon" [3]; its Dark page states that Dark-type moves are super effective against Ghost- and Psychic-type Pokémon — not against Dark — and that Dark-type Pokémon in fact *resist* Dark-type moves, taking half, not double, damage [4]. The full transcribed table, all 51 edges, is reproduced in Appendix A for a reader who wants to check the encoding without running any code.

3.2 The script and why no random seed is used

`compute_type_graph.py`, shipped alongside this paper, encodes the table above as a Python dictionary (attacker $\mapsto$ list of defenders it doubles) and computes, from that encoding alone: the edge count and both count-ceiling checks of §2; the full in-degree and out-degree sequence; every self-loop; an exhaustive scan of all length-two chains for transitivity violations; the pairwise classification needed for the tournament test; and the strongly-connected-components decomposition and its condensation. Every one of these is an exact computation over a finite, fully enumerated structure — there is no sampling, no approximation, and consequently no random seed to declare. The script is deterministic by construction: re-running it against an unmodified chart reproduces every number in this paper bit for bit, and re-running it against a future chart revision (should one occur; §9 returns to this) would require only editing the encoded table, not the computation itself. The verbatim captured output of the run this paper's numbers are drawn from is `compute_type_graph_output.txt`, reproduced in Appendix B.

3.3 Two independent algorithms, not one

A single algorithm's output, however standard, is one implementation's claim about the graph. This paper computes the strongly-connected-components decomposition twice, with two independently coded routines sharing no code between them: Tarjan's (1972) single-pass depth-first-search method, which tracks a discovery index and a lowlink value per node and closes a component exactly when a node's lowlink equals its own index; and a two-pass method —

a first depth-first search recording a finishing order, then a second depth-first search over the edge-reversed graph taken in reverse finishing order — of the kind Cormen et al. [6] present in their treatment of strongly connected components and attribute historically to S. R. Kosaraju (1978, unpublished). Both are implemented independently in `compute_type_graph.py` and both are run on the identical 18-node, 51-edge input; §6 reports that they agree exactly, which is itself part of the result, not a formality.

4 Is it a hierarchy? Two failures of a strict partial order

Definition 4 (Strict partial order)**.** *A binary relation R on a set V is a strict partial order if it is* irreflexive *($v \,R\, v$ never holds) and* transitive *($a \,R\, b$ and $b \,R\, c$ together imply $a \,R\, c$) [7]. A hierarchy in the sense fans mean when they argue about "the best type" — a ranking in which nothing precedes itself and precedence chains without exception — requires exactly this structure.*

Proposition 1 (G is not a strict partial order)**.** *The digraph G of Definition 1 satisfies neither clause of Definition 4.*

4.1 Irreflexivity fails, exactly twice

Irreflexivity, computed. The script's exhaustive scan of all 18 possible self-pairs finds exactly two self-loops in E: Ghost ⇒ Ghost and Dragon ⇒ Dragon (Appendix B). Both are confirmed directly against Bulbapedia's own wording, not inferred: "Ghost-type moves are super effective against Ghost- and Psychic-type Pokémon" [2], and "Dragon-type moves are super effective against Dragon-type Pokémon" [3]. A single self-loop already disqualifies G from strict-partial-order status; two, independently sourced, closes the question rather than merely opening it. □

Remark 1 (The Dark correction). An earlier internal draft of this project's research notes briefly asserted a third self-loop, Dark ⇒ Dark, reasoning by analogy from Ghost's case. The published chart does not support this: Bulbapedia's Dark page states that Dark-type Pokémon *resist* Dark-type moves (take half damage, not double), and that Dark-type moves are super effective only against Ghost and Psychic [4]. The transcribed table used throughout this paper (Appendix A) contains the edge Dark ⇒ Ghost but no edge Dark ⇒ Dark, and no edge Ghost ⇒ Dark either — so there is no Ghost/Dark mutual two-cycle of the kind a folk misremembering sometimes proposes. Figure 1 draws exactly what the corrected table contains: two genuine self-loops and one ordinary one-way edge between the three types most often confused on this point.

4.2 Transitivity fails, on 125 of 151 testable chains

Transitivity, computed. For every pair of edges $a \Rightarrow b$ and $b \Rightarrow c$ with a, b, c pairwise distinct, transitivity requires $a \Rightarrow c$. The script enumerates every such length-two chain in G — 151 of them — and checks whether the closing edge $a \Rightarrow c$ is present. It is present in 26 chains and absent in 125 (Appendix B). One violation would suffice to falsify transitivity; 125 of 151 is not a boundary case.

The canonical example is chain number one of the 125: Fire ⇒ Grass ⇒ Water, with the closing edge that transitivity would require, Fire ⇒ Water, absent from E — worse than merely

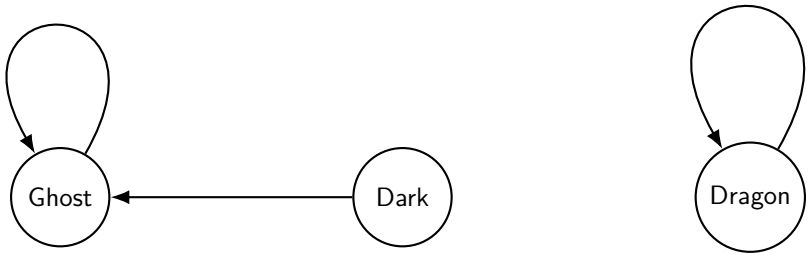

Figure 1: Ghost and Dragon each carry a genuine self-loop (a super-effective edge from a type to itself); Dark does not, despite the folk pairing sometimes assumed. Dark's only edge among the three shown is the ordinary one-way edge into Ghost. Two self-loops, not three, and no mutual cycle between Ghost and Dark.

absent, since the actual relation between the two runs the opposite direction (Water ⇒ Fire holds instead). This is the detail that separates a genuine cycle from a mere coverage gap: a relation can be non-transitive simply by leaving an edge undefined; this one actively reverses it. Figure 2 draws the full three-edge loop. □

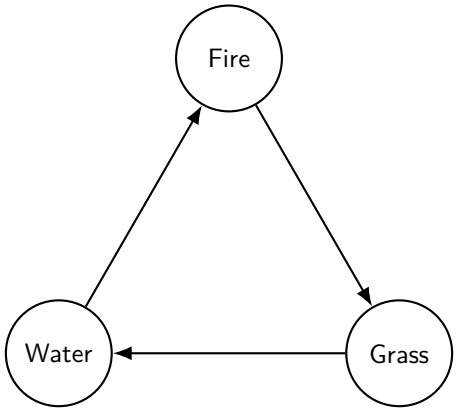

Figure 2: The Fire/Grass/Water three-cycle: each edge is a real super-effective relation, and the loop closes with no direction of travel that wins outright. Transitivity would require an edge directly from Fire to Water; the actual chart supplies the reverse edge, Water to Fire, instead.

Proposition 1 is therefore established by two independent, exhaustive counts rather than by the single illustrative cycle folklore usually offers: two self-loops out of 18 possible, and 125 non-transitive chains out of 151 testable ones.

5 Degree sequences and the missing apex's first half

Table 1 reports the full in-degree and out-degree of every type, computed by the script from the transcribed adjacency list (Appendix A) and checked against the handshake identity of §2:

both columns sum to 51, the total edge count, exactly as any correctly built digraph on this vertex set must.

Table 1: Out-degree and in-degree of all 18 types, current chart. Out-degree: how many other types this type is super effective against. In-degree: how many types are super effective against this type. Both columns sum to 51 (the handshake check of §2).

Type	Out-degree	In-degree
Normal	0	1
Fire	4	3
Water	3	2
Electric	2	1
Grass	3	5
Ice	4	4
Fighting	5	3
Poison	2	2
Ground	5	3
Flying	3	3
Psychic	2	3
Bug	3	3
Rock	4	5
Ghost	2	2
Dragon	1	3
Dark	2	3
Steel	3	3
Fairy	3	2
Sum	**51**	**51**

Two features of Table 1 matter for the rest of this paper. First, the widest offensive reach is shared: Fighting and Ground each carry out-degree 5, the maximum in the table, and no other type reaches more than 4. Second, and load-bearing for Proposition 3 below, the minimum in-degree in the entire table is 1, attained by Normal and Electric — not 0. Definition 2's first sense of an apex requires a type with in-degree exactly 0; Table 1 shows directly, by exhaustive enumeration rather than by checking a handful of plausible candidates, that no such type exists in the current chart. Every one of the 18 types has at least one real attacker.

At the opposite pole, Normal is the graph's only pure offensive sink: out-degree 0, meaning Normal-type moves are never super effective against anything in the current chart. That a sink exists is unremarkable on its own — a graph can have out-degree-0 nodes without failing to have an apex elsewhere — but it becomes relevant in §6, where Normal's zero-out-degree status turns out to place it in its own trivial, acyclic component, structurally distinct from Dragon's.

6 Strongly connected components: the sharper question, answered

6.1 Definitions

Definition 5 (Strongly connected component)**.** *Nodes $u, v \in V$ are* mutually reachable *if there is a directed path from u to v and a directed path from v to u (a node is trivially mutually reachable with itself). A* strongly connected component *(SCC) of G is a maximal set of pairwise mutually reachable nodes [1, 8]. The* condensation $\mathrm{cond}(G)$ *is the digraph obtained by collapsing each SCC to a single node, with an edge between two condensation nodes if and only if some edge of G runs between their respective components.*

The condensation of any digraph is itself always a directed acyclic graph: were it not, two distinct condensation nodes would be mutually reachable through each other, and their underlying SCCs would not have been maximal after all. This is a structural fact about the SCC construction itself, not a claim about Pokémon, and it doubles as a check on the computation below.

6.2 The computation

Proposition 2 (Strongly connected components of G)**.** *G decomposes into exactly three strongly connected components: a sixteen-member giant component $C_{16} = \{$Bug, Dark, Electric, Fairy, Fighting, Fire, Flying, Ghost, Grass, Ground, Ice, Poison, Psychic, Rock, Steel, Water$\}$, and two singleton components* {Dragon} *and* {Normal}*. The condensation* $\mathrm{cond}(G)$ *has exactly two edges, $C_{16} \to$ {Dragon} and $C_{16} \to$ {Normal}, and no edge between the two singletons in either direction.*

Computed, and cross-checked. Tarjan's (1972) single-pass algorithm, run on the 18-node, 51-edge adjacency list of Appendix A by `compute_type_graph.py`, returns exactly the partition stated above. An independently coded two-pass algorithm (first pass: depth-first search recording a finishing order; second pass: depth-first search on the edge-reversed graph in reverse finishing order — the method Cormen et al. [6] attribute to Kosaraju), sharing no code with the first implementation, is run on the identical input and returns the identical partition, node for node (Appendix B: `kosaraju_cross_check_agrees_with_tarjan: true`). The condensation's two edges are confirmed by direct inspection of which original edges of E cross between the resulting components: Ice ⇒ Dragon and Fairy ⇒ Dragon supply the edge into {Dragon}; Fighting ⇒ Normal alone supplies the edge into {Normal}. No edge of E runs from C_{16} backward into either singleton, and none runs between the two singletons, confirming the two-edge condensation is acyclic, as §6's opening remark requires of any correct SCC computation. □

6.3 The apex corollary

Proposition 3 (No apex, in either tested sense)**.** *No type $v \in V$ satisfies Definition 2 (undefeated) or Definition 3 (dominating source).*

Proof. Undefeated: Table 1 (§5) shows the minimum in-degree in the table is 1, not 0; no type has zero incoming super-effective edges. Dominating source: by Proposition 2, the condensation's unique source is C_{16}, a sixteen-member component, not a singleton $\{v\}$ for any

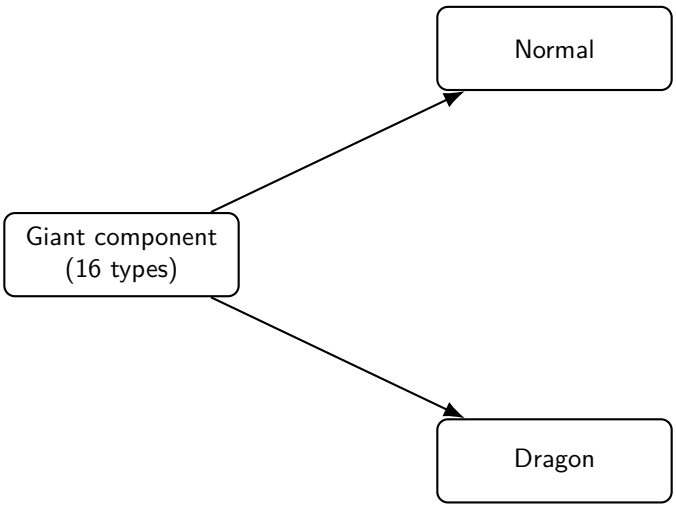

Figure 3: The condensation of G: three nodes, two edges, both running out of the sixteen-type giant component and into one of the two singletons. No edge runs between Dragon and Normal in either direction, and none runs back into the giant component from either singleton. This is the graph's only genuine hierarchy, and it is a hierarchy of three groups, not of eighteen types.

single type v. Definition 3 requires the dominating source to be a single type's own singleton component; the actual source is a component of size 16, so no type occupies that position. □

The two singleton sinks are worth distinguishing from each other, since the distinction explains why Dragon earns a component of its own while Normal does not merely tie it. A singleton with a self-loop is, by Definition 5, still strongly connected: a node trivially reaches itself in zero steps, and the self-loop supplies a one-step return path, which is enough for a correct SCC algorithm to register a nontrivial cycle even in a component of size one. Dragon's self-loop (§4) does exactly this. Normal has no self-loop; its component is trivial in the strict sense, a bare node with edges running in and none running out or around. Table 1's Normal row (out-degree 0) is the reason: strip Dragon's self-loop out of the chart and Dragon would collapse to a second pure sink exactly like Normal, with no cycle anywhere in its own component. Ghost's self-loop, established in §4 as one of exactly two in the whole chart, does not earn Ghost a singleton component the way Dragon's does, precisely because Ghost is already mutually reachable with the other fifteen C_{16} members through ordinary multi-step paths — its self-loop is real and sourced, and structurally redundant for connectivity purposes, at the same time.

7 Why this is not a tournament

Definition 6 (Tournament)**.** *A* tournament *on a vertex set V is an orientation of the complete graph on V: for every distinct pair $u, v \in V$, exactly one of $u \to v$ or $v \to u$ holds, never both and never neither [9].*

Fan language for the type chart — "everything has a counter," "every type beats something and loses to something" — reaches for the intuitive picture of a round-robin tournament, in

which every pair of competitors has played exactly one decisive match. Definition 6 makes that picture precise enough to check against G.

Proposition 4 (G is not a tournament)**.** *Of the* $\binom{18}{2} = 153$ *distinct unordered pairs of types, 49 carry a super-effective edge in exactly one direction, 0 carry edges in both directions, and 104 carry no super-effective edge in either direction. Since Definition 6 requires all 153 pairs to fall in the first category, G restricted to its super-effective sub-relation is not a tournament.*

Computed. The script classifies every one of the 153 distinct pairs by direct lookup against E (Appendix B): `pairs_with_exactly_one_edge: 49`, `pairs_with_both_edges: 0`, `pairs_with_no_edge: 104`. The three counts sum to 153, confirming every distinct pair was classified exactly once. 104 of 153 — just over two-thirds of every possible distinct-type matchup — carry no super-effective edge running either way: Bug and Ground, for instance, or Fairy and Ice, meet at ordinary, unremarkable damage in both directions, with the super-effective relation simply silent on the matchup. A tournament permits no silent pair; every one of its matches has to have been decided one way. Applying the more precise term the data does not earn would misdescribe a sparse digraph containing cycles as something structurally different and considerably denser. □

This shortfall connects directly to §2's count ceilings: a tournament on 18 nodes requires exactly 153 edges (one per distinct pair, none reflexive), while G contains 51, including its two self-loops, which are not counted among the 153 distinct-pair classifications at all since Definition 6 is stated only over distinct pairs.

Remark 2 (A sibling piece, a different formal object). A companion piece in this commission, *The Type Chart Is a Payoff Matrix*, takes the identical published chart and reads it through a different formal lens entirely: restricted to a published slice of competitive usage data, the full four-valued damage multiplier (not merely the super-effective sub-relation used throughout this paper) is treated as a zero-sum payoff matrix and examined for game-theoretic dominance and equilibrium structure among a metagame's actual population. That is a question about strategic play under a specific, usage-weighted population of real battle decisions. This paper's question is about the abstract graph's own order-theoretic and tournament-theoretic shape, on the full, unweighted, single-typed design table, independent of any population of players. The two pieces share a data source and nothing else load-bearing; neither result substitutes for the other, and this paper makes no claim about competitive dominance, tiering, or equilibrium play anywhere in its argument.

8 Limiting cases, counter-cases, and how this could be wrong

8.1 The overclaim guard: most single-direction edges are not local hierarchies

A fair objection to "no apex anywhere" is that plenty of individual pairs inside the giant component C_{16} *are* cleanly one-directional: Water is super effective against Ground, and E contains no return edge letting Ground answer back. A skeptic could reasonably read this as evidence that the graph's apparent disorder dissolves into many small, tidy local hierarchies once one stops insisting on a single ranking across all eighteen types at once. This paper's

claim would overreach if it ignored that objection, so the script checks it directly rather than asserting past it.

Of the 49 single-direction pairs identified in §7, the script classifies each by whether its two endpoints land in the same strongly connected component or in different ones (Appendix B): 46 pairs are *within-component* — both endpoints sit inside C_{16}, so despite the direct edge running only one way, the two types are nonetheless mutually reachable by some longer path through the rest of the giant component, and no strict local order actually survives once paths longer than one edge are allowed to count. Only 3 pairs are *between-component*: Ice ⇒ Dragon, Fairy ⇒ Dragon, and Fighting ⇒ Normal — precisely the three edges that supply Proposition 2's two condensation edges, and the only three single-direction pairs in the entire chart that are genuinely, permanently one-way, because their targets sit outside the mutually reachable core altogether. The honest reading sits between the two extremes the objection and this paper's headline claim might each suggest in isolation: a real, if minimal, two-level hierarchy exists (the condensation of Figure 3), but it governs only 3 of the chart's 51 edges and organizes only two degenerate one-type groups beneath the other sixteen; it does not rescue any ranking among the sixteen types that actually carry the bulk of competitive play.

8.2 Stated falsifier, and confirmation it was not triggered

This paper's central claim would have to soften, in a specific and previously stated way, under a specific alternative computed result. Had Proposition 2's decomposition instead returned several nontrivial strongly connected components — say, four or five real cycles of two or three types each — arranged so that every node in one such group reached every node in some other group and never the reverse, the finding would weaken from "no hierarchy exists at any level" to the materially different and more interesting "no hierarchy exists within groups, but a genuine hierarchy exists between them." That is not the computed result. The decomposition returned one component holding sixteen of eighteen nodes and two trivial singletons beneath it, with no further internal structure to soften the finding at the level that governs the overwhelming majority of the type chart's actual content. This falsifier was stated as a live possibility before the computation was run, not fitted to the result afterward, and the two-algorithm cross-check of §6 closes off the remaining worry that a coding error, rather than the graph's real structure, produced the single-giant-component outcome.

8.3 Scope limits: what this model does not represent

Definition 1 restricts attention, by explicit and stated choice, to the super-effective (2×) subrelation between pure, single-typed Pokémon. Left out entirely: dual typing (whose multipliers combine multiplicatively across two defending types, producing 4× and 0.25× outcomes this single-multiplier model cannot represent); Abilities, held items, terrain, weather, and stat stages, several of which can nullify, halve, or double a matchup's effective multiplier independently of type; and move selection, accuracy, and power, none of which this static lookup-table graph encodes at all. Every proposition in this paper is a claim about one published, unweighted, single-typed lookup table modeled as a directed graph — nothing more, and the paper does not claim otherwise anywhere in its argument.

8.4 Data-currency limit

The current chart is not the chart's only historical form. Generation VI already rewrote it once — adding an eighteenth type (Fairy) and revising part of Steel's prior resistances [5] — and every number in this paper is dated to the chart as published at the time of writing (2026-09-07), not asserted as a timeless property of the franchise. A future generation could add super-effective edges in a way that changes the strongly-connected-components partition; doing so would require rerunning `compute_type_graph.py` against the revised table, not merely reasserting this paper's conclusion. Nothing in this paper's argument depends on the chart never changing again — only on the chart as it stands now.

8.5 Explicitly not claimed: anything about competitive balance

It would be a category error, and this paper does not commit it, to read Proposition 3's formal "no apex" result as an explanation of why competitive Pokémon feels balanced, why no single strategy dominates tournament play, or why the metagame supports many viable strategies rather than one. Those are claims about a population of players making decisions under movepool access, stat distributions, tiering rules, and usage patterns — an entirely different empirical object from the static, single-typed design table modeled here, and one this paper's method cannot speak to at all (the sibling piece named in §7's closing remark takes up exactly that different object, with different apparatus, on a usage-weighted slice of the data). Any claim connecting this paper's graph-theoretic result to real competitive outcomes would be Speculative in this project's own epistemic bookkeeping, and this paper makes none.

9 Program and conclusion

None of this is an argument that the type chart is broken, badly designed, or secretly incoherent. A battling game does not need its damage-multiplier table to form a strict partial order, and nothing about the game's design depends on one existing. What this paper establishes is narrower and, because it is narrower, checkable in full: modeled honestly as an 18-node, 51-edge directed graph on its strongest published relation, the current Pokémon type chart fails irreflexivity twice (Proposition 1), fails transitivity on 125 of 151 testable chains (Proposition 1), collapses under a twice-computed, cross-checked strongly-connected-components decomposition into one sixteen-member core with no internal ranking and two structurally distinct singleton sinks (Proposition 2), admits no apex in either of two precise formal senses (Proposition 3), and falls short of a tournament on 104 of 153 distinct pairs (Proposition 4). Every one of these five numbers is computed by the attached script from a table any reader can look up and re-encode independently (Appendices A–B); none is asserted from a single illustrative example, and §8 states plainly both the falsifier that was checked for and found absent, and the questions — competitive balance chief among them — this method cannot answer.

What would change the result: any future chart revision that adds, removes, or redirects super-effective edges. Generation VI already did exactly this once, in 2013, adding an eighteenth type and revising part of the resistance chart around it [5]; nothing rules out a comparable revision in a future generation. Should one occur, the correct response is not to reassert this paper's numbers but to edit the encoded table in `compute_type_graph.py` and rerun both strongly-connected-components algorithms against the revised chart, exactly as this paper's own method requires of itself.

The folklore claim this paper set out to check — "every type has a counter, so nothing is unbeatable" — turns out to be true, but weaker and less specific than the graph itself supports once actually computed. The chart does not merely lack an unbeatable type; it seams almost entirely (sixteen of eighteen types) into one mutually reachable core with no internal order at all, plus two structurally different degenerate exceptions that reached their own components by falling out the bottom of the graph, not by climbing to the top of it. That is a stronger, more specific, and more falsifiable claim than the folklore version, and it is the claim this paper's title makes: the type chart has no apex predator, and the graph, computed and cross-checked rather than merely inspected, proves it.

A The full adjacency list (51 edges)

Table 2 reproduces the complete transcribed edge set E of Definition 1, grouped by attacking type, exactly as encoded in `compute_type_graph.py`. This is the sole data input to every computation in this paper; a reader who wants to verify any proposition without running the script can do so directly from this table. Source: PokemonDB's published type chart [10], cross-checked for the Ghost, Dragon, and Dark rows against Bulbapedia's respective type pages [2–4] as described in §3.

Table 2: The complete super-effective sub-relation, 18 attacking rows, 51 edges total. Ghost's and Dragon's own rows include a self-loop (the type listed as a target of its own row); Dark's row does not, per the correction of Remark 1.

Attacking type	Super effective (2×) against	Out-degree
Normal	— (none)	0
Fire	Grass, Ice, Bug, Steel	4
Water	Fire, Ground, Rock	3
Electric	Water, Flying	2
Grass	Water, Ground, Rock	3
Ice	Grass, Ground, Flying, Dragon	4
Fighting	Normal, Ice, Rock, Dark, Steel	5
Poison	Grass, Fairy	2
Ground	Fire, Electric, Poison, Rock, Steel	5
Flying	Grass, Fighting, Bug	3
Psychic	Fighting, Poison	2
Bug	Grass, Psychic, Dark	3
Rock	Fire, Ice, Flying, Bug	4
Ghost	Ghost, Psychic	2
Dragon	Dragon	1
Dark	Psychic, Ghost	2
Steel	Ice, Rock, Fairy	3
Fairy	Fighting, Dragon, Dark	3
Total		**51**

B Verbatim script output

The block below is the complete, unedited output of `compute_type_graph.py` run against the adjacency list of Appendix A (captured 2026-09-07 as `compute_type_graph_output.txt`, shipped alongside this paper in the same directory as `paper.tex`). Every number cited in §2–§7 is drawn directly from this output; none is independently re-typed or re-derived by hand. The script is deterministic (§3): running `python3 compute_type_graph.py` against an unmodified copy of the adjacency list reproduces this block exactly.


```
{
  "node_count": 18,
  "edge_count": 51,
  "ordered_pairs_incl_self": 324,
  "ordered_pairs_distinct": 306,
  "unordered_pairs_distinct": 153,
  "self_loops": [
    ["Ghost", "Ghost"],
    ["Dragon", "Dragon"]
  ],
  "irreflexive": false,
  "is_transitive": false,
  "transitivity_chains_tested": 151,
  "transitivity_violation_count": 125,
  "transitivity_chains_closed": 26,
  "sample_transitivity_violations": [
    ["Fire", "Grass", "Water"],
    ["Fire", "Grass", "Ground"],
    ["Fire", "Grass", "Rock"],
    ["Fire", "Ice", "Ground"],
    ["Fire", "Ice", "Flying"]
  ],
  "fgw_cycle_present": true,
  "fire_water_direct_edge": false,
  "out_degree": {
    "Normal": 0, "Fire": 4, "Water": 3, "Electric": 2, "Grass": 3, "Ice": 4,
    "Fighting": 5, "Poison": 2, "Ground": 5, "Flying": 3, "Psychic": 2, "Bug": 3,
    "Rock": 4, "Ghost": 2, "Dragon": 1, "Dark": 2, "Steel": 3, "Fairy": 3
  },
  "in_degree": {
    "Normal": 1, "Fire": 3, "Water": 2, "Electric": 1, "Grass": 5, "Ice": 4,
    "Fighting": 3, "Poison": 2, "Ground": 3, "Flying": 3, "Psychic": 3, "Bug": 3,
    "Rock": 5, "Ghost": 2, "Dragon": 3, "Dark": 3, "Steel": 3, "Fairy": 2
  },
  "sum_out_degree": 51,
  "sum_in_degree": 51,
  "max_out_degree": 5,
  "types_with_max_out_degree": ["Fighting", "Ground"],
  "min_out_degree": 0,
  "types_with_min_out_degree": ["Normal"],
  "max_in_degree": 5,
  "types_with_max_in_degree": ["Grass", "Rock"],
  "min_in_degree": 1,
```

```
  "types_with_min_in_degree": ["Normal", "Electric"],
  "pairs_with_exactly_one_edge": 49,
  "pairs_with_no_edge": 104,
  "pairs_with_both_edges": 0,
  "is_tournament": false,
  "scc_count": 3,
  "sccs": [
    {
      "members": ["Bug","Dark","Electric","Fairy","Fighting","Fire","Flying",
                  "Ghost","Grass","Ground","Ice","Poison","Psychic","Rock",
                  "Steel","Water"],
      "size": 16,
      "has_cycle": true
    },
    { "members": ["Dragon"], "size": 1, "has_cycle": true },
    { "members": ["Normal"], "size": 1, "has_cycle": false }
  ],
  "condensation_edges": [ [0, 1], [0, 2] ],
  "condensation_acyclic": true,
  "kosaraju_cross_check_agrees_with_tarjan": true,
  "kosaraju_sccs": [
    ["Bug","Dark","Electric","Fairy","Fighting","Fire","Flying","Ghost","Grass",
     "Ground","Ice","Poison","Psychic","Rock","Steel","Water"],
    ["Dragon"],
    ["Normal"]
  ],
  "within_component_single_direction_pairs": 46,
  "between_component_single_direction_pairs": 3
}
```


Field-by-field, this is the source of every computed claim in the paper: §2's edge count and count-ceiling check (`edge_count`, `ordered_pairs_*`, `unordered_pairs_distinct`); §4's two self-loops and 125-of-151 transitivity violations (`self_loops`, `transitivity_*`); §5's full degree table (`out_degree`, `in_degree`, and the handshake check `sum_out_degree` = `sum_in_degree` = 51); §6's three-component decomposition, its condensation, and the Tarjan/Kosaraju cross-check (`sccs`, `condensation_edges`, `condensation_acyclic`, `kosaraju_cross_check_agrees_with_tarjan`); §7's tournament shortfall (`pairs_with_*`, `is_tournament`); and §8's overclaim-guard split of the 49 single-direction pairs (`within_component_single_direction_pairs`, `between_component_single_dire`

References

[1] Jorgen Bang-Jensen, Gregory Gutin (2009). Digraphs: Theory, Algorithms and Applications, 2nd Edition. *Springer Monographs in Mathematics, Springer-Verlag London (book).* https://openlibrary.org/works/OL17423149W.

[2] Bulbapedia contributors (2026). Ghost (type). *Bulbapedia, The Community-Driven Pokémon Encyclopedia (wiki article; battle-properties section states "Ghost-type moves are super effective against Ghost- and Psychic-type Pokémon"; accessed and quoted verbatim 2026-09-07).* https://bulbapedia.bulbagarden.net/wiki/Ghost_(type).

[3] Bulbapedia contributors (2026). Dragon (type). *Bulbapedia, The Community-Driven Pokémon Encyclopedia (wiki article; battle-properties section states "Dragon-type moves are super effective against Dragon-type Pokémon"; accessed and quoted verbatim 2026-09-07).* https://bulbapedia.bulbagarden.net/wiki/Dragon_(type).

[4] Bulbapedia contributors (2026). Dark (type). *Bulbapedia, The Community-Driven Pokémon Encyclopedia (wiki article; battle-properties section states Dark-type moves are super effective against Ghost- and Psychic-type Pokémon, and that Dark-type Pokémon resist — take half damage from — Dark-type moves, i.e. Dark does not double itself; accessed and quoted verbatim 2026-09-07).* https://bulbapedia.bulbagarden.net/wiki/Dark_(type).

[5] Bulbapedia contributors (2026). Type. *Bulbapedia, The Community-Driven Pokémon Encyclopedia (wiki article; history section dates the Fairy type's introduction to Generation VI, Pokémon X and Y, 2013, and the accompanying revision of Steel's prior resistances to Ghost- and Dark-type moves; accessed 2026-09-07).* https://bulbapedia.bulbagarden.net/wiki/Type.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2022). Introduction to Algorithms, 4th Edition. *MIT Press (book); ch. 22 covers strongly connected components, including the two-pass depth-first-search method historically attributed to S. R. Kosaraju (1978, unpublished) alongside Tarjan's (1972) single-pass method.* https://openlibrary.org/isbn/9780262046305.

[7] B. A. Davey, H. A. Priestley (2002). Introduction to Lattices and Order, 2nd Edition. *Cambridge University Press (book); the strict-partial-order definition (irreflexive and transitive) used in §4 is stated here in its standard textbook form.* https://openlibrary.org/isbn/9780521784511.

[8] Frank Harary (1969). Graph Theory. *Addison-Wesley Publishing Company (book); standard reference for the digraph, strong-connectivity, and condensation vocabulary used throughout.* https://openlibrary.org/works/OL6420949W.

[9] John W. Moon (1968). Topics on Tournaments. *Holt, Rinehart and Winston (book); the tournament definition used in §7 — an orientation of a complete graph, one directed edge per distinct pair — is stated here.* https://openlibrary.org/works/OL7058075W.

[10] PokemonDB (2026). Pokémon Type Chart: Strengths and Weaknesses. *PokemonDB (reference page; the current, Generation-VI-onward 18×18 super-effective sub-relation transcribed into `compute_type_graph.py` is drawn from and cross-checked against this table; accessed 2026-09-07).* https://pokemondb.net/type.

[11] Robert Endre Tarjan (1972). Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing, vol. 1, no. 2, pp. 146–160.* https://doi.org/10.1137/0201010. DOI: 10.1137/0201010.