The History of Spherepop: From Language Games to Universal Computation

Flyxion

January 11, 2026

Abstract

This essay traces the conceptual lineage of Spherepop from its philosophical origins in Wittgenstein's theory of language games through its mathematical equivalence to classical foundations of computation. We argue that Spherepop emerges naturally once meaning is treated as an activity constrained by scope, order, and irreversibility rather than as a static mapping between symbols and referents. By examining the role of parentheses in arithmetic precedence, abstraction in lambda calculus, and control in Turing machines, we show that Spherepop is not an alternative to these systems but a re-expression of their common core under an explicitly historical and event-based ontology.

1. Language Games and the Primacy of Use

The conceptual origin of Spherepop lies not in computation but in philosophy of language, specifically in Ludwig Wittgenstein's later work on language games (Wittgenstein 1953). Wittgenstein rejected the idea that meaning is grounded in reference to abstract objects or mental representations. Instead, he argued that meaning arises from use within rule-governed practices. To understand a word is to understand how it functions within an activity, not to grasp an invisible semantic object that accompanies it.

This shift has a crucial structural consequence. If meaning is use, then meaning is inherently temporal. Utterances occur in sequences, follow rules, and alter what may coherently follow them. Language games are therefore not static systems but unfolding histories of permissible moves. Once a move is made, the space of future moves is constrained. This is the first and most important step toward Spherepop: meaning is not a state but an event.

Wittgenstein's language games already imply scope. A word or expression has meaning only within a particular practice, and that practice defines what counts as a valid continuation. This notion of contextual enclosure is the philosophical ancestor of Spherepop's nested bubbles. Each bubble corresponds to a local game, and evaluation consists in resolving that game according to its internal rules before re-entering a broader context.

2. Parentheses and the First Step of PEMDAS

The same structure appears in elementary arithmetic, long before formal computation is introduced. Consider the order of operations summarized by PEMDAS. The first rule is not multiplication or addition but parentheses. Parentheses establish scope. They determine which sub-expressions must be resolved before others can be meaningfully combined.

This is not merely a pedagogical convention. Without parentheses, arithmetic expressions are ambiguous. The expression $1 + 2 \times 3$ has a conventional reading, but $(1 + 2) \times 3$ forces a different evaluation. The parentheses create a local context that must be resolved as a unit. Once resolved, the internal structure disappears, leaving behind a value that constrains subsequent operations.

This is already a pop. The interior of the parentheses is evaluated once, irreversibly, and cannot be re-entered without reintroducing the entire expression. The result is authoritative. Arithmetic proceeds by constructing nested scopes and collapsing them in a well-defined order. Spherepop makes this structure explicit and generalizes it beyond numbers.

3. Abstraction and Application in Lambda Calculus

Lambda calculus formalizes this scoping principle and elevates it to the foundation of computation (Church 1936). A lambda abstraction introduces a local variable scope, and application resolves that scope by substituting an argument and reducing the expression. The reduction step is again irreversible. Once a beta-reduction is performed, the original expression is no longer present except implicitly through its effect on the result.

The essential operation of lambda calculus is not symbolic substitution but scope discharge. An abstraction defines a region of meaning. Application enters that region, performs a computation, and exits with a result. The internal structure of the abstraction does not remain available unless it is explicitly re-encoded.

Spherepop recognizes this pattern and strips it of its syntactic clothing. Instead of treating abstraction as a textual construct, Spherepop treats it as a geometric or historical enclosure. A sphere corresponds to a lambda abstraction. A pop corresponds to application. The result is not a symbolic value but a constrained future in which certain distinctions have been resolved and others remain open.

From this perspective, lambda calculus is a linearized projection of Spherepop. It preserves the nesting and collapse structure while discarding the explicit history of how scopes were entered and resolved. Spherepop retains that history as a first-class object.

4. Turing Machines and the Authority of the Past

Turing machines appear, at first glance, to depart from scoping in favor of global state (Turing 1936). However, their defining feature is not the tape but the irreversible advance of time. Each transition consumes an instruction and moves the machine into a new configuration. Although the tape may be

revisited, the computation itself is a sequence of committed steps.

The authority of a Turing computation lies in its history. The halting configuration is meaningful only because of the path that led to it. The machine cannot undo a transition without explicitly encoding that reversal. In practice, computation proceeds by accumulating irreversible commitments.

Spherepop makes this implicit structure explicit. Instead of modeling computation as state transition, it models it as event accumulation. Each pop is a committed act that alters the space of possible futures. The tape of a Turing machine becomes an optional representation, not the foundation. What matters is the sequence of constraints imposed by past actions.

Under this interpretation, Turing machines are another shadow of the same underlying principle: computation is the disciplined reduction of possibility through irreversible steps.

5. Spherepop as the Unifying Perspective

Spherepop emerges when these observations are taken seriously and unified. From Wittgenstein, it inherits the idea that meaning is use within constrained practices. From arithmetic, it inherits the primacy of scope and evaluation order. From lambda calculus, it inherits abstraction and collapse. From Turing machines, it inherits irreversibility and the authority of history.

What Spherepop adds is explicitness. It refuses to treat scope as syntactic sugar, reduction as mere rewriting, or history as an implementation detail. Instead, it places events at the center of the formalism. Worlds are constructed by popping nested spheres of possibility. Identity is defined by shared histories. Computation is the progressive elimination and binding of futures.

In this sense, Spherepop is not a new computational model competing with classical ones. It is a clarification of what those models have always been doing implicitly. It restores time, commitment, and scope to their rightful place at the foundation of meaning.

6. A General Framework

The history of Spherepop is therefore not a chronological story of invention but a conceptual unfolding. Once meaning is understood as an activity governed by scope, order, and irreversibility, the move to an event-based calculus becomes unavoidable. Parentheses, abstractions, and machine steps are all instances of the same act: the creation and collapse of local worlds.

Spherepop names this act, formalizes it, and refuses to let it disappear behind notation. In doing so, it offers not only a model of computation but a general framework for understanding how meaning, agency, and structure arise in finite worlds.

7. Comparison with Category Theory

Category theory occupies a unique position in the foundations of mathematics and computation (Mac Lane 1971). Rather than describing systems in terms of elements, states, or instructions, it describes them in terms of relations and composition. Objects are defined only by how morphisms connect them,

and meaning is located in the structure of these connections rather than in internal representation. For this reason, category theory is often invoked as a unifying language across logic, computation, and geometry.

At first glance, Spherepop appears closely aligned with this perspective. Both frameworks reject element-centric ontology. Both treat composition as fundamental. Both emphasize structure over representation. Indeed, many constructions in Spherepop admit natural categorical descriptions, and several of its operators can be expressed using categorical machinery. However, the two approaches diverge at a deeper ontological level, particularly with respect to time, irreversibility, and authority.

Category theory is fundamentally atemporal. A category describes what compositions are possible, not when or how they occurred. Morphisms compose associatively, but the theory does not distinguish between alternative histories that lead to the same composite arrow. From the categorical perspective, only the existence of a morphism matters, not the sequence of commitments that produced it. As a result, category theory excels at describing equivalence, invariance, and abstraction, but it deliberately brackets the question of historical cost.

Spherepop, by contrast, is explicitly historical. Two constructions that yield isomorphic structures may nevertheless be distinct if they arise from different sequences of events. This distinction is not cosmetic. In Spherepop, meaning is inseparable from the path taken. Irreversible commitments constrain the future, and those constraints accumulate even when their internal detail is later collapsed. Where category theory identifies objects up to isomorphism, Spherepop distinguishes worlds by their histories unless an explicit collapse operator is applied.

This difference becomes especially clear when considering quotienting and abstraction. In category theory, quotient constructions are universal and timeless. Once an equivalence relation is imposed, the quotient object simply exists. In Spherepop, collapse plays a similar role but with an additional semantic constraint: collapse is an event that occurs at a specific point in history. It does not erase the past but compresses it. The quotient carries the weight of prior commitments even as descriptive complexity is reduced. Abstraction therefore has a cost and a timing, not merely a universal property.

From a computational perspective, category theory often serves as a metalanguage. It describes the structure of computation without itself being the computation. Monads, functors, and adjunctions organize effects, but they do not execute them. Spherepop occupies a different role. It is not a metalanguage about computation but a calculus of computation itself, one in which execution is identified with the irreversible transformation of option spaces. Categorical structure may be used to analyze Spherepop histories, but those histories remain primary.

There is also a difference in how locality is treated. In category theory, locality is implicit in the domain and codomain of morphisms, but it does not carry spatial or temporal force. Spherepop treats locality as a concrete constraint. Nested spheres define regions of evaluation whose resolution is mandatory before re-entry into a broader context. This makes scope not just a typing discipline but an operational fact. The evaluation order is not an artifact of strategy but a consequence of geometric inclusion.

Despite these differences, the relationship between the two frameworks is not antagonistic. Spherepop histories may be organized into categories, with morphisms corresponding to admissible trans-

formations between histories and collapse policies inducing functors between levels of description. Conversely, category theory provides a powerful lens for understanding the invariants preserved by Spherepop operations. What Spherepop insists upon, however, is that such categorical views are derived rather than foundational.

In this sense, Spherepop can be seen as complementary to category theory. Category theory excels at describing what structures exist and how they relate. Spherepop explains how those structures come to be, what they cost, and why they constrain the future. One speaks the language of equivalence; the other speaks the language of commitment. Together, they outline a fuller picture of meaning and computation than either can provide alone.

8. Comparison with Type Theory

Type theory occupies a central position in contemporary foundations of logic and computation, serving both as a theory of programs and as a theory of proofs. In its various forms, from simple typed lambda calculus to dependent and homotopy type theory, it seeks to control meaning by constraining what expressions may be formed and how they may be combined. Types function as guarantees of coherence, ensuring that certain classes of errors are rendered impossible by construction.

Spherepop shares this concern with coherence, but it approaches the problem from a fundamentally different direction. In type theory, coherence is established a priori through judgments that precede execution. An expression is either well-typed or it is not, and only well-typed expressions are admitted into the space of computation. In Spherepop, coherence is established a posteriori through irreversible events. A world is not required to be coherent in advance; it becomes coherent by paying the cost of commitment.

This difference reflects a deeper divergence in how possibility is treated. Type theory assumes a static universe of admissible constructions defined by typing rules. Computation is the exploration of this universe under reduction. Spherepop assumes a dynamic universe of possibilities that is progressively reduced through action. Computation is not exploration but construction. Possibilities do not merely fail to type; they are actively excluded, bound, or collapsed as events occur.

The treatment of abstraction illustrates this contrast. In type theory, abstraction introduces a parameterized expression whose validity is guaranteed for all admissible arguments of a given type. The abstraction itself does not commit to any particular instance. In Spherepop, abstraction corresponds to the creation of a local sphere of possibility. This sphere is not a promise of generality but a region awaiting resolution. Commitment occurs when the sphere is popped, at which point internal distinctions are discharged and the future is constrained. Generality, in this sense, is not assumed but earned through repeated compatible collapses.

Dependent type theory intensifies the role of context by allowing types to depend on values. This brings type theory closer to Spherepop's emphasis on locality and scope. However, even here the dependency is logical rather than historical. A dependent type records conditions under which an expression is valid, but it does not record the sequence of events by which those conditions were established. Spherepop insists on this distinction. Two expressions that satisfy the same conditions

may nevertheless differ in meaning if they arise from different histories of commitment.

The Curry–Howard correspondence further clarifies the divergence between proof and computation (Church 1936). Under Curry–Howard, programs are proofs and types are propositions. Computation is proof normalization, and termination corresponds to logical consistency. Spherepop does not reject this view, but it reinterprets it. Proof is not normalization toward a timeless truth, but the accumulation of irreversible constraints that make certain futures impossible. A proof, in Spherepop terms, is not something one has, but something one has done.

From an operational perspective, type systems are designed to prevent certain actions from occurring. They rule out invalid states before execution begins. Spherepop, by contrast, allows actions to occur and records their consequences. Errors are not type violations but historical facts. Responsibility is not enforced by exclusion from the language but by the authority of the event log. This makes Spherepop particularly suited to domains in which meaning arises through interaction, negotiation, and commitment rather than through static correctness.

Despite these differences, type theory remains an invaluable analytical tool for Spherepop. Typing judgments can be understood as derived views that summarize which histories are compatible with which future actions. Collapse policies may induce effective type systems by identifying histories that behave equivalently with respect to certain interactions. In this way, types emerge as abstractions over event histories rather than as primitive constraints.

The comparison therefore reveals a complementary relationship. Type theory excels at describing what may be constructed without error. Spherepop explains how construction becomes binding, how meaning accumulates over time, and why coherence must sometimes be enforced after the fact rather than before it. One governs possibility by prohibition; the other governs it by commitment. Together, they illuminate different aspects of what it means to compute in a finite world.

9. Operational Semantics, Reduction, and Resource Sensitivity

Operational semantics provides a formal account of computation by specifying how programs execute step by step. In small-step semantics, computation is described as a transition system in which each reduction transforms a configuration into a successor configuration. Meaning is identified with the totality of possible execution traces, and correctness is established by showing that these traces satisfy desired properties such as termination, confluence, or progress.

Spherepop shares with operational semantics a commitment to execution rather than mere denotation. Both frameworks insist that meaning emerges through action, not solely through static interpretation. However, they diverge sharply in how they treat time, reversibility, and the status of intermediate configurations. In operational semantics, reductions are typically treated as abstract relations. A configuration may be rewritten, replaced, or forgotten without semantic consequence so long as the transition relation is respected. The focus is on reachability, not on the authority of the path taken.

Spherepop instead treats each reduction-like act as an irreversible event. A pop is not merely a transition from one configuration to another but a commitment that permanently alters the space of

admissible futures. Intermediate configurations are not disposable scaffolding; they constitute the history that gives the final configuration its meaning. Two executions that reach the same apparent outcome may nevertheless be distinct if they arrive there through different sequences of pops, refusals, or bindings.

This difference reframes the notion of small-step reduction. In Spherepop, there is no neutral small step. Every step has cost, and every step carries semantic weight. Reduction is not a means of simplifying an expression but an act of world construction. The usual equivalence between small-step and big-step semantics therefore appears not as a theorem but as a choice of collapse policy: one may identify histories that differ only in their internal sequencing, but such identification is itself an explicit event rather than a background assumption.

The contrast becomes even clearer when considering resource-sensitive and linear logics. Linear logic was introduced to correct a blind spot in classical logic and type theory: the assumption that resources may be duplicated or discarded freely (Girard 1987). By tracking usage explicitly, linear logic restores a notion of cost and consumption to formal reasoning. Proofs become processes that must account for how assumptions are spent.

Spherepop can be seen as extending this insight from logical resources to historical ones. Where linear logic tracks how propositions are consumed, Spherepop tracks how possibilities are consumed. A pop corresponds to a non-reusable expenditure of optionality. Refusal records deliberate non-use. Binding constrains future use without immediate consumption. Collapse compresses prior expenditures without negating them. The calculus thus generalizes resource sensitivity from syntax to ontology.

Unlike linear logic, however, Spherepop does not enforce resource discipline through typing rules alone. Resource sensitivity is not checked before execution; it is incurred during execution. A system may overcommit, exhaust its optionality, or collapse distinctions too early. These are not type errors but historical outcomes. The calculus does not prevent such paths; it records them. Responsibility is enforced not by prohibition but by irreversibility.

This perspective clarifies the relationship between correctness and consequence. In traditional operational semantics, an incorrect program is one that gets stuck or violates a specification. In Spherepop, incorrectness is reframed as regret: a history that leads to an impoverished or incoherent future. The system does not roll back such histories; it learns from them by treating their consequences as binding.

Operational semantics and linear logic thus appear, from the Spherepop perspective, as partial recognitions of a deeper principle. Computation is not merely the transformation of symbols under rules, but the irreversible expenditure and organization of possibility. What these frameworks enforce syntactically or relationally, Spherepop enforces temporally.

By making irreversibility explicit and unavoidable, Spherepop unifies execution, cost, and meaning into a single formal object: the history of events. Operational semantics describes how computation can proceed. Resource-sensitive logic constrains how it should proceed. Spherepop explains why proceeding at all changes what remains possible.

10. Denotational Semantics, Event Sourcing, and Historical Computation

Denotational semantics approaches computation by assigning mathematical objects to programs. Rather than describing how a program executes step by step, it specifies what a program means by mapping it into a semantic domain such as a lattice, a function space, or a domain-theoretic structure. Two programs are equivalent if they denote the same object, regardless of how they compute it. This perspective has been enormously influential in establishing rigor, compositionality, and abstraction in programming language theory.

Spherepop shares with denotational semantics a rejection of operational detail as the sole source of meaning. Both frameworks seek compositional explanations that transcend low-level execution traces. However, they diverge sharply in how they treat history. Denotational semantics deliberately erases execution order, intermediate states, and irreversibility. The denotation of a program is timeless. It exists independently of how, when, or even whether the program is run.

Spherepop rejects this erasure. In Spherepop, there is no timeless denotation that stands apart from history. Meaning is not what a computation converges to but what it commits to along the way. Two histories that yield indistinguishable outcomes may nevertheless differ in semantic weight because they impose different constraints on the future. Denotation, in this sense, is not a primitive but a derived collapse: an identification of histories that behave equivalently with respect to some chosen horizon of concern.

This distinction becomes particularly important when considering partiality and nontermination. Denotational semantics typically models nontermination as a bottom element or as divergence in a domain. Spherepop models nontermination as a lived fact: a history that continues to consume optionality without resolving certain distinctions. The difference is subtle but consequential. One treats divergence as an abstract value; the other treats it as an ongoing expenditure of possibility.

The practical implications of this perspective are visible in systems that already operate historically, even if they are not usually described in formal semantic terms. Event-sourced systems, append-only logs, and version control systems all treat history as authoritative. Changes are recorded as events rather than as overwrites of state. The current state is a view derived from replaying those events, not a fundamental object.

Spherepop can be understood as a formalization of this intuition at the semantic level. An event-sourced system already embodies the idea that meaning lies in what has been done, not merely in what is currently the case. Rollbacks are simulated by new events rather than by erasure. Branches preserve alternative futures without negating the past. Merges identify histories that can be coherently unified. These operations closely mirror Spherepop's pop, bind, refuse, and collapse operators.

Version control systems provide an especially vivid analogy. A commit is an irreversible act that constrains future development. Even when changes are reverted, the fact of the commit remains part of the history. Squashing commits resembles collapse: internal distinctions are compressed, but their cumulative effect persists. Rebasing rewrites local views while leaving the authoritative history intact. These systems succeed precisely because they respect the authority of the past while allowing

abstraction over it (Stonebraker and Pavlo 2018).

What Spherepop adds is a unifying semantics for these practices. Instead of treating event sourcing and version control as engineering techniques, it treats them as instances of a more general principle: that computation in finite worlds must be historical. Denotational views, operational traces, and resource analyses all become perspectives layered atop an underlying event history.

From this standpoint, denotational semantics is not rejected but repositioned. It becomes a powerful tool for summarizing and reasoning about histories once they have been constructed. Its abstractions are indispensable for proof and analysis. But the act of computation itself remains irreducibly temporal. Meaning is something that happens, not something that merely is.

Spherepop therefore bridges a long-standing gap between formal semantics and real-world practice. This view aligns closely with the treatment of computation as structured data flow rather than opaque execution (Meijer 2011; Meijer 2012). It explains why systems built around logs, commits, and events feel more robust, auditable, and meaningful than systems built around mutable state. They align, perhaps unintentionally, with the deeper structure of computation as historical commitment. By making this structure explicit, Spherepop provides a foundation on which both theory and practice can meet without contradiction.

11. Databases, Relational Algebra, and Phenomenological Time

Databases and relational algebra provide another instructive comparison, because they occupy a boundary zone between abstract semantics and practical computation. Classical relational theory treats data as a set of relations that may be queried, joined, projected, and updated. The mathematical elegance of relational algebra lies in its declarative nature: queries specify what is desired, not how it is obtained. In its pure form, relational algebra is timeless. A relation simply exists, and queries are mappings over that existence.

Yet real databases are never purely relational. They must account for insertion, deletion, update, transaction ordering, and failure. As soon as these realities are acknowledged, time enters the picture. Transactions occur in sequences. Constraints are enforced incrementally. Rollback and recovery depend on logs rather than on idealized relations. The database ceases to be merely a mathematical object and becomes a historical artifact.

Spherepop can be understood as formalizing this tension. The relational view corresponds to a collapsed abstraction over history, a snapshot derived from accumulated events. The event log, by contrast, corresponds to the authoritative Spherepop history. Queries are views. Joins are derived compositions. Updates are not overwrites but irreversible events that constrain future states. What relational algebra treats as a primitive relation, Spherepop treats as a stabilized consequence of prior commitments.

This distinction clarifies long-standing difficulties in database theory surrounding temporal databases, auditability, and consistency. Attempts to graft time onto relational systems often introduce ad hoc mechanisms such as timestamps, version tables, or bitemporal schemas. Spherepop avoids this layering problem by reversing the ontology. Time is primary. Relations are secondary. A

relation is meaningful precisely because it summarizes a history that has already occurred.

The comparison extends beyond engineering into philosophy. Phenomenological accounts of time, particularly those concerned with agency and responsibility, emphasize that the present is shaped by irrevocable past acts. One cannot undo a decision; one can only act in light of it. Meaning arises not from the mere availability of options but from the fact that some options are no longer available. The past exerts authority.

This phenomenological structure mirrors Spherepop exactly. A Spherepop world is not defined by what could happen but by what can still happen given what has already happened. The future is conditioned by accumulated exclusions, bindings, and collapses. This is not a defect of finitude but its defining feature. To act is to narrow the future, and to narrow the future is to give it shape.

Seen from this angle, Wittgenstein's language games appear not merely as social practices but as temporal commitments. To make a move in a language game is to foreclose certain replies and invite others. Grammar is not a static system of rules but a sedimented history of use. Spherepop generalizes this insight into a formal calculus. Grammar becomes history. Meaning becomes consequence.

Databases, programming languages, logical systems, and lived experience thus converge on the same underlying structure. Systems that deny the authority of history must simulate it indirectly, through logs, constraints, or recovery mechanisms. Systems that embrace history directly can reason about responsibility, auditability, and coherence without contradiction.

Spherepop belongs to this latter class. It is not merely a computational model but a theory of finite action. It explains why meaning cannot be fully captured by states, denotations, or relations alone. Meaning is something that has happened, something that cannot unhappen, and something that continues to matter precisely because it is irreversible.

In closing the arc from Wittgenstein to computation and back to lived time, Spherepop reveals a unifying insight. Language, arithmetic, programs, databases, and human agency are all structured by the same constraint: that to do something is to change what can be done next. Spherepop names this constraint, formalizes it, and treats it not as a limitation but as the ground of meaning itself.

12. Circles of Evaluation and a Minimal Implementation

An especially direct way to understand Spherepop is through the notion of circles of evaluation, a pedagogical device commonly used to teach arithmetic and functional programming. In this representation, expressions are drawn as nested circles, with an operator at the center and its arguments arranged around it. Evaluation proceeds from the innermost circles outward. Each circle is resolved as a unit, and once resolved, its internal structure disappears, replaced by a value that participates in a larger context.

This visual device is not merely instructional. It encodes the same structural commitments that underlie Spherepop. This emphasis on visual structure echoes broader arguments for geometric and diagrammatic reasoning in mathematics (Needham 1997). Each circle defines a local scope of meaning. The interior must be resolved before the exterior can proceed. Evaluation is irreversible in the sense that once a circle has been evaluated, the internal distinctions that produced the result are no longer

available unless the entire expression is reconstructed. The circle is popped.

Spherepop generalizes this idea by treating circles not as static syntax but as event-scoped regions of possibility. A circle of evaluation corresponds to a sphere. Evaluating the circle corresponds to a pop. The enclosing circle receives the result, but not the internal history, unless that history is explicitly preserved as a distinct object. What elementary arithmetic teaches visually, Spherepop elevates into a general calculus of meaning and computation.

This correspondence becomes especially clear in languages such as Racket, whose syntax already exposes evaluation structure through parentheses. A parenthesized expression is already a circle of evaluation. Racket's evaluator resolves nested expressions in a disciplined order, producing values that replace their originating expressions. By making this process explicit and treating it as an event rather than a rewrite, we obtain a minimal Spherepop-like core.

The following implementation demonstrates this idea. It does not aim to reproduce the full Spherepop calculus. Instead, it shows how nested scopes, irreversible evaluation, and history can be modeled with a small amount of code.

```
#lang racket

;; A sphere is represented as a delayed computation with a label.
(struct sphere (label thunk) #:transparent)

;; A pop evaluates the sphere exactly once and records the event.
(define (pop s history)
  (define value ((sphere-thunk s)))
  (values value
          (append history
                  (list (cons (sphere-label s) value)))))

;; Example spheres corresponding to circles of evaluation
(define s1
  (sphere 'inner
          (lambda () (+ 1 2))))

(define s2
  (sphere 'outer
          (lambda ()
            (define-values (v h)
              (pop s1 '()))
            (* v 3))))

;; Running the outer pop
(define-values (result history)
  (pop s2 '()))

(result)
(history)
```

In this fragment, each sphere encloses a computation. The inner sphere corresponds to an inner circle of evaluation. Popping it produces a value and records an event. The outer sphere uses that value but does not regain access to the internal computation except through the history log. Evaluation is monotone and irreversible. Once a sphere is popped, its effect is carried forward, not revisited.

Although minimal, this example already exhibits the essential structure. Parentheses induce nesting. Evaluation proceeds inward to outward. Each pop produces a value and a trace. History is explicit rather than implicit. If two spheres were popped in different orders or under different bindings, the resulting histories would differ even if the final numeric result were the same.

This is precisely the distinction Spherepop insists upon. Circles of evaluation teach that structure matters. Spherepop adds that history matters as well. By unifying the visual intuition of circles with an explicit event log, Spherepop transforms a familiar pedagogical tool into a foundation for historical computation. This perspective aligns with broader philosophical critiques of overly mechanistic accounts of computation (Fant 1995).

Seen this way, Spherepop does not introduce a foreign idea. It makes concrete what circles of evaluation have always suggested: that meaning emerges by resolving local worlds and carrying their consequences forward, one irreversible step at a time.

The same structural pattern reappears outside of programming languages, most clearly in the analysis of electrical circuits. When computing the equivalent resistance of a circuit, one does not treat the circuit as a flat collection of components. Instead, one identifies subcircuits that may be reduced independently. A series subcircuit is collapsed into a single equivalent resistor. A parallel subcircuit is likewise collapsed according to its own rule. These reductions are performed locally and irreversibly with respect to the larger analysis.

Once a subcircuit has been reduced, its internal structure no longer participates in the computation. The enclosing circuit does not remember how the equivalent resistance was derived; it only inherits the constraint that the subcircuit imposes. The reduction behaves exactly like a pop. The interior distinctions between individual resistors are discharged, and the result becomes authoritative for all subsequent calculations.

Crucially, the order of reduction matters. Different groupings of components correspond to different circles of evaluation. While the final numerical resistance may be invariant under certain reorderings, the intermediate structure determines what reductions are available at each step. Circuit simplification therefore proceeds by constructing and collapsing nested regions of analysis, just as arithmetic and functional evaluation do.

Spherepop makes explicit what circuit theory assumes implicitly. Each reduction is an event that constrains future possibilities. Once a branch has been collapsed into an equivalent resistance, it cannot later be treated as parallel or series with its former internal elements. The history of reduction defines what further simplifications are valid. Circuit analysis is thus a physical instance of historical computation.

An analogous structure appears in shell languages such as Bash, where scope resolution governs how commands are evaluated. Parentheses, subshells, and command substitutions create nested execution contexts. A command executed in a subshell may modify variables, change directories,

or redirect output, but those effects do not propagate outward unless explicitly exported. When the subshell exits, its internal state is discarded, and only its observable result is returned.

This behavior mirrors the semantics of a pop. Entering a subshell creates a local world of possible actions. Executing commands within that world constrains its internal future but does not directly constrain the enclosing shell. When the subshell completes, its internal history collapses into a value, typically an exit status or a stream of text. The enclosing shell receives the result but not the internal state that produced it.

Command substitution provides an especially clear example. In an expression such as `$(command)`, the command is evaluated in a nested context. Its output is captured and substituted into the surrounding command line. The internal steps of the substitution are irretrievable once evaluation completes. Only the emitted value survives. The shell cannot revisit or partially reuse the computation without re-executing it in full.

Once again, this is a circle of evaluation enacted in a different medium. The shell enforces scope boundaries, resolves them inward to outward, and collapses internal structure into authoritative results. Spherepop recognizes this pattern as a general law rather than a language-specific feature. Whether the domain is arithmetic, electrical networks, or process execution, meaning is constructed by isolating local contexts, resolving them, and carrying their consequences forward.

Across these domains, the same principle recurs. Evaluation is not merely the transformation of symbols or signals but the elimination of alternatives. A resistor network, a subshell, or a parenthesized expression all represent a temporary suspension of the wider world. Their resolution is an event that narrows what remains possible. Spherepop unifies these practices by treating such resolutions as first-class semantic acts.

Seen in this light, circles of evaluation are not a teaching aid but a deep structural insight. They reveal that computation, analysis, and action proceed by carving out local worlds and then collapsing them. Spherepop extends this insight by insisting that the collapse is not just structural but historical. What has been resolved continues to matter precisely because it cannot be unresolved.

13. A Formal Unification of Nested Evaluation

The preceding examples may be unified under a single abstract description. Let $\mathcal{O}$ denote an option space, understood as the set of all admissible continuations of a system at a given moment. A local context, whether a parenthesized expression, a subcircuit, or a subshell, corresponds to a subspace $\mathcal{O}' \subseteq \mathcal{O}$ whose internal structure is temporarily insulated from the surrounding world. Evaluation within this context induces a monotone map

$$\pi : \mathcal{O}' \to \overline{\mathcal{O}},$$

where $\overline{\mathcal{O}}$ is a quotient of $\mathcal{O}$ obtained by identifying all internal distinctions of $\mathcal{O}'$ that are no longer relevant after evaluation.

The map π is monotone in the sense that it only removes distinctions; it never introduces new

possibilities. It is also irreversible: there is, in general, no inverse map from $\overline{\mathcal{O}}$ back to $\mathcal{O}'$ without reconstructing the entire prior history. Once applied, π constrains all future evaluation by enforcing the consequences of the resolved context.

In arithmetic, $\mathcal{O}'$ corresponds to the set of possible reductions of a subexpression, and π collapses that space to a single numerical value. In circuit analysis, $\mathcal{O}'$ corresponds to the configuration space of a subnetwork, and π maps it to an equivalent resistance. In shell evaluation, $\mathcal{O}'$ corresponds to the space of possible internal command executions, and π maps it to an exit status or output stream.

In each case, the enclosing system interacts only with the quotient $\overline{\mathcal{O}}$, not with the internal structure that produced it. The evaluation order is therefore governed by inclusion of option spaces, and computation proceeds by successive application of such quotient maps. Spherepop takes this abstract pattern as primitive. A pop is precisely the application of π as an event, and a history is the composition of such monotone quotient maps over time. Meaning arises not from the final quotient alone, but from the irreversible sequence by which these quotients were imposed.

14. Conclusion

This paper has traced the conceptual and formal lineage of Spherepop from its philosophical origins to its computational consequences. Beginning with Wittgenstein's account of language games, we identified meaning as an activity governed by rules, scope, and irreversibility rather than as a static correspondence between symbols and objects. That shift immediately placed time and commitment at the center of semantics, establishing the conditions under which a historical calculus of meaning becomes not optional but necessary.

From this foundation, we followed the same structural pattern through elementary arithmetic, where parentheses establish local contexts that must be resolved before further combination is possible. What appears in PEMDAS as a convention of evaluation order was shown to be a primitive act of world construction: the creation and collapse of nested scopes. Lambda calculus refined this pattern into abstraction and application, and Turing machines rendered it operational as irreversible sequences of steps whose authority derives from their history. In each case, computation proceeded not by free exploration of possibility but by its disciplined reduction.

The comparison with category theory, type theory, operational and denotational semantics clarified what Spherepop both inherits and rejects. These frameworks provide indispensable tools for abstraction, correctness, and equivalence, but they systematically treat history as secondary or eliminable. Spherepop reverses this priority. It treats events as primary, abstractions as compressions of history, and equivalence as something that must be enacted rather than assumed. Coherence is not guaranteed in advance; it is achieved by paying the cost of commitment.

By extending the analysis to circuits, shell evaluation, databases, and version control, the paper demonstrated that Spherepop is not an idiosyncratic formalism but an explicit articulation of a structure already present in successful real-world systems. Wherever local contexts are resolved and their consequences carried forward irreversibly, Spherepop's core operators are already at work. The calculus does not impose a foreign discipline; it names an existing one and makes it explicit.

The formal unification of these examples revealed a common mathematical core: computation as a sequence of monotone quotient maps over nested option spaces. Each evaluation step collapses internal distinctions, constrains future action, and contributes irreversibly to a history. Meaning, on this view, is neither a static denotation nor a transient execution trace, but the accumulated consequence of having acted.

Spherepop therefore offers a unifying perspective on meaning, computation, and agency in finite worlds. It explains why scope matters, why history cannot be ignored, and why irreversibility is not an implementation detail but the ground of significance itself. To compute, to speak, or to act is to change what can happen next. Spherepop provides a formal language for this fact, treating commitment not as a limitation to be minimized but as the substance from which meaning is made.

Appendix A: Spherepop as a History-Indexed Rewrite System

This appendix presents a minimal formalization of Spherepop as a rewrite system indexed by history rather than by state. The purpose is not to specify an implementation, but to make explicit the sense in which Spherepop replaces state-based semantics with event-based semantics.

Let Σ be a set of syntactic forms, and let $\mathscr{H}$ be the set of all finite histories, where a history $h \in \mathscr{H}$ is a finite sequence of events. We write $h \cdot e$ for the history obtained by appending event e to history h.

A Spherepop configuration is a pair (σ, h), where $\sigma \in \Sigma$ is a current form and $h \in \mathscr{H}$ is the history of committed events.

Evaluation is given by a partial function

$$\mathrm{pop} \;:\; \Sigma \times \mathscr{H} \rightharpoonup \Sigma \times \mathscr{H},$$

such that

$$\mathrm{pop}(\sigma, h) = (\sigma', h \cdot e)$$

for some event e recording the collapse of a local context in σ.

The defining constraint of the calculus is monotonicity of history. If

$$(\sigma_0, h_0) \xrightarrow{\mathrm{pop}} (\sigma_1, h_1) \quad \text{and} \quad (\sigma_1, h_1) \xrightarrow{\mathrm{pop}} (\sigma_2, h_2),$$

then h_0 is a prefix of h_1, and h_1 is a prefix of h_2. No transition removes or rewrites history.

Rewrite equivalence is therefore history-relative. Two forms σ and τ are definitionally equivalent only relative to a collapse policy

$$\sim_C \subseteq\; \Sigma \times \Sigma$$

that explicitly identifies histories whose event sequences may be quotient-collapsed.

In the absence of such a collapse, equivalence is intensional:

$$(\sigma, h) \not\equiv (\tau, h') \quad \text{unless} \quad h = h'.$$

This distinguishes Spherepop from classical rewriting systems, in which rewrite sequences are treated as auxiliary and equivalence is defined extensionally over normal forms. In Spherepop, normal forms do not erase the history that produced them, and rewriting is an act of commitment rather than substitution.

Appendix B: Nested Scope as a Poset of Option Spaces

This appendix formalizes the notion of nested scope used throughout the paper as a partially ordered structure of option spaces. The aim is to make precise the sense in which evaluation order is governed by inclusion rather than by syntactic convention.

Let $\mathcal{O}$ be a set of option spaces. An element $O \in \mathcal{O}$ represents the set of all admissible continuations

available to a system at a given point in its history.

We equip $\mathcal{O}$ with a partial order $\preceq$ defined by inclusion of admissible futures. For $O_1, O_2 \in \mathcal{O}$, we write

$$O_1 \preceq O_2$$

if every continuation admissible in O_1 is also admissible in O_2. Intuitively, O_1 is a more constrained context than O_2.

A nested scope structure is a finite poset $(\mathcal{O}, \preceq)$ such that for any two option spaces $O_1, O_2 \in \mathcal{O}$, either $O_1 \preceq O_2$, $O_2 \preceq O_1$, or they are incomparable. Incomparable option spaces correspond to disjoint local contexts.

Evaluation proceeds by selecting a minimal element $O_{\min}$ of the poset, corresponding to an innermost scope. A pop operation is defined as a monotone map

$$\pi \,:\, O_{\min} \to O',$$

where O' is an option space satisfying $O' \preceq O$ for all O such that $O_{\min} \preceq O$.

The effect of π is to remove $O_{\min}$ from the poset and replace all occurrences of it in upper bounds with O'. The resulting poset $(\mathcal{O}', \preceq')$ has strictly fewer elements than $(\mathcal{O}, \preceq)$.

Termination of evaluation corresponds to the existence of a unique maximal element $O_\top$, representing the fully collapsed option space. No further pops are admissible once this element is reached.

This formalization abstracts the common structure underlying parentheses in arithmetic, subcircuits in electrical networks, subshells in process execution, and nested spheres in Spherepop. In all cases, evaluation order is determined by the partial order of option-space inclusion rather than by linear sequencing alone.

Appendix C: Collapse as Quotient with Memory

This appendix formalizes the notion of collapse as an explicit quotient operation that preserves historical information. The purpose is to distinguish Spherepop collapse from classical quotienting, in which prior structure is erased rather than compressed.

Let $(\mathcal{O}, \preceq)$ be a poset of option spaces as defined in Appendix B. Let $O \in \mathcal{O}$ be an option space selected for collapse. A collapse policy is specified by an equivalence relation

$$\equiv_C \subseteq\ O \times O$$

that identifies distinctions within O deemed no longer semantically relevant.

The collapse operation induces a quotient space

$$\overline{O}\ :=\ O/{\equiv_C},$$

together with a canonical projection

$$q \,:\, O \to \overline{O}.$$

Unlike classical quotient constructions, collapse in Spherepop is indexed by history. Formally, a collapse event is a triple

$$c := (O, \equiv_C, h),$$

where $h \in \mathscr{H}$ is the history prefix at which the collapse occurs.

The application of a collapse event extends the history:

$$h' = h \cdot c,$$

and replaces O in the ambient poset with $\overline{O}$, ordered so that

$$\overline{O} \preceq O' \quad \text{whenever} \quad O \preceq O'.$$

Crucially, the projection q is not invertible, but it is recorded. The history retains the information that $\overline{O}$ arose as a quotient of O under $\equiv_C$, even though the distinctions identified by $\equiv_C$ are no longer available for future computation.

Two quotient spaces $\overline{O}_1$ and $\overline{O}_2$ are considered extensionally equivalent only if they arise from identical quotient relations applied at identical history prefixes. Otherwise, they are intensionally distinct despite being isomorphic as sets.

This construction formalizes the claim that collapse reduces descriptive complexity without erasing responsibility. Collapse compresses the past but does not annul it. Meaning is carried forward as constrained possibility together with an irreducible record of how that constraint was imposed.

Appendix D: Evaluation as a Functor from Histories to Views

This appendix formalizes the distinction between authoritative history and derived observation by treating evaluation as a functor from histories to views. The goal is to make precise the sense in which observation in Spherepop is non-causal and non-authoritative.

Let $\mathscr{H}$ be the category of histories. Objects of $\mathscr{H}$ are finite histories $h = e_1 \cdot e_2 \cdots e_n$. There exists a unique morphism

$$h \to h'$$

if and only if h' is obtained by appending a finite sequence of events to h. Composition is given by concatenation, and identities are given by empty extensions. The category $\mathscr{H}$ is therefore a thin category encoding prefix order.

Let $\mathscr{V}$ be a category of views. Objects of $\mathscr{V}$ are representational structures such as graphs, tables, summaries, or numerical projections. Morphisms in $\mathscr{V}$ are view-preserving transformations that do not introduce new semantic commitments.

An evaluation policy is defined as a functor

$$F : \mathscr{H} \to \mathscr{V}.$$

For a given history h, the object $F(h)$ represents the view obtained by replaying h according to a fixed interpretation rule. For a morphism $h \to h'$, functoriality requires that

$$F(h \to h') : F(h) \to F(h')$$

be a view update induced purely by the additional events in h'. No view may influence the structure of $\mathscr{H}$ itself.

Different functors $F_1, F_2 : \mathscr{H} \to \mathscr{V}$ may yield distinct views of the same history. These differences correspond to alternative projections, summaries, or analyses. However, all such functors are observational. None introduce new morphisms in $\mathscr{H}$, and none may alter existing histories.

A collapse policy corresponds to a natural transformation between functors

$$\eta : F \Rightarrow G$$

that identifies distinctions in the view space without modifying the underlying history. Such transformations are irreversible at the level of $\mathscr{V}$ but leave $\mathscr{H}$ intact.

This formalization captures the core architectural discipline of Spherepop. History is authoritative and monotone. Views are derived and replaceable. Evaluation is not execution but interpretation. Computation occurs in $\mathscr{H}$; understanding occurs in $\mathscr{V}$. The separation of these roles ensures that observation, abstraction, and summarization can proceed freely without compromising the integrity of the past.

Appendix E: Refusal and Binding as Monotone Constraints

This appendix formalizes refusal and binding as explicit, monotone constraints on option spaces. The aim is to complete the operator set by showing how Spherepop represents non-selection and conditional commitment without reverting to negation or backtracking.

Let $(\mathscr{O}, \preceq)$ be a poset of option spaces as in Appendix B, and let $\mathscr{H}$ be the category of histories as in Appendix D. For a given history $h \in \mathscr{H}$, let $O_h \in \mathscr{O}$ denote the option space admissible at h.

A refusal is an event that excludes a subset of possibilities without selecting an alternative. Formally, a refusal event is a pair

$$r = (O_h, R),$$

where $R \subseteq O_h$ is a nonempty subset of options to be excluded. The application of r induces a new option space

$$O_{h'} = O_h \setminus R,$$

with $h' = h \cdot r$. By construction, $O_{h'} \preceq O_h$. Refusal is therefore monotone: it strictly reduces the set of admissible continuations without introducing new ones.

A binding is an event that commits to a relation between future options without immediately

collapsing them. Formally, a binding event is a triple

$$b = (O_h, \beta, h),$$

where β is a constraint predicate over O_h specifying conditions that future options must satisfy. The application of b yields a constrained option space

$$O_{h'} = \{o \in O_h \mid \beta(o)\},$$

with $h' = h \cdot b$. Binding thus restricts future admissibility while preserving internal distinctions among remaining options.

Refusal and binding differ from collapse in that they do not identify options with one another. Refusal removes options outright. Binding preserves options but constrains their compatibility with future events. Collapse, by contrast, identifies options and compresses distinctions. All three operations are monotone with respect to $\preceq$, but they impose different kinds of semantic commitment.

Two histories h_1 and h_2 that differ only by the order of refusals and bindings may yield extensionally equivalent option spaces while remaining intensionally distinct. The order of constraint application is therefore semantically significant unless an explicit collapse event identifies them.

This formalization captures a key aspect of Spherepop's treatment of agency. Not all meaningful acts consist in choosing among alternatives. Some acts consist in refusing certain paths, while others consist in binding future actions to shared constraints. By representing these acts as first-class, irreversible events, Spherepop models decision-making as the progressive shaping of possibility rather than as instantaneous selection.

Together with pop and collapse, refusal and binding complete a minimal algebra of historical constraint. Every Spherepop history may be understood as a finite sequence of monotone transformations on option spaces, each narrowing, structuring, or compressing the future. Meaning arises from this sequence itself, not from any single terminal configuration.

Appendix F: Confluence, Divergence, and Regret as Properties of Histories

This appendix introduces formal notions of confluence, divergence, and regret as properties of histories rather than of states or expressions. The aim is to characterize correctness and failure in Spherepop without appealing to backtracking, normalization, or error states.

Let $\mathscr{H}$ be the category of histories defined in Appendix D, and let O_h denote the option space admissible at history h. Two histories h_1 and h_2 are said to be extensionally equivalent at horizon k if their induced option spaces agree up to k further admissible events. Formally, we write

$$h_1 \approx_k h_2$$

if the sets of length-k extensions of h_1 and h_2 are equal.

Confluence is defined relative to a collapse policy. A family of histories $\{h_i\}_{i \in I}$ is confluent if there exists a history h_c and a collapse policy C such that for all $i \in I$,

$$h_i \cdot C \approx_0 h_c.$$

Confluence therefore does not assert that histories are identical, but that they may be made equivalent by an explicit act of identification. In the absence of such a collapse, histories remain distinct even if their futures coincide.

Divergence is the failure of such confluence. A set of histories diverges if no collapse policy exists that identifies them without violating monotonicity of option spaces. Divergence is not an error condition but a fact about incompatibility of commitments. Some differences in history cannot be compressed without loss of admissible futures.

Regret is defined as a property of a single history. A history h exhibits regret if there exists a history h' such that h' is reachable from a common prefix with strictly greater option space, but h' is no longer reachable from h. Formally, regret occurs when there exists a prefix p and histories $h = p \cdot e_1 \cdots e_n$ and $h' = p \cdot e'_1 \cdots e'_m$ such that

$$O_h \prec O_{h'}.$$

Regret is therefore not a logical inconsistency or a runtime error. It is the recognition that a sequence of irreversible commitments has led to a strictly more constrained future than was otherwise possible. Regret cannot be undone; it can only be acknowledged and worked around through further commitments or through collapse that compresses the significance of the regretted distinctions.

This framing replaces classical notions of correctness and failure. A history is not incorrect because it diverges or exhibits regret. It is simply more constrained. Evaluation succeeds whenever it produces a history. The question is not whether the history is valid, but whether its remaining option space aligns with the goals of the agent or system.

By shifting correctness from states to histories, Spherepop accommodates learning, error, and adaptation without retreating to backtracking or exception handling. Systems improve not by undoing the past but by acting coherently in light of it. Confluence, divergence, and regret become descriptive properties of lived computation rather than pathologies to be eliminated.

Appendix G: A Minimal BNF Grammar for Spherepop

This appendix presents a minimal Backus–Naur Form grammar for Spherepop. The grammar is not intended to specify a concrete syntax for any particular implementation, but to capture the abstract structure of Spherepop expressions, events, and histories in a form suitable for formal reasoning.

Let terminal symbols be written in typewriter font and non-terminals in angle brackets.

Lexical Domains

```
<Identifier> ::= letter (letter | digit | "_")*
```

```
<Label>      ::= <Identifier>
<Value>      ::= <Identifier> | <Number>
<Number>     ::= digit+
```

Expressions and Spheres

```
<Expr>   ::= <Value>
           | <Sphere>

<Sphere> ::= "(" <Label> ":" <Expr>* ")"
```

A sphere is a labeled enclosure containing zero or more subexpressions. Nesting induces scope.

Events

```
<Event> ::= <Pop>
          | <Collapse>
          | <Refusal>
          | <Binding>

<Pop>      ::= "pop" "(" <Label> ")"
<Collapse> ::= "collapse" "(" <Label> "," <Equiv> ")"
<Refusal>  ::= "refuse" "(" <Label> "," <Set> ")"
<Binding>  ::= "bind" "(" <Label> "," <Predicate> ")"
```

Auxiliary Structures

```
<Equiv>     ::= "{" <Pair> ("," <Pair>)* "}"
<Pair>      ::= <Value> "~" <Value>

<Set>       ::= "{" <Value> ("," <Value>)* "}"

<Predicate> ::= <Identifier>
```

Equivalence relations, sets, and predicates are treated abstractly. Their internal structure is not specified at the grammatical level.

Histories

```
<History> ::= <Event>*
```

A history is a finite sequence of events. There is no grammatical provision for deletion or reordering.

Configurations

```
<Config> ::= "<" <Expr> "," <History> ">"
```

A configuration pairs a current expression with its authoritative history.

Evaluation

```
<Eval> ::= <Config> "=>" <Config>
```

Evaluation is a relation between configurations induced by the application of a single event.

This grammar makes explicit the central commitments of Spherepop. Expressions are nested. Events are first-class. Histories are monotone. Evaluation produces new configurations by appending events rather than rewriting expressions in place. All further semantic structure is defined relative to this grammar.

References

[1] L. Wittgenstein. *Philosophical Investigations*. Blackwell Publishing, Oxford, 1953.

[2] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[3] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.

[4] K. Fant. *Computer Science Reconsidered: The Challenge of Computers and the Mind*. Addison–Wesley, Reading, MA, 1995.

[5] T. Needham. *Visual Complex Analysis*. Oxford University Press, Oxford, 1997.

[6] E. Meijer. Your mouse is a database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012.

[7] E. Meijer. The duality of computation. *Communications of the ACM*, 54(5):41–47, 2011.

[8] S. Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, New York, 1971.

[9] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[10] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, 1999.

[11] M. Stonebraker and A. Pavlo. What goes around comes around. *Communications of the ACM*, 61(1):16–18, 2018.