ED and Computation: How Symbolic Systems Become Algorithms and Algorithms Become Procedural Architecture

Allen Proxmire
February 2026

Abstract

Computation emerges when symbolic systems become procedural — when symbolic transformations acquire operational structure, when rules become executable, and when execution becomes systematic. Symbolic systems provide formal architectures; computation requires procedural architectures: rule-governed processes that transform symbolic states through explicit, repeatable, and deterministic operations. In the ED ontology, computation is not mechanical at its root. It arises when formal systems become operational, when operations become procedures, and when procedures become algorithmic.

This paper develops the symbolic → computational → algorithmic threshold. It shows how symbolic rules become operations, how operations become procedures, how procedures become reusable, and how reusability becomes algorithmic structure. Computation is presented as the first ED regime where becoming becomes procedural — where the dynamics of transformation are themselves structured, repeatable, and executable. This transition sets the stage for Paper 20, where computation becomes recursive, recursion becomes self-modifying, and self-modifying architectures become the ED regime where becoming becomes autoprocedural.

1. Introduction — Why Computation Is the Next ED Threshold

The moment where formal systems become procedural

Symbolic systems gave minds the ability to manipulate formal structures. They provided discrete, rule-governed architectures in which symbolic forms could be combined, transformed, and interpreted. But symbolic systems alone are static. They define what transformations are possible, but they do not yet define how those transformations unfold. Computation emerges when symbolic systems become procedural — when symbolic transformations acquire operational structure, when rules become executable, and when execution becomes systematic.

In the ED ontology, this transition is not mechanical. It is architectural. Computation arises when formal systems become operational architectures: systems in which symbolic transformations are carried out through explicit sequences of steps. A computation is not a machine. It is a procedural unfolding of symbolic structure — a structured trajectory through symbolic space governed by stable rules.

Symbolic systems provide the formal substrate.
Computation provides the procedural substrate.

Computation emerges when:

- symbolic rules become operations
- operations become repeatable
- repeatability becomes determinacy
- determinacy becomes executability
- executability becomes procedure

This is the architecture of computational becoming — the threshold where formal systems acquire motion.

Computation is not a departure from symbolic reasoning.
It is its proceduralization.

It allows minds and machines to:

- execute symbolic transformations step by step
- stabilize complex procedures
- build reusable algorithmic motifs
- coordinate symbolic operations across time
- generate procedural structures that exceed the capacity of any single symbolic system

Computation is the first ED regime where becoming becomes procedural — where the dynamics of transformation are themselves structured, repeatable, and executable.

In this paper, we develop the symbolic → computational → algorithmic threshold. We show how symbolic systems become operational, how operations become procedures, and how procedures become algorithmic architectures capable of supporting executable inference, procedural explanation, and algorithmic abstraction. Computation is presented as the first ED regime where formal becoming becomes procedural becoming.

Symbolic systems gave the universe formal worlds.
Computation gives it procedural worlds.

2. From Formal Systems to Computational Processes

How symbolic transformations become operations

Formal systems provide structured symbolic architectures — systems of rules, constraints, and combinatorial possibilities. But formal systems alone are inert. They specify what transformations are allowed, but they do not yet specify how those transformations unfold. Computation begins when formal systems become operational: when symbolic transformations acquire explicit steps, when rules become executable, and when execution becomes systematic.

In the ED ontology, computation is not the mechanical application of rules. It is the proceduralization of formal structure — the moment when symbolic transformations become operations, when operations become repeatable, and when repeatability becomes determinacy. Computation emerges when symbolic systems gain the capacity to do what they describe.

Computation begins when symbolic transformations acquire four properties:

- operationalization — rules become executable steps
- repeatability — steps can be applied reliably
- determinacy — steps produce predictable outcomes
- executability — steps can be carried out by agents or machines

These properties transform formal systems into computational processes — structured trajectories through symbolic space.

2.1 Operationalization
Operationalization is the moment when symbolic rules become operations — explicit transformations that can be carried out step by step. Operationalization emerges when:

- symbolic rules are expressed as actionable steps
- steps can be applied to symbolic states
- each step produces a new symbolic configuration
- the sequence of steps defines a coherent transformation

Operationalization is not yet computation.
It is the activation of formal structure.

In ED terms: Operationalization is the conversion of symbolic rules into executable transformations of symbolic ED motifs.

This is the first time symbolic structure becomes procedurally active.

2.2 Repeatability
Operations become computational when they can be reliably repeated. Repeatability emerges when:

- operations produce consistent results
- consistency holds across contexts
- operations can be applied multiple times
- repeated application preserves structural coherence

Repeatability is the foundation of procedural stability.
It ensures that operations behave the same way each time they are executed.

In ED terms: Repeatability is the stabilization of operational transformations across repeated execution.

This is the first time symbolic operations become procedurally stable.

2.3 Determinacy
Determinacy emerges when operations produce predictable outcomes. Determinacy is not rigidity; it is the structural guarantee that:

- the same operation applied to the same state yields the same result
- operations follow well-defined trajectories
- procedural paths are constrained by the system's architecture
- outcomes can be anticipated before execution

Determinacy is what makes computation reliable.

In ED terms: Determinacy is the predictable propagation of symbolic ED motifs through operational transformations.

This is the first time symbolic processes become procedurally predictable.

2.4 Executability
Executability is the final step in the transition from formal systems to computation. Executability emerges when:

- operations can be carried out by agents or machines
- execution does not require interpretation at each step
- procedures can unfold autonomously once initiated
- symbolic transformations become performable

Executability is not mechanization.
It is the embodiment of procedure in an agent, system, or machine capable of carrying it out.

In ED terms: Executability is the capacity of operational ED motifs to be performed by an agent or machine.

This is the first time symbolic transformations become procedurally embodied.

Formal systems become computational processes when:
- rules become operations
- operations become repeatable
- repeatability becomes determinacy
- determinacy becomes executability

This is the architecture of computational emergence — the threshold where formal systems acquire procedural motion.

3. The Architecture of Computation

How operations become computational systems

Computation is not simply the execution of isolated operations. A single operation does not constitute a computational system any more than a single symbolic transformation constitutes a symbolic system. Computation emerges when operations become organized — when they are embedded within an architecture that provides structure, continuity, and control. A computational system is a procedural architecture: a system in which symbolic states, operations, control structures, and memory interact to produce systematic trajectories through symbolic space.

In the ED ontology, computation is the architectural continuation of formal systems. Formal systems provide symbolic structure; computation provides procedural structure. Computation arises when symbolic states can be transformed by operations, when operations can be sequenced by control structures, and when sequences can be stabilized by memory. These components together form the architecture of computational becoming.

A computational system requires four structural elements:
- states — symbolic configurations
- operations — transformations of states
- control — structures that guide operations
- memory — persistence across steps

These elements transform symbolic operations into computational processes.

3.1 State as Symbolic Configuration

Computation operates on states — symbolic configurations that encode the current condition of the system. A

state is not a static snapshot; it is a procedural position within a trajectory of transformations. States emerge when:

- symbolic structures are organized into coherent configurations
- configurations can be modified by operations
- each configuration determines which operations are applicable
- the system's future trajectory depends on its current state

States are the substrate on which computation acts.

In ED terms: A state is a symbolic ED configuration that determines the system's procedural possibilities.

This is the first time symbolic structures become procedurally situated.

3.2 Operations as State Transformations

Operations are the transformations that move the system from one state to another. Operations become computational when:

- they take symbolic states as input
- they produce new symbolic states as output
- they preserve the system's structural coherence
- they can be composed into larger procedural sequences

Operations are the building blocks of computation. They define the system's procedural dynamics.

In ED terms: An operation is a procedural transformation of symbolic ED motifs that produces a new state.

This is the first time symbolic transformations become procedurally generative.

3.3 Control as Procedural Structure

Control structures determine which operations occur when. Control is the architecture that guides the system's procedural unfolding. Control emerges when:

- operations are sequenced according to rules
- branching structures determine alternative paths
- loops allow repeated execution
- procedural flow becomes part of the system's architecture

Control is not external direction.
It is the internal organization of procedural possibility.

In ED terms: Control is the procedural architecture that governs the sequencing of operational ED motifs.

This is the first time symbolic processes acquire procedural direction.

3.4 Memory as Procedural Continuity

Memory provides continuity across computational steps. Memory emerges when:

- symbolic states can persist across operations
- intermediate results can be stored and retrieved
- procedural context can be maintained
- the system's history influences its future trajectory

Memory is not storage.
It is the persistence of procedural structure across time.

In ED terms: Memory is the persistence of symbolic ED motifs that stabilizes computational trajectories.

This is the first time symbolic processes become procedurally continuous.

Computation arises when:

- states provide symbolic configurations
- operations transform those configurations
- control structures guide the transformations
- memory stabilizes the process across steps

This is the architecture of computational systems — the procedural extension of symbolic structure.

4. Computation as Proceduralization

When computation becomes procedural architecture

Computation is not merely the execution of operations. A single operation, even if executable, does not constitute a computational system. Computation becomes meaningful when operations are organized into procedures — structured sequences that unfold across time, guided by control, stabilized by memory, and capable of producing coherent trajectories through symbolic space. Proceduralization is the moment when computation becomes architecture.

In the ED ontology, proceduralization is the continuation of operationalization. Operationalization turns rules into operations; proceduralization turns operations into procedures. A procedure is not just a list of steps. It is a structured pattern of execution — a reusable, generalizable, and systematically applicable motif that transforms symbolic states in predictable ways. When procedures stabilize, they become algorithmic.

Proceduralization is the threshold where computation becomes systematic, reusable, and architecturally generative.

4.1 When Operations Become Procedures

Operations become procedures when they are:

- sequenced into coherent structures
- organized by control flow
- applied to symbolic states in systematic ways
- capable of producing multi‑step transformations

A procedure is a trajectory through symbolic space — a structured unfolding of operations that transforms one state into another through a series of intermediate configurations.

Procedures emerge when:

- operations are chained
- chains become stable

- stability becomes reusable
- reuse becomes part of the system's architecture

In ED terms: A procedure is a structured sequence of operational ED motifs that produces a coherent transformation across states.

This is the first time symbolic operations become procedurally extended.

4.2 When Procedures Become Reusable
Reusability is the moment when procedures become general — when they can be applied across contexts, inputs, and symbolic configurations. Reusability emerges when:

- procedures operate on classes of states
- their structure is independent of specific content
- they can be invoked repeatedly
- they produce consistent results across applications

Reusability is not repetition.
It is generalization — the extraction of procedural invariants.

In ED terms: Reusability is the stabilization of procedural ED motifs into general patterns that apply across contexts.

This is the first time procedures become procedurally general.

4.3 When Reusability Becomes Algorithmic
Algorithms emerge when reusable procedures become generalized procedural structures — patterns that define how to solve classes of problems, not just individual instances. Algorithmic structure appears when:

- procedures are parameterized
- parameterization supports generality
- generality supports abstraction
- abstraction becomes part of the system's architecture

An algorithm is not a procedure.

It is a procedural schema — a general pattern for generating procedures.

In ED terms: An algorithm is a generalized procedural ED motif that defines a reusable pattern of transformation.

This is the first time computation becomes procedurally abstract.

4.4 When Algorithms Become Architecture
Algorithms become architectures when:

- multiple algorithms interact
- interactions form higher‑order procedural structures
- these structures support complex computational trajectories
- algorithmic patterns become part of the system's identity

Algorithmic architectures are not collections of algorithms.

They are systems of procedural possibility — architectures that define how procedures can be generated, combined, and transformed.

In ED terms: An algorithmic architecture is a system of interacting procedural ED motifs that governs computational becoming.

This is the first time computation becomes procedurally architectural.

Computation becomes proceduralization when:
- operations become procedures
- procedures become reusable
- reusability becomes algorithmic
- algorithms become architecture

This is the ED threshold where computation becomes systematic, general, and architecturally generative — the foundation of algorithmic reasoning.

5. The Emergence of Algorithmic Reasoning

How computation transforms inference

Computation does not merely execute procedures. Once procedures become reusable, general, and architecturally organized, they enable a new mode of reasoning: algorithmic reasoning. Algorithmic reasoning is the explicit, executable, and systematically generative manipulation of procedural structures. It is the moment where inference becomes procedural, where explanations become algorithmic, and where abstraction becomes the extraction of procedural invariants.

In the ED ontology, algorithmic reasoning is not a replacement for symbolic reasoning. It is its procedural extension. Symbolic reasoning operates on symbolic structures; algorithmic reasoning operates on procedural structures. This shift allows reasoning to become:
- executable — inference unfolds as a sequence of operations
- systematic — procedures follow stable control structures
- general — algorithms apply across classes of problems
- verifiable — results can be reproduced by executing the same procedure
- scalable — procedural motifs can be nested, composed, and reused

Algorithmic reasoning is the first ED regime where inference becomes procedurally explicit.

5.1 Executable Inference

Inference becomes executable when reasoning is expressed as a procedure — a structured sequence of operations that transforms one symbolic state into another. Executable inference emerges when:
- reasoning steps are operationalized
- operations are sequenced by control structures
- sequences produce predictable outcomes
- outcomes can be reproduced by re-execution

Executable inference is not simply “doing the steps.”
It is the proceduralization of reasoning.

In ED terms: Executable inference is the expression of reasoning as a sequence of operational ED motifs that can be performed by an agent or machine.

This is the first time inference becomes procedurally performable.

5.2 Algorithmic Explanation
Explanation becomes algorithmic when relationships between structures can be demonstrated through procedural derivations. Algorithmic explanation emerges when:

- procedures reveal how one structure transforms into another
- transformations can be executed step by step
- each step is justified by the system’s architecture
- explanations can be reproduced by re-execution

Algorithmic explanation is not narrative.
It is procedural demonstration — the unfolding of structure through execution.

In ED terms: Algorithmic explanation is the use of procedural ED motifs to demonstrate structural relationships through execution.

This is the first time explanations become procedurally demonstrable.

5.3 Algorithmic Abstraction
Abstraction becomes algorithmic when procedural patterns are extracted, generalized, and stabilized as algorithmic motifs. Algorithmic abstraction emerges when:

- procedures reveal recurring patterns
- patterns are parameterized
- parameterization supports generality
- generality becomes part of the system’s architecture

Algorithmic abstraction is not simplification.
It is the procedural elevation of patterns within computational systems.

It produces:

- algorithmic schemas
- procedural templates
- higher-order algorithms
- architectures of procedural possibility

In ED terms: Algorithmic abstraction is the extraction of generalized procedural ED motifs from patterns of execution.

This is the first time abstraction becomes procedurally generative.

Algorithmic reasoning emerges when:

- inference becomes executable
- explanation becomes algorithmic
- abstraction becomes procedural

Algorithmic reasoning is the ED regime where computation becomes an engine of procedural thought — architectures capable of supporting executable inference, procedural explanation, and algorithmic abstraction.

6. Computational Cognition

When computation becomes a cognitive regime

Computational systems extend cognition by providing procedural scaffolds that stabilize, amplify, and reorganize the dynamics of thought. When computational systems become part of a mind's ongoing cognitive ecology, computational cognition appears.

Computational cognition is the first ED regime where conceptual and symbolic becoming are shaped by procedural architectures.

6.1 Computational Scaffolding

Computational systems extend cognition by providing procedural scaffolds — external structures that support and amplify reasoning. Computational scaffolding emerges when:

- procedures stabilize complex symbolic or conceptual transformations
- algorithmic structures reduce cognitive load
- executable processes enable new forms of reasoning
- computational tools become part of the mind's problem-solving repertoire

Scaffolding is not outsourcing.
It is procedural extension — the integration of computational structures into cognitive dynamics.

In ED terms: Computational scaffolding is the use of procedural ED motifs to support and extend conceptual and symbolic cognition.

This is the first time cognition becomes procedurally extended.

6.2 Computational Reorganization

Computational systems do not merely support cognition; they reorganize it. Computational reorganization emerges when:

- algorithmic distinctions reshape conceptual categories
- procedural structures introduce new conceptual and symbolic possibilities
- algorithmic abstractions reorganize conceptual hierarchies
- computational architectures alter the trajectories of reasoning

Computational systems become architectural forces that reshape how minds conceptualize, infer, and explain. They introduce new forms of structure — loops, recursion, modularity, control flow — that reorganize conceptual and symbolic space itself.

In ED terms: Computational reorganization is the restructuring of conceptual and symbolic ED motifs through algorithmic architecture.

This is the first time conceptual landscapes are shaped by procedural structure.

6.3 Computational Identity

As computational scaffolding and computational reorganization stabilize, a mind's computational repertoire becomes part of its identity. Computational identity is not a set of skills or a programming profile. It is the procedural architecture that shapes how a mind reasons, abstracts, explains, and interacts with the world.

Computational identity emerges when:

- computational systems become habitual cognitive tools
- algorithmic distinctions shape interpretation
- procedural practices shape reasoning trajectories
- computational forms become part of a mind's self-organization

Computational identity is not a worldview.
It is the ED architecture of a mind's procedural becoming.

In ED terms:

> Computational identity is the stabilized configuration of procedural ED motifs that shapes a mind's reasoning and interpretation.

This is the first time the universe produces systems whose identity is procedurally structured.

Computational cognition emerges when:

- computational scaffolding extends conceptual and symbolic cognition
- computational reorganization reshapes conceptual and symbolic space
- computational identity stabilizes procedural practice

Computational cognition is the ED regime where computation becomes a cognitive architecture — where procedures, algorithms, control structures, and algorithmic abstractions form a coherent system that guides a mind's becoming.

7. The ED Architecture of Computation

Computation is not an add-on to symbolic systems. It is their procedural continuation. When symbolic systems become operational, when operations become procedures, when procedures become reusable, and when reusability becomes algorithmic, computation emerges as a new ED regime — one in which symbolic structure acquires motion, direction, and procedural generativity.

Computation arises through a sequence of structural transitions:

- symbolic rules become operations
- operations become procedures
- procedures become reusable
- reusability becomes algorithmic
- algorithms become architectures

This is the ED ladder from formalization to proceduralization.

In the ED ontology, computation is the first domain where:

- symbolic structures become operational
- operational structures become repeatable
- repeatable structures become deterministic
- deterministic structures become executable
- executable structures become procedural
- procedural structures become reusable
- reusable structures become algorithmic
- algorithmic structures become architectural
- architectural structures become cognitive

These transitions are not optional. They are the structural consequences of systems that:

- operationalize symbolic rules into executable steps
- stabilize operations into reusable procedures
- generalize procedures into algorithmic motifs
- organize algorithmic motifs into procedural architectures
- integrate computational structures into conceptual and symbolic cognition
- navigate procedural architectures through executable inference
- stabilize procedural practice into computational identity

Computation is the first ED regime where becoming is no longer merely conceptual, symbolic, or formal.
It is procedural.

In ED terms: Computation is the ED regime where symbolic structures become executable processes that support algorithmic reasoning.

This is the architectural meaning of computation.

Computation is the hinge between symbolic systems and algorithmic architectures. It is the domain where:

- symbolic form becomes operational
- operational structure becomes procedural
- procedural structure becomes algorithmic
- algorithmic structure becomes architectural
- architectural structure becomes cognitive

These capacities do not yet constitute recursion, self-modification, or autoprocedural systems.
But they form the organizational foundation from which all three will arise.

Paper 20 will develop the next threshold: how computation becomes recursive, how recursion becomes self-modifying, and how self-modifying architectures become the ED regime where becoming becomes autoprocedural.

8. Conclusion — Computation as ED's First Procedural Threshold

Computation marks the moment where the architecture of becoming becomes procedural. Symbolic systems gave the universe formal structures that could be manipulated explicitly. Computation gives it operational structures — systems capable of executing, repeating, and composing transformations in systematic ways. This is the decisive threshold where symbolic rules become operations, where operations become procedures, and where procedures become algorithmic architectures capable of supporting executable inference, procedural explanation, and algorithmic abstraction.

The computational arc has shown that computation arises when:

- symbolic rules become operational
- operational structures become repeatable
- repeatable structures become deterministic
- deterministic structures become executable
- executable structures become procedural
- procedural structures become reusable
- reusable structures become algorithmic
- algorithmic structures become architectural
- architectural structures become cognitive

These transitions are not optional. They are the structural consequences of systems that must not only formalize symbolic structure, but proceduralize it — systems whose coordination, problem-solving, and self-organization depend on the ability to stabilize, execute, and systematically transform procedural motifs.

Computation is the first ED regime where coherence is not merely conceptual, symbolic, or formal, but procedural — where the architecture of becoming is shaped by explicit operations, control structures, and algorithmic patterns.

In ED terms: Computation is the ED regime through which symbolic structures become executable processes that support algorithmic reasoning.

This is the architectural meaning of computation.

Computation is the hinge between symbolic systems and algorithmic architectures. It is the domain where:

- symbolic form becomes operational
- operational structure becomes procedural
- procedural structure becomes algorithmic
- algorithmic structure becomes architectural
- architectural structure becomes cognitive

These capacities do not yet constitute recursion, self-modification, or autoprocedural systems.
But they form the organizational foundation from which all three will arise.

Paper 20 will develop the next threshold: how computation becomes recursive, how recursion becomes self-modifying, and how self-modifying architectures become the ED regime where becoming becomes autoprocedural.