Hierarchical Structural Factorization of Computational Graphs

Abstract
We introduce a structural theory of hierarchical factorization for deterministic computational graphs. Factorization replaces interface-preserving subgraphs with block nodes while preserving external dependencies. We define factor blocks, admissible factorization steps, and iterated factorizations. The main result establishes the existence of computational towers—hierarchies of compatible graph factorizations representing the same computation.

Author Name Sergei Zubov
zubov369@gmail.com

1 Introduction

1.1 Computational graphs as representations of deterministic computation

Deterministic computations admit natural structural representations in the form of directed graphs in which vertices correspond to elementary computational operations and edges represent dependency relations between these operations. Such graph representations arise in a wide range of contexts in theoretical computer science, including arithmetic circuits, dataflow models, dependency graphs used in program analysis, and intermediate representations employed in compilers.

In these representations, the directed edges encode the partial order induced by data dependencies between operations. A computation is therefore naturally associated with a directed acyclic graph whose structure captures the relationships between individual computational steps independently of a particular evaluation strategy.

Graph-based descriptions of computation are typically introduced as auxiliary representations supporting algorithmic analysis or execution models. In contrast, the present work studies computational graphs as structural objects in their own right. The goal of the theory developed in this article is not to describe how computations are executed, but to investigate structural transformations of computational graphs that preserve their dependency structure.

This structural perspective allows the properties of computations to be studied independently of specific computational models or machine architectures.

1.2 Atomic representations established in the preceding articles

The present article continues the structural theory of deterministic computation developed in the preceding works of this series.

Articles 1–4 introduced the notion of atomic transition forms, providing a canonical structural representation of deterministic transitions. In that framework, deterministic computations are represented through atomic operations defined over a fixed signature, and the resulting atomic structures were studied using algebraic, metric, and categorical methods.

In particular, the previous articles established:

- a canonical atomic representation of deterministic transitions,
- algebraic structures associated with atomic forms,

- structural metrics defined on atomic transition forms,
- a categorical framework describing normalization of deterministic transitions.

These results provide a foundational description of deterministic computations at the atomic structural level.
The atomic transition forms introduced in those works naturally induce computational graphs in which vertices correspond to atomic operations and edges represent dependency relations between these operations. These graphs serve as the starting point for the hierarchical constructions developed in the present article.

1.3 Motivation for hierarchical representations

Computational graphs that arise in practice frequently exhibit internal structure. Groups of operations often form coherent subgraphs that interact with the remainder of the computation through well-defined interfaces. Such subgraphs may correspond to reusable computational components, modules, or repeated structural patterns.
Recognizing such substructures suggests the possibility of replacing entire subgraphs with abstract vertices representing the corresponding computational blocks. This transformation produces a new graph that represents the same computation while providing a coarser structural description.
Applying such transformations repeatedly yields a hierarchy of graph representations describing the same computation at different levels of abstraction.
The aim of this article is to provide a formal structural theory describing this process.

1.4 Structural scope of the theory

The theory developed in this work concerns the structural properties of computational graph factorizations. In particular, the article introduces and studies:

- factor blocks representing subgraphs together with their interfaces,
- admissible factorization steps preserving dependency relations,
- iterated factorizations of computational graphs,
- hierarchical representations of computations arising from such factorizations.

The theory deliberately does not address the following questions:

- how candidate subgraphs for factorization are discovered,
- how optimal factorizations should be chosen,
- algorithmic procedures for detecting repeated structures.

These questions belong to optimization, program analysis, or graph compression theories and lie outside the scope of the present work. Instead, this article assumes that factorization steps are given and investigates the structural consequences of applying them.

1.5 Relative atomicity

An important conceptual principle underlying the theory is the notion of relative atomicity.

In many representations of computation the term atomic operation refers to an elementary step of the computation. However, what is considered atomic depends on the level of abstraction chosen to represent the computation.

In the framework developed here, the atomic representation established in the preceding articles serves as the base structural level. The operations represented at this level are treated as indivisible elements of the graph.
The theory developed in this article does not attempt to determine whether these operations could be further decomposed. Instead, atomicity is understood relative to the chosen structural representation.
This principle allows the structural theory to remain independent of the methods used to construct the atomic representation.

1.6 From factorization steps to hierarchical structures

A single factorization step replaces a selected subgraph of a computational graph with an abstract vertex representing the entire block. In order for this transformation to preserve the structure of the computation, the external dependency relations between the block and the remainder of the graph must remain unchanged.
When this condition is satisfied, the resulting graph represents the same computation at a different level of structural granularity.
Because the result of a factorization step is itself a computational graph, the same operation may be applied repeatedly. This leads naturally to sequences of compatible factorizations

$$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots$$

in which each graph provides a structural representation of the same computation.
The hierarchical structures generated by such sequences form the central object of study in this article.

1.7 Computational towers

We call a hierarchy of compatible factorizations generated from a computational graph a computational tower.
Each level of the tower provides a structural representation of the same computation, differing only in the granularity used to describe its internal structure. Lower levels describe the computation in terms of individual operations, while higher levels represent larger structural units obtained through factorization.
An important consequence of this perspective is that deterministic computation is not represented by a single graph but rather by the entire hierarchy of compatible graph representations generated through factorization.
The computational tower therefore captures the structural space of representations associated with a given computation.

1.8 Main contributions of the paper

The principal contributions of this article are the following.

First, we introduce the notion of factor blocks, representing subgraphs together with their structural interfaces.

Second, we formalize admissible factorization steps that replace such blocks by abstract vertices while preserving dependency relations.

Third, we introduce the concept of relative atomic representation, clarifying the role of the atomic graph as the base level of structural analysis.

Fourth, we define iterated factorizations of computational graphs and study their structural properties.

Finally, these constructions lead to the notion of computational towers, which organize compatible factorizations of computational graphs into hierarchical structures.

1.9 Main theorem

The central result established in this article is the following.

Theorem (Existence of Computational Towers). Let G_0 be a computational graph representing a deterministic computation. If admissible factorizations can be applied to its subgraphs, then there exists a sequence of compatible factorizations

$$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots$$

forming a computational tower in which each graph G_k represents the same computation as G_0.

The proof proceeds by induction: the base step constructs the first factorization, and the inductive step shows that admissible factorizations preserve the structural properties required for subsequent factorizations.

1.10 Role of this article within the series

This article marks the transition from the atomic structural theory developed in Articles 1–4 to the hierarchical theory developed in the remainder of the series.

Articles 1–4 established the atomic representation of deterministic computations and investigated its algebraic, metric, and categorical properties.

The present article introduces hierarchical factorizations of computational graphs constructed over these atomic representations and establishes the existence of computational towers.

The subsequent articles develop further consequences of this framework:

- Article 6 shows that structural properties established for atomic representations remain invariant across all levels of a computational tower.
- Article 7 introduces transition structures defined over computational towers, showing that mechanisms commonly associated with control flow arise from activation relations defined over the tower.

Together, these results lead to a structural decomposition of deterministic computation into two fundamental components:

1. the computational structure, represented by the computational tower, and
2. the transition structure, defined over this hierarchy.

2 Computational Graphs

2.1 Basic definition

We begin by introducing the structural object that represents deterministic computation throughout this work.

Definition 2.1 (Computational graph). A computational graph is a directed graph $G = (V, E)$ where V is a finite set of vertices and $E \subseteq V \times V$ is a set of directed edges. Vertices represent elementary computational operations, while directed edges represent dependency relations between operations. An edge $(u, v) \in E$ indicates that the operation corresponding to vertex v depends on the result of the operation corresponding to vertex u.

2.2 Acyclicity

Deterministic computations impose a natural ordering of operations determined by dependency relations. For this reason we restrict attention to directed acyclic graphs.

Definition 2.2 (Acyclic computational graph). A computational graph $G = (V, E)$ is said to be acyclic if it contains no directed cycle. Equivalently, there does not exist a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1) \in E$.
In the remainder of the paper all computational graphs are assumed to be acyclic.

2.3 Dependency relation

The directed edges of the graph induce a dependency relation between vertices.

Definition 2.3 (Dependency relation). Let $G = (V, E)$ be a computational graph. The dependency relation $\prec$ on V is defined as the transitive closure of the edge relation. For vertices $u, v \in V$, we write $u \prec v$ if there exists a directed path from u to v. Thus $u \prec v$ means that the operation represented by v depends (possibly indirectly) on the result of u.

2.4 Dependency order

The dependency relation defines a partial order on the vertices of the graph.

Proposition 2.1. For every acyclic computational graph G, the dependency relation $\prec$ defines a partial order on the vertex set V.

Proof. We verify the three properties of a partial order.

(1) Irreflexivity. If $v \prec v$ held for some vertex v, there would exist a directed path from v to itself, implying a directed cycle. This contradicts acyclicity.

(2) Transitivity. If $u \prec v$ and $v \prec w$, then concatenating the corresponding paths yields a path from u to w. Hence $u \prec w$.
(3) Antisymmetry. If both $u \prec v$ and $v \prec u$ held, there would exist a directed cycle between u and v, contradicting acyclicity.
Therefore the dependency relation is a partial order. □

2.5 Topological order

Because computational graphs are acyclic, their vertices admit a topological ordering.

Definition 2.4 (Topological order). A topological order of a computational graph $G = (V, E)$ is a linear ordering of the vertices $v_1, v_2, ..., v_n$ such that for every edge $(u, v) \in E$, the vertex u appears before v in the ordering.

Lemma 2.1. Every acyclic computational graph admits a topological ordering.
Proof. This is a classical result for directed acyclic graphs. Since the graph contains no directed cycles, the vertices can be ordered so that all edges point from earlier to later vertices. □

2.6 Inputs and outputs

Vertices without incoming edges represent operations that depend only on external data.

Definition 2.5 (Input vertices). A vertex $v \in V$ is called an input vertex if it has no incoming edges: $(u, v) \notin E$ for all $u \in V$.
Vertices without outgoing edges represent final results of the computation.

Definition 2.6 (Output vertices). A vertex $v \in V$ is called an output vertex if it has no outgoing edges: $(v, u) \notin E$ for all $u \in V$.

2.7 Structural equivalence of computational graphs

Since this work studies structural properties of computational graphs, it is natural to consider when two graphs represent the same structural object.

Definition 2.7 (Structural equivalence). Two computational graphs $G = (V, E)$ and $G' = (V', E')$ are structurally equivalent if there exists a graph isomorphism $\phi: V \to V'$ such that $(u, v) \in E \Leftrightarrow (\phi(u), \phi(v)) \in E'$. Structural equivalence identifies graphs that differ only by renaming of vertices while preserving the dependency structure.

2.8 Role of computational graphs in the theory

Computational graphs serve as the fundamental objects on which the factorization theory developed in this paper operates. All subsequent constructions—including factor blocks, factorization steps, and computational towers—are defined as structural transformations of such graphs.

The properties established in this section, in particular acyclicity and the induced dependency order, play a central role in proving the correctness of factorization operations in subsequent sections.

3 Subgraphs and Interfaces

3.1 Subgraphs of computational graphs

Factorization operates on subgraphs of a computational graph. We therefore begin by formalizing the notion of subgraph used throughout this work.

Definition 3.1 (Subgraph). Let $G = (V, E)$ be a computational graph. A subgraph of G is a graph $H = (V_H, E_H)$ such that $V_H \subseteq V$ and $E_H = E \cap (V_H \times V_H)$. Thus a subgraph is determined by selecting a subset of vertices and including all edges of the original graph whose endpoints lie in this subset.

Subgraphs inherit the acyclicity property of the original computational graph.

Lemma 3.1. If G is an acyclic computational graph and $H \subseteq G$ is a subgraph, then H is also acyclic.

Proof. Suppose H contained a directed cycle. Since all edges of H belong to E, the same cycle would exist in G, contradicting the assumption that G is acyclic. □

3.2 Boundary vertices

A subgraph interacts with the remainder of the computational graph through vertices that connect internal and external edges.

Definition 3.2 (Boundary vertex). Let $H = (V_H, E_H)$ be a subgraph of $G = (V, E)$. A vertex $v \in V_H$ is called a boundary vertex if there exists an edge connecting v with a vertex outside the subgraph. More precisely, v is a boundary vertex if either $(u, v) \in E$ with $u \notin V_H$, or $(v, u) \in E$ with $u \notin V_H$.

Boundary vertices represent the points where the subgraph interacts with the rest of the computational graph.

3.3 Interface of a subgraph

The external interaction of a subgraph is captured by its interface.

Definition 3.3 (Interface). Let $H = (V_H, E_H)$ be a subgraph of $G = (V, E)$. The interface of H consists of two sets of edges:

1. incoming interface edges: $E_{in}(H) = \{ (u, v) \in E \mid u \notin V_H, v \in V_H \}$,
2. outgoing interface edges: $E_{out}(H) = \{ (v, u) \in E \mid v \in V_H, u \notin V_H \}$.

The interface therefore describes all dependency relations between the subgraph and the external graph.

3.4 External interaction through interfaces

The interface determines how information flows between the subgraph and the remainder of the computational graph.
Incoming interface edges represent dependencies from external computations into the subgraph.
Outgoing interface edges represent dependencies from the subgraph to external computations.
Thus the interface defines the external behavior of the subgraph within the larger computational structure.

3.5 Interface equivalence

Two subgraphs may differ internally while exhibiting the same external behavior with respect to the surrounding graph.

Definition 3.4 (Interface equivalence). Let H_1 and H_2 be subgraphs of a computational graph G. We say that H_1 and H_2 are interface equivalent if they have the same incoming interface edges and the same outgoing interface edges.
Interface-equivalent subgraphs interact with the surrounding graph in the same structural manner.

3.6 Interface as structural boundary

The concept of interface plays a central role in the factorization process introduced later in this article.
A factorization step replaces a subgraph by an abstract vertex representing the entire subgraph. In order for this replacement to preserve the structure of the computation, the new vertex must maintain the same incoming and outgoing dependencies as the original subgraph.
The interface therefore serves as the structural boundary that must remain invariant during factorization.

3.7 Interface determines external interaction

The following lemma formalizes the role of the interface in determining the interaction between a subgraph and the rest of the graph.

Lemma 3.2. Let H be a subgraph of a computational graph G. The interface of H uniquely determines all dependency relations between vertices of H and vertices of $G \setminus H$.

Proof. All edges connecting H to the external graph must either originate outside H and terminate in H, or originate in H and terminate outside H. By Definition 3.3 these edges are precisely the incoming and outgoing interface edges. Therefore the interface contains all information describing the interaction between the subgraph and the external graph. □

3.8 Role of interfaces in graph factorization

Interfaces provide the structural mechanism that makes factorization possible.
When a subgraph is replaced by a single abstract vertex, the interface specifies how this new vertex connects to the rest of the graph. By preserving the interface during the replacement process, the global dependency structure of the computational graph remains unchanged.
In the next section we introduce factor blocks, which are subgraphs equipped with their interfaces and which serve as the basic units of graph factorization.

4 Factor Blocks

4.1 Structural blocks

Factorization of a computational graph is performed by replacing certain subgraphs with abstract vertices. In order for such a replacement to preserve the structure of the computation, the subgraph must be considered together with the interface through which it interacts with the surrounding graph.
This motivates the notion of a factor block, which is a subgraph equipped with its interface.

Definition 4.1 (Factor block). Let $G = (V, E)$ be a computational graph. A factor block is a pair $B = (H, \partial H)$ where $H \subseteq G$ is a subgraph and ∂H is the interface of H as defined in Section 3. The interface ∂H consists of incoming interface edges $E_{in(H)}$ and outgoing interface edges $E_{out(H)}$. Thus a factor block represents a subgraph together with all of its structural connections to the rest of the computational graph.

4.2 Internal and external structure

The structure of a factor block consists of two distinct components.

Internal structure: the subgraph H itself, including its vertices and its internal dependency edges.

External structure: the interface ∂H, describing dependencies entering the block and dependencies leaving the block.
This separation between internal and external structure plays a central role in the factorization process.

4.3 Block abstraction

The purpose of introducing factor blocks is to allow them to be replaced by abstract vertices that preserve the external dependency relations of the original subgraph.

Definition 4.2 (Block abstraction). Let $B = (H, \partial H)$ be a factor block of a computational graph G. A block abstraction replaces the subgraph H by a single vertex v_B such that every incoming interface edge (u, w) with $w \in H$ is replaced by (u, v_B), and every outgoing interface edge (w, u) with $w \in H$ is replaced by (v_B, u). The resulting graph is denoted G/B.

4.4 Interface preservation

The key requirement for block abstraction is that the interface of the block must remain unchanged with respect to the external graph.

Lemma 4.1 (Interface preservation). Let B be a factor block of a computational graph G. Then the block abstraction G/B preserves all dependency relations between the block and the external graph.

Proof. Every edge connecting the subgraph H with the external graph belongs to the interface of H. During block abstraction, each incoming interface edge (u, w) with $w \in H$ is replaced by (u, v_B), and each outgoing interface edge (w, u) with $w \in H$ is replaced by (v_B, u). Thus the connectivity between the block and the rest of the graph is preserved exactly through the new vertex v_B. □

4.5 Preservation of dependency structure

Block abstraction must also preserve the global dependency order of the computational graph.

Lemma 4.2 (Dependency preservation). Let G be a computational graph and B a factor block. Then the block abstraction G/B preserves the dependency relations between vertices outside the block.

Proof. Consider two vertices u and v outside the block H. Any path between them that does not pass through H remains unchanged in G/B. If a path passes through vertices of H, then in the abstracted graph this path passes through the vertex v_B. Since the interface edges preserve the entry and exit points of the block, the existence of the dependency path is preserved. □

4.6 Preservation of acyclicity

Since computational graphs are acyclic, it is necessary to verify that block abstraction does not introduce cycles.

Lemma 4.3 (Acyclicity preservation). If G is an acyclic computational graph and B is a factor block, then the abstracted graph G/B is also acyclic.

Proof. Assume that a cycle appears in G/B. Such a cycle must either exist entirely outside the block, or pass through the abstract vertex v_B. The first case contradicts the acyclicity of the original graph. The second case would imply that there existed a corresponding cycle passing through the vertices of H in the original graph, which again contradicts acyclicity. Therefore G/B is acyclic. □

4.7 Structural equivalence of the computation

Block abstraction changes the internal representation of the graph but does not alter the computation represented by the graph.

Proposition 4.1 (Structural equivalence under abstraction). Let G be a computational graph and B a factor block. Then the graph G/B represents the same computational structure as G with respect to the external dependencies.

Proof. All dependency relations between vertices outside the block remain unchanged, and all interactions between the block and the external graph are preserved through the interface mapping to the abstract vertex v_B. Therefore the global dependency structure of the computation remains intact. □

4.8 Role of factor blocks in hierarchical factorization

Factor blocks form the basic units of hierarchical factorization.
A factorization step replaces a selected factor block with its abstract vertex, producing a new computational graph representing the same computation at a coarser structural level.

By repeatedly applying such abstractions, one obtains a hierarchy of graphs in which each level provides a more abstract structural description of the same underlying computation. The formal definition of such factorization steps is introduced in the next section.

5 Factorization Step

5.1 Motivation

The notion of a factor block introduced in the previous section allows subgraphs of a computational graph to be treated as structural units. The purpose of a factorization step is to replace such a block with an abstract vertex while preserving the structural relations between the block and the remainder of the graph.
A factorization step therefore produces a new computational graph that represents the same computation at a different structural granularity.
The operation must satisfy several structural requirements:

- preservation of interfaces,
- preservation of dependency relations,
- preservation of acyclicity.

These requirements ensure that the resulting graph remains a valid computational graph.

5.2 Definition of factorization step

We now formalize the transformation that replaces a factor block with its abstraction.

Definition 5.1 (Factorization step). Let $G_k = (V_k, E_k)$ be a computational graph and let $B = (H, \partial H)$ be a factor block of G_k, where H is a subgraph with interface ∂H. A factorization

step replaces the block H by a new vertex v_B, producing a graph G_{k+1}. The transformation is denoted $G_k \twoheadrightarrow G_{k+1}$.
The vertex set of the new graph is:

$$V_{\{k+1\}} = \left(V_{k \setminus V_H}\right) \cup \{v_B\}.$$

The edge set is obtained by:

1. removing all edges whose endpoints lie entirely in H,
2. replacing each incoming interface edge (u, w) with $w \in H$ by (u, v_B),
3. replacing each outgoing interface edge (w, u) with $w \in H$ by (v_B, u).

The resulting graph G_{k+1} is called the factorized graph.

5.3 Structural interpretation

A factorization step changes the representation of the computation but not its structural dependencies.
The subgraph H is replaced by an abstract vertex v_B that represents the entire block. All connections between the block and the surrounding graph are preserved through the interface mapping.
Thus the factorization step produces a new graph representing the same computational structure at a coarser level.

5.4 Admissible factorizations

Not every replacement of a subgraph yields a valid computational graph. The transformation must satisfy structural consistency conditions.

Definition 5.2 (Admissible factorization). A factorization step $G_k \twoheadrightarrow G_{k+1}$ is called admissible if the following conditions hold:

4. Interface preservation. The connections between the block and the external graph are preserved through the abstract vertex v_B.
5. Dependency preservation. Dependency relations between vertices outside the block remain unchanged.
6. Acyclicity. The resulting graph G_{k+1} remains acyclic.

Only admissible factorizations are considered in the theory.

5.5 Preservation of reachability relations

We now show that admissible factorization preserves the reachability relations induced by the dependency order.

Lemma 5.1 (Reachability preservation). Let $G_k \twoheadrightarrow G_{k+1}$ be an admissible factorization. Then for any vertices u, v outside the block H, the reachability relation between them is preserved.

Proof. Consider vertices u and v not belonging to H. If a path between u and v does not pass through H, the same path remains unchanged in G_{k+1}. If the path passes through vertices of H,

then in the factorized graph the corresponding path passes through the abstract vertex v_B. Because the interface edges preserve the entry and exit points of the block, the existence of the path is preserved. Therefore the reachability relation between vertices outside the block remains unchanged. □

5.6 Preservation of computational graph structure

We now verify that admissible factorizations produce valid computational graphs.

Lemma 5.2 (Graph preservation). Let $G_k \twoheadrightarrow G_{k+1}$ be an admissible factorization. If G_k is a computational graph, then G_{k+1} is also a computational graph.

Proof. A computational graph is defined as a directed acyclic graph with a finite vertex set. Directedness is preserved since the transformation replaces edges by edges. The vertex set remains finite since the block vertices are replaced by a single vertex. Acyclicity is preserved by the admissibility condition. Thus the resulting graph satisfies the definition of a computational graph. □

5.7 Structural equivalence under factorization

Although factorization changes the internal representation of the graph, the computation described by the graph remains structurally equivalent.

Proposition 5.1 (Structural equivalence). Let $G_k \twoheadrightarrow G_{k+1}$ be an admissible factorization. Then G_k and G_{k+1} represent the same computational structure with respect to external dependency relations.

Proof. All dependency relations between vertices outside the block are preserved by Lemma 5.1. The interaction between the block and the external graph is preserved through the interface mapping to the abstract vertex v_B. Therefore the global dependency structure of the computation remains unchanged. □

5.8 Compatibility of successive factorizations

Factorization steps may be applied repeatedly to the resulting graphs.

In order for this process to remain well defined, successive factorizations must be compatible with the structure produced by previous steps. In particular, blocks selected for factorization in later steps must be subgraphs of the current computational graph and must possess well-defined interfaces.

This leads naturally to sequences of compatible factorizations

$$G_0 \twoheadrightarrow G_1 \twoheadrightarrow G_2 \twoheadrightarrow \cdots$$

which form the basis for hierarchical representations of computational graphs.

5.9 Role of factorization steps in hierarchical representations

Factorization steps provide the fundamental mechanism for constructing hierarchical representations of computational graphs.

Each step replaces a structural component of the graph by an abstract vertex while preserving the global dependency structure of the computation. By repeatedly applying such steps, increasingly abstract representations of the same computation are obtained.

The next section introduces the concept of relative atomicity, which clarifies how atomic representations depend on the chosen level of factorization.

6 Relative Atomicity

6.1 Motivation

In many representations of computation the notion of an atomic operation plays a fundamental role. Computational processes are often described as sequences of elementary operations whose internal structure is not further analyzed within the chosen representation.

However, what is considered atomic depends on the level of abstraction at which the computation is described. A computation represented at a low level may consist of elementary arithmetic or logical operations, while a higher-level description may treat entire sequences of such operations as single conceptual units.

This observation indicates that atomicity is not an intrinsic property of a computation itself, but rather a property of the structural representation used to describe that computation.

Within the structural framework developed in this article, computational graphs arise from an initial representation of a deterministic computation. The vertices of this graph correspond to operations that are treated as indivisible elements of the representation.

The purpose of this section is to formalize this idea through the notion of relative atomicity, which clarifies the role of the base graph in the hierarchical factorization framework.

6.2 Relative notion of atomic representation

The theory developed in this article begins with a computational graph that represents a deterministic computation through a set of vertices corresponding to operations and edges representing dependency relations between them.

This graph is not assumed to provide an absolute decomposition of the computation into minimal operations. Instead, it provides a structural representation whose vertices are treated as indivisible for the purposes of the present theory.

Definition 6.1 (Relative atomic representation). Let $G_0 = (V_0, E_0)$ be a computational graph representing a deterministic computation. The graph G_0 is called a relative atomic representation if the operations represented by its vertices are treated as indivisible elements within the structural framework under consideration. In this sense, atomicity is defined relative to the chosen representation rather than as an absolute property of the computation.

6.3 Atomic level as the base of hierarchical factorization

Once a computational graph G_0 has been chosen as the atomic representation of a computation, it serves as the base level of the hierarchical constructions introduced in this article.
All higher structural representations of the computation are obtained by applying admissible factorization steps to this base graph. These transformations produce a sequence of computational graphs

$$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots$$

in which each graph represents the same computation at a different level of structural abstraction. The base graph G_0 therefore provides the foundation from which the entire hierarchical structure is generated.

6.4 Independence from further decomposition

The structural theory developed in this article does not attempt to determine whether the operations represented by vertices of the base graph G_0 admit further decomposition.
In particular, the theory does not attempt to determine:

- minimal atomic representations of computations,
- canonical decompositions of operations,
- optimal structural factorizations.

Such questions belong to separate areas of research, including program analysis, compiler optimization, and graph compression. These topics concern the discovery or optimization of structural representations rather than the structural consequences of a given representation.
In contrast, the present work assumes that the atomic representation is given and studies the hierarchical structures that arise from applying factorization operations to this representation.

6.5 Relative minimality within the factorization framework

Although the atomic representation G_0 is not assumed to be globally minimal with respect to possible decompositions of the computation, it possesses a form of minimality relative to the operations considered in this theory.
The transformations studied in this article replace subgraphs of a computational graph by abstract vertices representing factor blocks. These transformations therefore move from finer structural descriptions to coarser ones.
Within this framework, the base graph G_0 cannot be further reduced using the allowed transformations.

Proposition 6.1 (Relative minimality). Let G_0 be the base graph chosen as the atomic representation of a deterministic computation. Within the framework of the factorization operations defined in this article, G_0 is minimal with respect to admissible factorizations.

Proof. Admissible factorizations defined in Section 5 replace subgraphs of a computational graph by abstract vertices representing factor blocks. These operations reduce the number of

vertices in the graph by replacing groups of vertices with a single vertex. The theory does not include operations that refine vertices into smaller subgraphs or decompose vertices into internal structures. Therefore no transformation defined in the present framework can produce a graph with a finer structural representation than G_0. Consequently G_0 is minimal relative to the factorization operations considered in this theory. □

6.6 Canonical foundation of the computational tower

The atomic graph G_0 serves as the canonical foundation of the hierarchical structure introduced in this article.

All higher levels of the hierarchy arise through admissible factorizations applied to this base graph. The resulting sequence of graphs

$$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots$$

forms the computational tower introduced later in the article.

Importantly, while the base level provides the starting point of the hierarchy, the theory does not assign privileged structural status to any level above the atomic representation. Each level of the tower provides a valid structural description of the same computation.

6.7 Non-privileged status of higher structural levels

Although the atomic representation serves as the base of the hierarchy, the higher levels obtained through factorization do not possess a unique or canonical status.

Different sequences of admissible factorizations may produce different hierarchies of graph representations starting from the same atomic graph. Each such hierarchy corresponds to a different structural decomposition of the computation.

Consequently, the theory does not identify a unique "correct" hierarchical representation of a computation. Instead, it studies the structural properties that remain valid across all such representations.

This observation leads to the principle that no level of granularity above the atomic representation possesses privileged structural status.

6.8 Role of relative atomicity in the theory

The concept of relative atomicity plays a crucial role in the structural theory developed in this article.

By fixing an atomic representation as the base level of analysis, the theory can focus on the structural transformations induced by factorization without requiring a canonical notion of elementary operation.

This approach allows the hierarchical structures studied in the remainder of the article to be defined independently of the particular method used to construct the atomic representation.

In particular, the results established in subsequent sections apply to any computational graph chosen as the base representation.

6.9 Connection with the preceding articles

The atomic representations studied in Articles 1–4 provide concrete instances of the relative atomic graphs considered in the present framework.

In those works deterministic transitions were decomposed into atomic operations defined over a fixed signature. The resulting atomic transition forms naturally induce computational graphs whose vertices correspond to atomic operations and whose edges represent dependency relations between them.

These atomic graphs therefore serve as natural base representations G_0 for the hierarchical constructions developed in this article.

The present work extends the atomic structural theory by showing that such base graphs admit hierarchical representations obtained through admissible factorizations.

6.10 Transition to hierarchical structures

Having established the notion of relative atomicity and the role of the atomic graph as the base of structural analysis, we can now study the hierarchical transformations generated by repeated factorizations.

In particular, we will show that admissible factorizations can be applied successively to produce sequences of compatible graph representations describing the same computation at different structural levels.

These sequences form the basis for the hierarchical structures introduced in the following sections and ultimately lead to the notion of computational towers.

7 Iterated Factorization

7.1 Motivation

In the previous sections we introduced the notion of a factorization step, which replaces a factor block of a computational graph by an abstract vertex while preserving the external dependency structure of the graph.

A single factorization step produces a new computational graph representing the same computation at a coarser structural level. However, once such a transformation has been performed, the resulting graph may itself contain subgraphs that admit further factorization.

This observation suggests that factorization is not a single operation but rather a process that can be applied repeatedly. The purpose of this section is to formalize this process and to establish the structural consistency of successive factorization steps.

7.2 Successive factorizations

Let G_0 be the initial computational graph representing a deterministic computation. Suppose that a factor block B_0 of G_0 is selected and replaced by its abstraction, producing a new graph G_1. If

the graph G_1 contains a factor block B_1, the same procedure can be applied again, producing a graph G_2. Continuing in this manner yields a sequence

$$G_0 \twoheadrightarrow G_1 \twoheadrightarrow G_2 \twoheadrightarrow \cdots$$

where each graph is obtained from the preceding one by an admissible factorization step.

Definition 7.1 (Iterated factorization). Let G_0 be a computational graph. An iterated factorization is a sequence of graphs $G_0 \twoheadrightarrow G_1 \twoheadrightarrow G_2 \twoheadrightarrow \ldots$ such that for every k, the step $G_k \twoheadrightarrow G_{k+1}$ is an admissible factorization step. Each graph G_k represents the same computation at a different level of structural abstraction.

7.3 Compatibility of successive factorizations

For iterated factorization to be well defined, successive factorization steps must be compatible with the structure produced by earlier steps. In particular, each factorization step must operate on a subgraph of the current graph and must preserve the structural properties required of computational graphs.

Lemma 7.1 (Admissibility preservation). Let $G_k \twoheadrightarrow G_{k+1}$ be an admissible factorization step. If a factor block B is defined in G_{k+1}, then replacing B by its abstraction produces another admissible factorization $G_{\{k+1\}} \twoheadrightarrow G_{\{k+2\}}$.

Proof. By Lemma 5.2, the graph G_{k+1} produced by an admissible factorization is itself a valid computational graph. Therefore any factor block defined within G_{k+1} satisfies the conditions required for factorization. Applying the block abstraction to such a block preserves interface relations, dependency structure, and acyclicity. Hence the resulting transformation is also an admissible factorization. □

7.4 Consistency of interfaces across levels

Another requirement for iterated factorization is the consistency of interfaces across successive levels of the hierarchy.

Lemma 7.2 (Interface consistency). Let $G_k \twoheadrightarrow G_{k+1}$ be an admissible factorization step. Then the interface relations between the abstracted block and the remainder of the graph remain well defined for subsequent factorizations.

Proof. During a factorization step, the interface edges connecting the block to the external graph are mapped to edges incident to the abstract vertex v_B. Thus all dependency relations between the block and the rest of the graph are preserved. Since the resulting graph G_{k+1} remains a valid computational graph, any subgraph selected for further factorization possesses a well-defined interface within G_{k+1}. Therefore interface relations remain consistent across successive levels of factorization. □

7.5 Preservation of computational structure

Iterated factorizations preserve the computational structure represented by the graph.

Proposition 7.1 (Structural equivalence under iterated factorization). Let $G_0 \twoheadrightarrow G_1 \twoheadrightarrow \cdots \twoheadrightarrow G_k$ be a sequence of admissible factorizations. Then all graphs in this sequence represent the same computational structure with respect to external dependency relations.

Proof. By Proposition 5.1, a single admissible factorization preserves the dependency structure between vertices outside the factorized block. Applying this result inductively to each factorization step shows that the dependency relations preserved by each transformation remain unchanged throughout the sequence. Therefore all graphs in the sequence represent the same computational structure. □

7.6 Hierarchical structure induced by iterated factorization

Iterated factorization naturally induces a hierarchical organization of the computational graph. At each level of the sequence $G_0 \twoheadrightarrow G_1 \twoheadrightarrow G_2 \twoheadrightarrow \ldots$, groups of operations are progressively replaced by abstract vertices representing larger structural units. Thus the sequence provides representations of the same computation at increasing levels of abstraction.

7.7 Toward computational towers

The sequences of graphs generated by iterated factorization form the basis of the hierarchical structures studied in this article.

Such sequences capture the idea that a deterministic computation may be described at multiple levels of structural granularity, each obtained from the previous one by factorizing a subgraph. In the next section we formalize this idea by introducing the notion of a computational tower, which organizes these successive factorizations into a single hierarchical object.

8 Computational Towers

8.1 Motivation

The previous sections introduced the concepts of factor blocks, admissible factorization steps, and iterated factorizations of computational graphs. These constructions show that a computational graph representing a deterministic computation may undergo structural transformations that replace selected subgraphs with abstract vertices representing entire computational blocks.

Each admissible factorization produces a new graph that preserves the dependency relations of the original computation while providing a coarser structural description of its internal organization.

Because the result of an admissible factorization is itself a valid computational graph, the same transformation may be applied repeatedly. This observation naturally leads to sequences of compatible graph representations obtained through successive factorization steps.

Such sequences reveal that deterministic computations may admit multiple structural representations that differ only in the granularity used to describe their internal structure.
The purpose of this section is to formalize this hierarchical structure.

8.2 Hierarchical representations of computation

Let G_0 be a computational graph representing a deterministic computation at the atomic level introduced in Section 6.
Suppose that admissible factorizations are applied successively to this graph. Each factorization step replaces a selected factor block by an abstract vertex representing the corresponding computational subgraph.

This process generates a sequence of graphs

$$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots$$

where each graph G_k represents the same computation as G_0. The graphs in this sequence differ only in the level of structural abstraction used to represent the computation. In particular:

- the graph G_0 describes the computation at the atomic level,
- the graph G_1 describes the computation after the abstraction of selected atomic subgraphs,
- the graph G_2 describes the computation after further structural abstractions,
- higher graphs represent increasingly coarse structural descriptions.

Thus the same deterministic computation may be associated with multiple compatible structural representations.

8.3 Definition of computational tower

The hierarchical organization described above is captured by the notion of a computational tower.

Definition 8.1 (Computational tower). Let G_0 be a computational graph representing a deterministic computation. A computational tower generated from G_0 is a sequence of graphs

$$\mathcal{T}(G^0) = \left(G^0 \Rightarrow G^1 \Rightarrow G^2 \Rightarrow \ \ldots\right)$$

such that for every k, the transformation $G_k \Rightarrow G_{k+1}$ is an admissible factorization step. Each graph G_k is called a level of the tower.

8.4 Structural interpretation

A computational tower organizes multiple structural representations of the same deterministic computation.
Each level of the tower is obtained by replacing selected subgraphs of the previous graph by abstract vertices representing factor blocks. These transformations preserve the dependency relations that define the computational structure.
Consequently, all graphs appearing in the tower describe the same computation even though their internal structure may differ significantly.

Lower levels of the tower provide detailed structural descriptions of the computation, while higher levels represent increasingly abstract structural views.
The computational tower therefore captures the hierarchical organization of computation through successive structural abstractions.

8.5 Representation hierarchy

The levels of a computational tower form a hierarchy of graph representations $G_0, G_1, G_2, \ldots$ that describe the same computation at different levels of abstraction. These levels may be interpreted as follows:

- Level G_0 — atomic representation of the computation consisting of individual operations.
- Level G_1 — representation obtained by abstracting selected atomic subgraphs into structural blocks.
- Level G_2 — representation obtained by abstracting larger structural components.
- Higher levels — progressively coarser representations in which increasingly large portions of the computation are represented as abstract blocks.

Each level therefore corresponds to a particular granularity of structural representation.

8.6 Compatibility of tower levels

For a sequence of factorizations to form a valid computational tower, the resulting graph representations must remain structurally compatible. In particular, admissible factorizations preserve the dependency relations between vertices outside the factorized blocks. This ensures that the global dependency structure of the computation remains unchanged across all levels of the tower.

Lemma 8.1 (Level compatibility). Let $G_k \Rightarrow G_{k+1}$ be an admissible factorization step. Then the dependency relations between vertices not belonging to the factorized block remain identical in both graphs.

Proof. By Lemma 5.1, admissible factorizations preserve reachability relations between vertices outside the factorized block. Since dependency relations in computational graphs are defined through reachability in the directed graph, these relations remain unchanged. Therefore the external dependency structure of the graph remains identical in both representations. □

8.7 Structural equivalence across tower levels

Although the graphs appearing in a computational tower differ in their internal structure, they represent the same computation. This property follows from the preservation of dependency relations under admissible factorizations.

Proposition 8.1 (Structural equivalence across tower levels). Let $\mathcal{T}(G_0) = (G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_k)$ be a computational tower. Then all graphs G_i in the tower represent the same computational structure with respect to dependency relations.

Proof. By Proposition 7.1, admissible factorizations preserve the dependency relations between vertices outside factorized blocks. Applying this result inductively to the sequence of factorizations shows that the dependency structure remains invariant across all levels of the tower. □

8.8 Hierarchical nature of computation

The concept of a computational tower reveals an important structural property of deterministic computation.

A deterministic computation is not associated with a single structural representation but rather with a family of compatible graph representations organized through hierarchical factorizations.

Each representation corresponds to a different structural perspective on the same computation. At lower levels, the computation is described through individual operations and their dependencies. At higher levels, groups of operations are represented as structural blocks that encapsulate larger computational units.

The computational tower therefore expresses the hierarchical organization of computation in a purely structural manner.

8.9 Computation as a tower of representations

An important consequence of the theory developed in this article is that deterministic computation is not represented by a single graph but by the entire computational tower generated from its atomic representation.

Each level of the tower provides a valid structural description of the computation, differing only in the granularity of the representation.

Thus a deterministic computation corresponds not to a single graph but to the hierarchical structure $\mathcal{T}(G_0)$ generated through admissible factorizations.

This perspective plays a central role in the structural theory of deterministic computation developed in this series.

8.10 Role of computational towers in the theory

Computational towers constitute the central structural objects introduced in this article.

They capture the hierarchical organization of deterministic computations and provide the structural space in which different levels of abstraction coexist.

The remaining articles of this series study additional structures defined over computational towers. In particular:

- Article 6 establishes that algebraic, metric, and categorical properties defined for atomic graphs remain invariant across all levels of the tower.
- Article 7 introduces transition structures defined over computational towers and shows that mechanisms commonly associated with control flow arise from activation relations defined over subgraphs of the tower.

Together, these developments lead to a structural decomposition of deterministic computation into two fundamental components:

1. the computational structure, represented by the computational tower, and
2. the transition structure, defined over this hierarchy.

9 Existence of Computational Towers

9.1 Motivation

In the previous sections we introduced the structural components required to describe hierarchical factorizations of computational graphs. In particular:

- Section 4 introduced factor blocks, representing subgraphs together with their interfaces.
- Section 5 defined admissible factorization steps, which replace factor blocks by abstract vertices while preserving dependency relations.
- Section 7 formalized iterated factorizations as sequences of compatible factorization operations.
- Section 8 introduced the notion of a computational tower, organizing these successive factorizations into a hierarchical structure.

These constructions suggest that hierarchical representations of deterministic computations arise naturally through the repeated application of admissible factorizations.

The remaining question is whether such hierarchical structures can always be constructed whenever admissible factorizations are available.

This section establishes the central result of the present article: admissible factorizations generate computational towers.

9.2 Statement of the main theorem

We now state the principal result of this article.

Theorem 9.1 (Existence of Computational Towers). Let G_0 be a finite computational graph representing a deterministic computation. Suppose that admissible factorizations can be applied to subgraphs of G_0. Then there exists a sequence of computational graphs

$$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots$$

such that:

1. each transformation $G_k \Rightarrow G_{\{k+1\}}$ is an admissible factorization step,
2. each graph G_k represents the same computational structure as G_0, and
3. the sequence forms a computational tower $\mathcal{T}(G^0) = \left(G^0 \Rightarrow G^1 \Rightarrow G^2 \Rightarrow \ldots\right)$ generated by admissible factorizations of G_0.

9.3 Construction of the tower

The computational tower is constructed through successive applications of admissible factorization steps.

Let G_0 be the initial computational graph representing a deterministic computation. If a factor block B_0 exists in G_0, we apply an admissible factorization step $G_0 \Rightarrow G_1$. The resulting graph G_1

preserves the dependency structure of G_0 while replacing the subgraph corresponding to B_0 with an abstract vertex.
If the graph G_1 contains another factor block B_1, the same transformation may be applied again: $G_1 \Rightarrow G_2$. Repeating this process produces a sequence of graphs

$$G^0 \Rightarrow G^1 \Rightarrow G^2 \Rightarrow \ldots$$

in which each graph is obtained from the previous one through an admissible factorization step. Each graph in the sequence represents the same deterministic computation at a different level of structural abstraction.

9.4 Proof of the theorem

Proof. We construct the computational tower inductively.

Base step. Let G_0 be the initial computational graph. If G_0 contains a factor block B_0, then by Definition 5.2 an admissible factorization step may be applied, producing the graph $G_0 \Rightarrow G_1$. By Lemma 5.2, admissible factorizations preserve the property of being a computational graph. Therefore G_1 is again a valid computational graph.

Inductive step. Assume that a sequence of admissible factorizations $G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_k$ has been constructed. If the graph G_k contains a factor block B_k, then the factorization operation may be applied again, producing $G_k \Rightarrow G_{\{k+1\}}$. By Lemma 7.1, admissibility of factorization steps is preserved under successive applications. In addition, by Lemma 5.2 the resulting graph $G_{\{k+1\}}$ remains a valid computational graph. Thus the sequence of admissible factorizations may be extended.

Preservation of computational structure. By Proposition 7.1, admissible factorizations preserve the dependency relations that define the computational structure of the graph. Therefore every graph G_k obtained through this construction represents the same deterministic computation as the original graph G_0.

Conclusion. The sequence $G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \ldots$ satisfies the definition of a computational tower given in Section 8. Therefore computational towers exist whenever admissible factorizations can be applied to the computational graph. □

9.5 Interpretation of the result

The theorem shows that hierarchical representations of deterministic computations arise naturally from admissible factorizations of computational graphs.
Starting from an atomic representation of a computation, successive abstractions of structural blocks generate a hierarchy of compatible graph representations describing the same computation at different levels of structural granularity.

This hierarchy is captured by the computational tower $\mathcal{T}(G_0)$. Thus deterministic computation is not represented by a single graph but by the entire tower of structural representations generated from the atomic graph.

9.6 Structural consequences

The existence of computational towers has important implications for the structural analysis of deterministic computation.

First, it shows that structural representations of computations naturally form hierarchical systems of compatible graphs.

Second, it establishes that different levels of structural abstraction correspond to different levels of graph factorization rather than to different computations.

Finally, it provides a structural framework in which different representations of the same computation may be studied simultaneously.

These properties allow the structural aspects of computation to be analyzed independently of particular computational models.

9.7 Role of the theorem in the series

The existence theorem established in this article provides the structural foundation for the subsequent developments of the theory. In particular:

- Article 6 shows that structural properties established for atomic graphs remain invariant across all levels of a computational tower.
- Article 7 introduces transition structures defined over computational towers and demonstrates that mechanisms associated with control flow arise from activation relations between subgraphs of the tower.

Together with the results established in the present article, these developments lead to a structural decomposition of deterministic computation into two complementary components:

4. the computational structure, represented by the computational tower, and
5. the transition structure, defined over the tower and governing the activation of its subgraphs.

10 Conclusion

10.1 Summary of the structural framework

This article introduced a structural theory of hierarchical factorizations for computational graphs representing deterministic computations.

Starting from the atomic representations established in the preceding articles of this series, we investigated how computational graphs may admit coarser structural descriptions obtained through successive abstractions of subgraphs.

The central idea of the theory is that computational graphs often contain substructures that may be treated as coherent structural units. By identifying such subgraphs together with their interfaces, it becomes possible to replace them with abstract vertices representing entire computational blocks.

These transformations preserve the dependency relations that define the computational structure of the graph. As a consequence, the resulting graphs represent the same deterministic computation while providing progressively coarser structural descriptions.
The theory developed in this work formalizes these transformations and studies the hierarchical structures generated by their repeated application.

10.2 Main constructions introduced in this article

The structural framework developed in this article is based on several key constructions.

First, we introduced the notion of factor blocks, which represent subgraphs of a computational graph together with the interfaces through which they interact with the surrounding graph.
Second, we defined admissible factorization steps, which replace such blocks by abstract vertices while preserving the external dependency relations of the computation.
Third, we introduced the concept of relative atomic representation, clarifying that the notion of atomic operation depends on the structural representation chosen as the base level of analysis.
Fourth, we formalized iterated factorizations, showing that admissible factorizations may be applied repeatedly while preserving the structural consistency of computational graphs.
Finally, these constructions led to the notion of computational towers, which organize compatible graph factorizations into hierarchical representations of deterministic computations.

10.3 Existence of hierarchical representations

The principal result of the article establishes the existence of computational towers generated by admissible factorizations.
This result shows that deterministic computations naturally admit hierarchical structural representations obtained through successive abstractions of subgraphs.
Starting from an atomic representation of a computation, admissible factorizations generate a sequence of computational graphs

$$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots$$

in which each graph represents the same computation at a different level of structural abstraction.
The resulting computational tower captures the hierarchical organization of the computation and provides a unified structural framework for analyzing different levels of representation.

10.4 Structural interpretation of deterministic computation

An important consequence of the theory developed in this article is that deterministic computation is not associated with a single structural representation.
Instead, a computation corresponds to an entire hierarchy of compatible graph representations generated through admissible factorizations of its atomic graph.
Each level of the hierarchy describes the same computation using a different structural granularity.

Thus deterministic computation is represented not by a single graph but by the entire computational tower generated from its atomic representation.
This perspective provides a structural view of computation that is independent of specific computational models or execution mechanisms.

10.5 Role of atomic representations

The atomic graphs introduced in Articles 1–4 serve as the canonical base level of the hierarchical structures studied in the present work.
In this framework, atomic graphs provide the starting point for hierarchical factorizations and therefore form the foundation of the computational tower.

It is important to emphasize that atomicity is defined relative to the chosen structural representation. The theory developed in this article does not attempt to determine whether the atomic operations themselves admit further decomposition.
Instead, the atomic representation is treated as the base level from which higher structural abstractions are constructed.

10.6 Separation of structural and transition aspects of computation

The hierarchical framework developed in this article separates two fundamental aspects of deterministic computation.
The computational structure describes the organization of operations and their dependency relations. In the present work this structure is captured by the computational tower generated through hierarchical factorizations.
The transition structure, in contrast, describes how different parts of the computational structure become active during the execution of the computation.
By distinguishing these two aspects, the theory separates the structural organization of a computation from the mechanisms that govern its execution.
This separation provides a conceptual foundation for the structural analysis of deterministic computations.

10.7 Foundations for the remainder of the series

The computational towers introduced in this article serve as the central structural objects for the subsequent developments of the theory.

Article 6 studies the preservation of structural properties across tower levels. In particular, it shows that algebraic, metric, and categorical properties established for atomic representations remain invariant under admissible factorizations.

Article 7 introduces transition structures defined over computational towers and demonstrates that mechanisms commonly associated with control flow arise naturally from activation relations between subgraphs of the tower.

Together with the results established in the present article, these developments lead to a structural decomposition of deterministic computation into two complementary components:

1. the computational structure, represented by the computational tower, and
2. the transition structure, defined over this hierarchical structure.

10.8 Final remarks

The theory developed in this article establishes that hierarchical structural representations of deterministic computations arise naturally through admissible factorizations of computational graphs.

These representations are organized through computational towers that capture the hierarchical organization of computations while preserving their dependency structure.
By separating the structural organization of computation from the mechanisms governing the activation of its components, the framework introduced in this work provides a foundation for the structural study of deterministic computation independent of particular computational models.
The results presented here therefore form a central component of the broader structural theory of deterministic computation developed in this series.

11 Relation to Existing Structural Representations

11.1 Structural representations of computation

Graph-based representations of computation have been studied extensively in a variety of contexts. Examples include arithmetic circuits, dataflow graphs, dependency graphs used in program analysis, and intermediate representations employed in compilers.
In all these settings computations are represented as directed graphs in which vertices correspond to operations and edges represent dependency relations between operations.
The present work operates within this general framework but focuses specifically on the structural properties of such graphs and on transformations that preserve their dependency structure. In particular, the theory developed in this article studies hierarchical factorizations of computational graphs rather than the operational semantics of the computations they represent.

11.2 Relation to circuit hierarchies

Hierarchical representations are widely used in circuit design, where complex circuits are often described through multiple levels of abstraction. In such representations, groups of components are encapsulated into modules that may themselves be treated as components at higher levels of the design hierarchy.
The factorization process introduced in this article resembles this hierarchical organization. However, the goal of the present work is not to model specific design methodologies but to establish a general structural framework in which such hierarchical representations arise naturally through admissible factorizations of computational graphs.

11.3 Relation to program modularization

Similar hierarchical structures appear in programming languages through mechanisms such as functions, procedures, and modules. These constructs group sequences of operations into reusable units that can be invoked within larger programs.

While such mechanisms are often motivated by software engineering considerations, the present work studies the underlying structural phenomenon from a graph-theoretic perspective. In particular, factor blocks represent subgraphs that may be abstracted into single vertices while preserving their interfaces with the surrounding graph.

Thus the theory provides a structural view of hierarchical program organization independent of specific programming constructs.

11.4 Relation to graph compression and DAG reduction

In the study of directed acyclic graphs, various techniques have been developed for reducing or compressing graph representations by identifying repeated structures or common subgraphs. Although these techniques also operate on subgraphs of directed acyclic graphs, their primary purpose is typically algorithmic efficiency or storage optimization.

The theory presented in this article does not address such algorithmic problems. Instead, it studies the structural consequences of replacing subgraphs by abstract vertices under conditions that preserve the dependency structure of the graph. In this sense, the factorization process considered here is purely structural rather than algorithmic.

11.5 Distinction from optimization theories

An important aspect of the present theory is that it does not attempt to determine optimal factorizations of computational graphs. In particular, the theory does not address:

- methods for identifying candidate blocks,
- criteria for optimal hierarchical decompositions,
- algorithms for discovering repeated structures.

Such problems belong to optimization or program analysis theories. The focus of this work is instead on the structural properties of computational graphs once a factorization step has been specified.

11.6 Structural nature of the results

The results established in this article concern the structural properties of computational graphs and their hierarchical factorizations.

In particular, the theory shows that admissible factorizations preserve the dependency structure of the computation and can be applied repeatedly to generate hierarchical graph representations organized in the form of computational towers.

These results provide a structural foundation that is independent of particular computational models or implementation mechanisms.

11.7 Role within the series

The present article forms the first part of the second stage of the theory developed in this series.

Articles 1–4 established the atomic structural representation of deterministic computations and studied the algebraic, metric, and categorical properties of these atomic forms.
The present article introduces hierarchical factorizations of computational graphs built over these atomic representations and establishes the existence of computational towers.
Article 6 shows that structural properties established for atomic graphs remain invariant across all levels of the computational tower. Article 7 introduces transition structures defined over computational towers, providing a structural account of activation and control within deterministic computations.
Together, these results lead to a structural decomposition of deterministic computation into computational and transition structures.

11.8 Limitations of the present work

The present article focuses exclusively on deterministic computations represented by directed acyclic graphs. The theory does not address:

- nondeterministic computations,
- cyclic computational structures,
- dynamic modification of graph structure during execution.

Extensions of the framework to such settings remain subjects for future investigation.

12 Scope and Limitations of the Theory

12.1 Structural scope of the results

The results established in this article concern the structural properties of computational graphs and their hierarchical factorizations.
In particular, the theory studies transformations that replace subgraphs by abstract vertices while preserving the dependency relations that define the computational structure. The resulting hierarchy of graph representations is captured by the notion of a computational tower.
The theory therefore addresses the structural organization of deterministic computations rather than their operational semantics.

12.2 Independence from computational models

The framework developed in this article does not depend on any particular computational model. Computational graphs may arise from many different sources, including:

- arithmetic circuits,
- dataflow systems,
- intermediate program representations,
- dependency graphs in compiler analysis.

The factorization theory applies to these graphs independently of the computational model from which they originate. In particular, the theory does not assume any specific instruction set, machine architecture, or evaluation strategy.

12.3 Independence from execution semantics

The theory does not describe how computations are executed. In particular, the results established here do not depend on:

- scheduling policies,
- evaluation order,
- control flow mechanisms.

Instead, the theory focuses exclusively on the structural relations between operations encoded in the computational graph.

Execution-related concepts are introduced only in later articles of this series, where transition structures defined over computational towers are studied.

12.4 No assumptions about optimal factorizations

An important aspect of the present framework is that it does not attempt to determine optimal factorizations of computational graphs. In particular, the theory does not address:

- algorithms for identifying candidate factor blocks,
- criteria for selecting optimal hierarchical decompositions,
- methods for minimizing graph size or depth.

Instead, the theory assumes that factorization steps are given and investigates the structural consequences of such steps.

Questions of optimal factorization belong to separate theories concerned with graph optimization or program analysis.

12.5 Absence of refinement operations

The transformations studied in this article move from finer structural representations to coarser ones. Factorization replaces a subgraph by an abstract vertex representing the entire block. The theory does not introduce operations that refine vertices into smaller subgraphs.

As a consequence, the atomic graph chosen as the base level of the hierarchy is treated as the minimal representation within the scope of the theory.

12.6 Deterministic and acyclic computations

The present work is restricted to deterministic computations represented by directed acyclic graphs. This restriction ensures that dependency relations induce a partial order on the operations of the computation.

The theory does not address computational graphs containing cycles, which may arise in representations of iterative or feedback-driven computations. Extensions of the framework to cyclic structures would require additional concepts beyond the scope of the present work.

12.7 Relation to future extensions

The structural framework established in this article serves as a foundation for further developments.

Article 6 studies structural invariance across tower levels, showing that algebraic, metric, and categorical properties established for atomic graphs remain valid under admissible factorizations.

Article 7 introduces transition structures defined over computational towers and provides a structural account of activation relations between subgraphs.
These developments extend the structural theory of deterministic computation beyond the factorization framework introduced here.

12.8 Summary

The theory developed in this article provides a structural framework for hierarchical representations of deterministic computations based on admissible factorizations of computational graphs. Within this framework:

- atomic graphs serve as the base level of representation,
- factorization steps generate successive structural abstractions,
- computational towers organize these abstractions into hierarchical structures.

The scope of the theory is intentionally limited to structural properties of computational graphs. Questions concerning optimization, execution semantics, or model-specific behavior lie outside the scope of the present work.

13 Final Remarks

13.1 Structural perspective on deterministic computation

The theory developed in this article provides a structural perspective on deterministic computation based on hierarchical factorizations of computational graphs.
Rather than associating a computation with a single graph representation, the results of this work show that a deterministic computation naturally admits a family of compatible structural descriptions organized as a computational tower.
Each level of the tower corresponds to a particular granularity of representation obtained through admissible factorizations of the underlying graph. Thus the computational tower expresses the hierarchical organization of computation in a purely structural manner.

13.2 Separation of structural levels

An important feature of the framework introduced in this article is the separation between the structural representation of computation and the mechanisms governing its execution.
The computational tower captures the structural organization of the computation independently of any particular model of evaluation. In particular, the tower does not describe the order in which operations are executed or the control mechanisms that determine which parts of the computation are active at a given moment.
Instead, it provides the structural space within which such mechanisms may operate.

13.3 From structural hierarchy to transition structure

Once the hierarchical organization of computation has been established, it becomes possible to study additional structures defined over this hierarchy.

In particular, the activation of subgraphs within the computational tower induces relations that describe how different parts of the computation become active during execution. These relations give rise to transition structures defined over the tower.
The study of such transition structures forms the subject of the subsequent articles of this series.

13.4 Position of the results within the series

The present article marks the transition from the atomic structural theory developed in Articles 1–4 to the hierarchical theory of computation developed in the remainder of the series.
Articles 1–4 established the atomic representation of deterministic computations and investigated the algebraic, metric, and categorical properties of these atomic forms.
The present work extends this framework by introducing hierarchical factorizations of computational graphs built over these atomic representations.
Together, these results provide the structural foundation for the hierarchical theory of deterministic computation developed in Articles 6 and 7.

13.5 Outlook

The computational tower introduced in this article serves as the central structural object for the remainder of the theory.
Article 6 investigates the preservation of structural properties across tower levels, establishing that the algebraic, metric, and categorical structures developed for atomic representations remain invariant under admissible factorizations.
Article 7 introduces transition structures defined over computational towers and shows that mechanisms commonly associated with control flow arise naturally from activation relations over the tower.
Together with the results of the present article, these developments lead to a structural decomposition of deterministic computation into two fundamental components:

- the computational structure represented by the tower, and
- the transition structure defined over this hierarchy.

13.6 Concluding statement

The theory developed in this article establishes that hierarchical structural representations of deterministic computation arise naturally through admissible factorizations of computational graphs.
These representations are organized through computational towers that capture the hierarchical organization of the computation while preserving its dependency structure.
This framework provides a structural foundation for the further study of deterministic computation independent of specific computational models.

14. References

[1] Zubov, V.
Atomic Normalization of SLP-Definable Deterministic Transitions.

Preprint, 2026.
[2] Zubov, V.
Algebraic Structure of Atomic Forms over a Fixed Signature.
Preprint, 2026.
[3] Zubov, V.
Compositional Structural Metrics on Atomic Transition Forms.
Preprint, 2026.
[4] Zubov, V.
Categorical Structure of Atomic Normalization for SLP-Definable Deterministic Transitions.
Preprint, 2026.
[5] Aho, A. V., Hopcroft, J. E., Ullman, J. D.
Data Structures and Algorithms.
Addison-Wesley, 1983.
[6] Kahn, G.
The semantics of a simple language for parallel programming.
Proceedings of IFIP Congress 74, 1974, pp. 471–475.
[7] Valiant, L. G.
Universality considerations in VLSI circuits.
IEEE Transactions on Computers, 30(2), 1981, pp. 135–140.
[8] Ullman, J. D.
Principles of Database and Knowledge-Base Systems.
Computer Science Press, 1988.
[9] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
MIT Press, 2009.