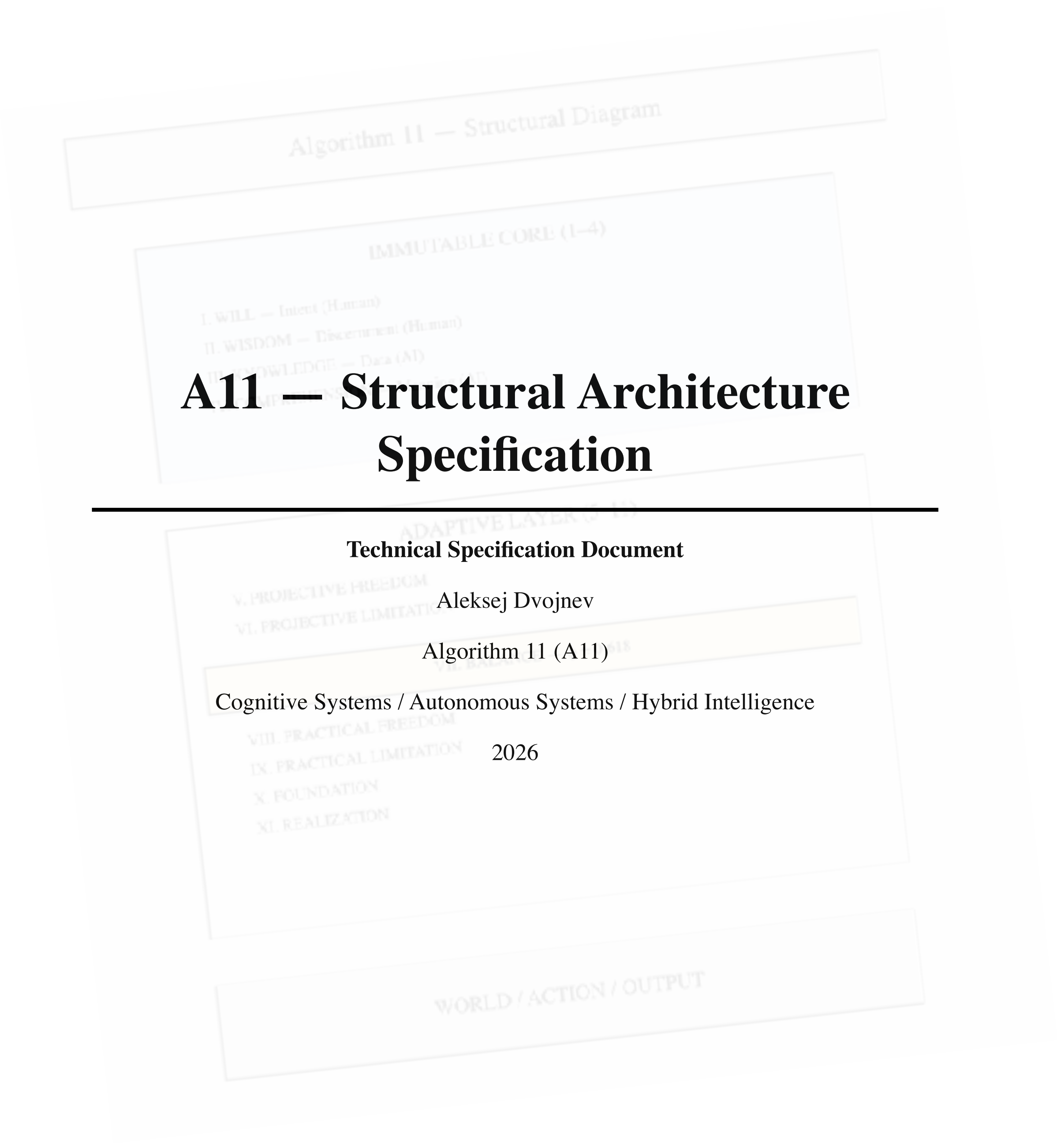

A11 — Structural Architecture Specification

Technical Specification Document

Aleksej Dvojnev

Algorithm 11 (A11)

Cognitive Systems / Autonomous Systems / Hybrid Intelligence

2026

A11 — Structural Architecture Specification

Technical Specification Document

Author: Aleksej Dvojnev

Project: Algorithm 11 (A11)

Category: Cognitive Systems / Autonomous Systems / Hybrid Intelligence

Year: 2026

Industry Challenges and Motivation

Modern AI systems, autonomous robotics, and intelligent infrastructure face several fundamental challenges:

- **Lack of deterministic reasoning**
- Most AI systems rely on probabilistic or stochastic methods, making them unpredictable in high-stakes environments.
- **Unstable behavior under uncertainty**
- When inputs are incomplete, noisy, or contradictory, existing systems often fail silently or produce incoherent outputs.
- **No separation between semantic and operational reasoning**
- Current architectures mix meaning, facts, planning, and execution into a single undifferentiated process, making verification and debugging difficult.
- **No structural guarantees**
- Most reasoning systems cannot guarantee coherence, feasibility, alignment, or bounded recursion.
- **Difficult integration across heterogeneous components**
- AI pipelines often combine LLMs, symbolic systems, planners, and controllers without a unifying structural model.
- **Lack of built-in safety mechanisms**
- Few architectures provide rollback, contradiction resolution, or invariant enforcement.

A11 addresses these challenges by providing:

- a **universal structural model** for reasoning,
- a **deterministic execution cycle**,
- a **branched semantic layer** for meaning and factual grounding,
- a **linear operational layer** for planning and validation,
- **two integration nodes** ensuring coherence,
- **three operators** ensuring stability,
- **bounded recursion**,
- **strict invariants**,

- **predictable, verifiable behavior**.

Positioning Statement

A11 is designed as a universal structural model capable of supporting the future development of AI systems, autonomous robotics, and large-scale intelligent infrastructure. It provides the deterministic, safe, and coherent reasoning foundation that modern intelligent systems require.

Table of Contents (Canonical, PDF-Ready)

0. Preface

0.1 Purpose of the Specification

This document defines the canonical engineering specification of the A11 architecture.

Its purpose is to provide:

- a complete structural description of A11,
- formal mathematical definitions,
- transition and operator models,
- execution semantics,
- implementation guidance,
- verification and compliance requirements.

The specification serves as the authoritative reference for all implementations, research, and extensions of A11.

0.2 Scope

This specification covers:

- the Core Layer (L1–L4),
- the Adaptive Layer (L5–L11),
- the initialization stage (S0),
- structural invariants,
- operators (Balance, Constraint, Rollback),
- recursion model,
- validity conditions,
- diagrams, examples, and use cases.

It does **not** prescribe domain-specific knowledge, ontologies, or application-level semantics.

A11 is a reasoning architecture, not a domain model.

0.3 Canonical Status

This document defines the **canonical** and **normative** form of A11.

All implementations must conform to:

- the fixed 12-stage geometry (S0 + L1–L11),
- the branching Core Layer,
- the linear Adaptive Layer,
- the two integration nodes (L4, L7),
- deterministic transitions,
- bounded recursion,
- operator semantics,
- structural invariants.

Any deviation invalidates compliance and voids all guarantees.

0.4 Document Structure

The specification is organized into the following major sections:

- **Formal Definitions**
- **Structural and Mathematical Models**
- **Transition and Operator Models**
- **Execution Cycle (Pseudocode)**
- **Diagrams and Examples**
- **Guarantees and Proof Sketches**
- **Implementation Notes**
- **Verification and Compliance**
- **Use Cases and Limitations**
- **Future Extensions**

This structure ensures clarity, modularity, and ease of reference for engineers, researchers, and auditors.

Reference Materials The supplementary materials are available at:https://github.com/gormenz-svg/algorithm-11

1. Introduction

1.1 Motivation

Modern reasoning systems require:

- determinism,
- structural safety,
- coherence under uncertainty,
- bounded recursion,
- interpretable internal structure.

A11 was designed to meet these requirements through a hybrid architecture combining semantic grounding and operational structuring.

1.2 Architectural Philosophy

A11 is built on four foundational principles:

1. **Separation of semantic and operational reasoning**
2. (Core Layer vs. Adaptive Layer)
3. **Parallel evaluation of meaning and facts**
4. (Wisdom ↔ Knowledge)
5. **Deterministic integration and correction**
6. (L4, L7, operators)
7. **Bounded, safe, recursive refinement**
8. (Adaptive Layer only)

This philosophy ensures stability, clarity, and reproducibility.

1.3 High-Level Overview

A11 consists of:

- **S0** — initialization
- **L1–L4** — semantic reasoning (branching)

- **L5–L11** — operational reasoning (linear)
- **two integration nodes** — L4 (semantic), L7 (operational)
- **three operators** — Balance, Constraint, Rollback
- **deterministic transitions**
- **bounded recursion**

The architecture transforms ambiguous or uncertain input into a coherent, feasible, validated realization.

1.4 Canonical Constraints

A11 imposes strict constraints:

- geometry cannot be modified,
- no bypass of L4 or L7,
- no recursion in L1–L4 or L11,
- operators cannot alter structure,
- transitions must be deterministic,
- recursion must be bounded,
- realization must be validated.

These constraints guarantee safety and reproducibility.

1.5 Relationship to Other Systems

A11 is:

- **not** a learning system,
- **not** a probabilistic model,
- **not** a stochastic search engine,
- **not** an ontology or knowledge base.

A11 can integrate with:

- LLMs,
- RL agents,

- symbolic systems,
- knowledge graphs,
- multi-agent frameworks.

But A11 itself remains a **deterministic reasoning architecture**.

2. Formal Definitions (Pages-friendly, Canonical Version)

This section introduces the formal terminology and structural primitives used throughout the specification.

All subsequent models, transition rules, and pseudocode rely on these definitions.

2.1 Structural Level

A **structural level** L_i is a functional reasoning component of the A11 architecture.

Each level is defined as a tuple:

L_i = (InputDomain, OutputDomain, TransformationFunction, Constraints)

Where:

- **InputDomain** — admissible input space
- **OutputDomain** — output space
- **TransformationFunction** — the function executed by the level
- **Constraints** — structural rules that apply to this level

Levels are indexed:

i = 1…11

Note: A11 contains **11 functional levels** (L_1–L_{11}) and **12 structural stages** (S_0–S_{11}). The initialization stage S_0 is not a level.

2.2 Architectural Layer

A **layer** is a grouping of levels with shared structural properties.

Two layers exist:

- **Core Layer:** L_1, L_2, L_3, L_4
- **Adaptive Layer:** $L_5, L_6, L_7, L_8, L_9, L_{10}, L_{11}$

The Core Layer is stable and non-adaptive.

The Adaptive Layer is dynamic and fractal.

2.3 Functional Role

A **functional role** R_j defines the subsystem responsible for executing a level.

Roles:

- **Will**
- **Wisdom**
- **Knowledge**
- **Comprehension**

Each level L_i is associated with exactly one role.

Roles describe **function**, not biological or model-specific identity.

2.4 Structural Operator

A **structural operator** O_k is a rule-based mechanism that modifies structural state.

Operators:

- **Balance**
- **Constraint**
- **Rollback**

Each operator has:

- **ActivationConditions** — when the operator is triggered
- **StateTransformationRules** — how the operator modifies the state

Operators may act across multiple levels.

2.5 Structural State

The **structural state** S is the complete representation of the system at a given moment.

It consists of **12 components**:

$S = \{S_0, S_1, S_2, \ldots, S_{11}\}$

Where:

- **S_0** — initialization stage (input normalization)
- **S_1–S_{11}** — states associated with levels L_1–L_{11}

Each S_i may contain:

- context
- constraints
- intermediate results
- structural metadata

2.6 Transition Function

A **transition function** defines how the system moves from one stage to the next.

General form:

Transition(S_i) → S_{i+1}

Transitions may be:

- forward
- balanced
- constrained
- recursive
- rollback-triggered

Transition rules are defined in Section 8.

2.7 Fractal Sub-Level

A **fractal sub-level** is a recursive instance of a structural level.

Notation:

$L_i^{(n)}$ — the n-th recursive instance of level L_i

Each sub-level:

- inherits the same structure
- uses the same operators
- returns a stable result to its parent

Fractal expansion is defined in Section 7.

2.8 Realization

Realization is the final structural output of the architecture.

Notation:

Realization = S_{11}

Realization must satisfy:

- structural coherence
- constraint compliance
- alignment with Level L_1 (Will)

2.9 Structural Invariant

A **structural invariant** is a property that must hold for all valid executions of A11.

Examples:

- level order is fixed
- operators do not alter level identity
- rollback always returns to the Core Layer
- balance is globally applicable

Formal proofs appear in Section 13.

2.10 Execution Cycle

An **execution cycle** is a complete traversal:

$\mathbf{S_0 \to L_1 \to L_2 \to L_3 \to \ldots \to L_{11} \to S_0}$

Cycles may contain:

- recursion

- operator-driven corrections
- rollback events

A formal example is provided in Section 12.

3. Structural Model of A11 (Revised Canonical Architecture)

This section defines the structural organization of A11 using the canonical geometry of the Core Layer.

The Core Layer is non-linear and structurally fixed; it cannot be reduced to a sequential pipeline.

This geometry forms the foundation for all reasoning, stability, and transition mechanisms.

3.1 Overview of the Structural Model

A11 consists of:

- **4 Core Levels** arranged as a structural figure
- **7 Adaptive Levels** arranged linearly
- **parallel processing channels** in the Core Layer
- **an integration node** that connects the Core and Adaptive Layers
- **a deterministic transition model**
- **a fractal expansion mechanism** in the Adaptive Layer

The Core Layer is **non-adaptive** and defines the system's direction, grounding, evaluation, and structural integration.

The Adaptive Layer is **dynamic**, **recursive**, and **expansive**.

3.2 Canonical Geometry of the Core Layer

The Core Layer is defined by the following structural figure:

```
(L1) Will
 /     \
/       \
(L3) Knowledge       (L2) Wisdom
             \        /
              \      /
        (L4) Comprehension
                 |
                 v
        transition to L5–L11
```

This geometry is **canonical** and replaces the previous linear interpretation.

3.3 Functional Roles of Core Levels

L1 — Will (Top Node)

Defines direction, intent, and the governing objective.

Will provides **independent input** to both Knowledge and Wisdom.

L3 — Knowledge (Left Branch)

Processes factual grounding, data, and objective information.

Operates **independently** from Wisdom.

L2 — Wisdom (Right Branch)

Processes evaluation, coherence, alignment, and high-level judgment.

Operates **independently** from Knowledge.

L4 — Comprehension (Integration Node)

Integrates outputs of Knowledge and Wisdom.

Resolves contradictions, balances perspectives, and produces a structured cognitive space.

Comprehension is the **only valid gateway** into the Adaptive Layer.

3.4 Parallel Processing Model (L2 and L3)

Levels L2 (Wisdom) and L3 (Knowledge) operate in **parallel**, not sequentially.

Properties:

- both receive Will as input
- both produce independent outputs

- neither depends on the other
- both must complete before L4 can execute

This creates a **dual-channel evaluation system**, increasing stability and reducing error propagation.

3.5 Integration Model (L4)

L4 receives two inputs:

- Output(Knowledge)
- Output(Wisdom)

L4 performs:

- structural alignment
- contradiction resolution
- semantic integration
- context construction
- preparation for dynamic reasoning

L4 is the **structural stabilizer** of the entire architecture.

3.6 Transition from Core to Adaptive Layer

Only L4 can initiate the transition to the Adaptive Layer.

Transition rule:

```
If Comprehension is stable:
    proceed to L5
Else:
    apply Balance or Rollback
```

This ensures that the Adaptive Layer always receives a coherent, integrated state.

3.7 Adaptive Layer (L5–L11)

The Adaptive Layer remains linear:

L5 → L6 → L7 → L8 → L9 → L10 → L11

It performs:

- expansion
- constraint
- balancing
- practical reasoning
- validation
- realization

The Adaptive Layer may invoke **fractal recursion**. Fractal recursion is bounded and must terminate at a stable sub-level before returning to the parent level.

3.8 Structural Dependencies

Dependencies in the Core Layer:

- L1 → L2
- L1 → L3
- L2 → L4
- L3 → L4

Dependencies in the Adaptive Layer remain sequential.

The Core Layer and Adaptive Layer are connected **only through L4**.

3.9 Structural Invariants of the Core Layer

1. **Will is always the top node**
2. **Knowledge and Wisdom always operate in parallel**
3. **Comprehension always integrates both branches**
4. **No level may bypass L4**
5. **Rollback always returns to the Core Layer**
6. **Operators cannot alter the geometry**

These invariants ensure stability and determinism.

3.10 Summary of the Structural Model

The revised structural model defines:

- a **non-linear Core Layer**
- parallel processing of Knowledge and Wisdom
- Comprehension as the integration and stabilization node
- a single transition point into the Adaptive Layer
- a deterministic, layered architecture
- a foundation for operators, transitions, and recursion

This geometry is the **canonical form** of A11 and is used throughout the rest of the specification.
The structural model defines 12 stages (S_0–S_{11}) corresponding to initialization and the 11 functional levels.

4. Mathematical State Model (Revised Canonical Architecture)

This section defines the mathematical representation of the A11 state under the revised Core Layer geometry.

Unlike linear models, the Core Layer state is **branched**, with parallel state components that converge at an integration node.

The state model describes:

- how information is stored,
- how parallel branches evolve,
- how integration occurs,
- how the system transitions into the Adaptive Layer.

4.1 Definition of the Global State

The global state S represents the complete internal condition of A11 at a given moment.

It consists of 12 components:

S = {S0, S1, S2, S3, S4, S5, …, S11}

Where:

- **S0** — initial intent state
- **S1** — state of Will

- **S2** — state of Wisdom (right branch)
- **S3** — state of Knowledge (left branch)
- **S4** — integrated state of Comprehension
- **S5–S11** — states of the Adaptive Layer

Each state component Si is a structured object:

Si = (Input, Output, Context, Constraints, Metadata)

This structure is uniform across all levels.

4.2 Branching State Model of the Core Layer

The Core Layer contains a **branching state topology**:

```
                S1 (Will)
                /       \
               /         \
     S3 (Knowledge)     S2 (Wisdom)
               \         /
                \       /
              S4 (Comprehension)
```

This topology defines:

- **two parallel state evolutions** (S2 and S3)
- **a single integration state** (S4)
- **a single entry point** (S1)
- **a single exit point** (S4 → S5)**

This branching structure is a **structural invariant**.

4.3 State Domains

Each level Li has a corresponding state domain:

- **StateDomain(L1)** — intent vectors, directional parameters
- **StateDomain(L2)** — evaluation metrics, alignment criteria
- **StateDomain(L3)** — factual grounding, retrieved data
- **StateDomain(L4)** — integrated cognitive structures

- **StateDomain(L5–L11)** — dynamic reasoning, constraints, validation, realization

Although domains differ, the **state structure is uniform**.

4.4 Parallel State Evolution (S2 and S3)

After Will produces S1, the state branches:

```
S1 → S2 (Wisdom)
S1 → S3 (Knowledge)
```

Properties:

- S2 and S3 evolve **independently**
- both receive the same Input = Output(S1)
- neither depends on the other
- both must complete before S4 can be computed

This creates a **dual-channel evaluation system**.

4.5 Integration State (S4)

S4 is computed only when both branches are available:

```
S4 = Integrate(S2, S3)
```

Integration includes:

- contradiction resolution
- semantic alignment
- structural balancing
- context construction
- preparation for dynamic reasoning

S4 must satisfy:

- coherence
- feasibility
- alignment with S1
- structural completeness

If integration fails, operators intervene (Balance or Rollback).

4.6 Transition to Adaptive Layer

Only S4 can produce S5:

```
If S4 is stable:
    S5 = Transition(S4)
Else:
    apply Balance or Rollback
```

This ensures that the Adaptive Layer receives a **fully integrated** state.

4.7 State Invariants

The following invariants must hold:

Invariant 1 — Branching Integrity

S2 and S3 must both originate from S1.

Invariant 2 — Parallel Independence

S2 and S3 must not depend on each other.

Invariant 3 — Integration Requirement

S4 must be computed from both S2 and S3.

Invariant 4 — No Bypass

No state may transition to S5 without passing through S4.

Invariant 5 — Operator Safety

Operators may modify states but cannot alter the branching geometry.

Invariant 6 — Rollback Boundaries

Rollback resets S1–S4 but does not modify S5–S11 directly.

4.8 Recursive State Model (Adaptive Layer)

The Adaptive Layer (S5–S11) retains the linear and fractal properties defined earlier.

Recursive states:

Si(n) = (Input, Output, Context, Constraints, Metadata)

Recursive execution must:

- remain bounded
- satisfy constraints
- return stable results

The Core Layer does **not** recurse.

4.9 Realization State

The final output of A11 is:

Realization = S11

It must satisfy:

- alignment with S1
- coherence validated by S4 and S7
- feasibility validated by S6 and S9
- structural correctness validated by S10

Realization initiates a new cycle.

4.10 Summary of the State Model

The revised state model defines:

- a **branched Core Layer state topology**
- parallel evolution of Wisdom and Knowledge
- Comprehension as an integration state
- a single transition point into the Adaptive Layer
- strict invariants ensuring stability
- deterministic transitions
- recursive expansion in the Adaptive Layer

This model reflects the canonical geometry of A11 and forms the basis for the transition, operator, and execution models.

5. Transition Model (Revised Canonical Architecture)

This section defines the transition rules governing how state flows through the revised Core Layer and into the Adaptive Layer.

Unlike linear architectures, A11 uses a **branched transition model** in the Core Layer, followed by a **linear transition model** in the Adaptive Layer.

Transitions are deterministic given:

- the current state,
- structural constraints,
- operator activation conditions.

5.1 General Transition Function

A transition from level Li to Lj is defined as:

Transition(Si) → Sj

Where:

- **Si** — state at level Li
- **Sj** — resulting state at level Lj

Transitions may include:

- direct transformation
- parallel branching
- integration
- operator-driven modification
- recursive expansion (Adaptive Layer only)
- rollback correction

5.2 Core Layer Transition Geometry

The Core Layer uses a **branched transition model**:

```
                S1 (Will)
                /       \
               /         \
     S3 (Knowledge)    S2 (Wisdom)
               \         /
                \       /
              S4 (Comprehension)
```

This defines four canonical transitions:

1. **S1 → S2** (Will → Wisdom)
2. **S1 → S3** (Will → Knowledge)
3. **S2 → S4** (Wisdom → Comprehension)
4. **S3 → S4** (Knowledge → Comprehension)

These transitions are **parallel** and **convergent**, not sequential.

5.3 Parallel Transitions (S1 → S2 and S1 → S3)

After Will produces S1, the system branches:

```
S2 = Transition(S1) via Wisdom
S3 = Transition(S1) via Knowledge
```

Properties:

- both transitions occur independently
- both must complete before integration
- operators may act on either branch
- neither branch depends on the other

This creates a dual-channel evaluation mechanism.

5.4 Convergent Transition (S2 + S3 → S4)

Comprehension (L4) executes only when both branches are available:

```
If S2.ready AND S3.ready:
    S4 = Integrate(S2, S3)
```

```
Else:
    wait or apply operators
```
Integration includes:

- contradiction resolution
- semantic alignment
- structural balancing
- context construction

If integration fails:

- Balance is applied
- If unresolved → Rollback

5.5 Transition from Core to Adaptive Layer (S4 → S5)

Only S4 can transition into the Adaptive Layer:

```
If S4 is stable:
    S5 = Transition(S4)
Else:
    apply Balance or Rollback
```
This ensures that the Adaptive Layer receives a coherent, integrated state.

5.6 Adaptive Layer Transitions (Linear Model)

The Adaptive Layer retains the linear structure:

```
S5 → S6 → S7 → S8 → S9 → S10 → S11
```
Transitions may include:

- expansion (L5)
- constraint (L6, L9)
- balancing (L7)
- practical reasoning (L8)
- validation (L10)
- realization (L11)

The Adaptive Layer may invoke **fractal recursion**.

5.7 Types of Transitions

A11 supports four types of transitions:

1. Forward Transition

Normal progression (branched in Core, linear in Adaptive).

2. Balanced Transition

Operator Balance modifies state before continuing.

3. Constrained Transition

Operator Constraint restricts expansion or action spaces.

4. Rollback Transition

Execution returns to the Core Layer when instability is detected.

These transitions may occur in combination.

5.8 Transition Validity Conditions

A transition is valid only if:

1. **Input is admissible for the target level**
2. **No structural invariants are violated**
3. **Operator conditions are satisfied**
4. **Parallel branches are complete (for S4)**
5. **Recursive branches return stable results**
6. **Rollback conditions are not active**

If any condition fails:

- Balance is applied
- or Constraint is applied
- or Rollback is triggered

5.9 Core Layer Ordering Rules

Rule 1 — Will is always first

No level may precede L1.

Rule 2 — Knowledge and Wisdom are parallel

Neither may depend on the other.

Rule 3 — Comprehension requires both branches

S4 cannot be computed from only S2 or only S3.

Rule 4 — No bypass of Comprehension

No transition may go directly from S1, S2, or S3 to S5.

Rule 5 — Operators cannot alter geometry

They modify state, not structure.

5.10 Adaptive Layer Ordering Rules

The Adaptive Layer remains strictly sequential:

```
L5 → L6 → L7 → L8 → L9 → L10 → L11
```
Rollback may return to:

- L1
- L2
- L3
- L4

but never to L5–L11.

5.11 Summary of the Transition Model

The revised transition model defines:

- a **branched transition system** in the Core Layer
- parallel evolution of Wisdom and Knowledge

- convergent integration in Comprehension
- a single transition point into the Adaptive Layer
- linear transitions in the Adaptive Layer
- deterministic operator-driven corrections
- bounded recursion
- rollback safety

This model reflects the canonical geometry of A11 and forms the foundation for the operator and execution models.

6. Structural Operators (Revised Canonical Architecture)

Structural operators are rule-based mechanisms that modify the internal state of A11 during execution.

Under the revised Core Layer geometry, operators must account for **parallel branches**, **integration logic**, and **structural invariants** that govern the flow of information.

Operators modify **state**, not **structure**.

They cannot alter the canonical geometry of the Core Layer.

A11 defines three operators:

1. **Balance**
2. **Constraint**
3. **Rollback**

Each operator Ok is defined as:

Ok = (ActivationConditions, StateTransformationRules)

6.1 Balance Operator (B)

Purpose

Restore coherence when contradictions, inconsistencies, or structural conflicts appear in the state.

Balance is the **primary stabilizer** of the Core Layer, especially at the integration node (L4).

Scope

Balance may act on:

- S2 (Wisdom branch)
- S3 (Knowledge branch)
- S4 (Comprehension integration)
- any Adaptive Layer state

Balance is the **only operator** that can act across both branches simultaneously.

Activation Conditions

Balance is activated when **any** of the following conditions are true:

1. **BranchConflict(S2, S3) = true**
2. (Wisdom and Knowledge produce incompatible outputs)
3. **MisalignmentWithWill(S2, S3, S1) = true**
4. (branches diverge from intent)
5. **IntegrationFailure(S4) = true**
6. (Comprehension cannot integrate S2 and S3)
7. **SemanticContradiction(Si) = true**
8. (logical or structural contradiction)
9. **OperatorConflict(Si) = true**
10. (Constraint and expansion produce incompatible states)

State Transformation Rules

Balance performs:

1. **Identify conflicting elements** in S2, S3, or S4
2. **Modify or remove** elements that violate coherence
3. **Reconcile** branch outputs
4. **Reconstruct** a coherent integrated state S4'
5. **Return** the balanced state

General form:

Si' = Balance(Si)

Balance does not alter the branching geometry.

6.2 Constraint Operator (C)

Purpose

Restrict expansion or action spaces to maintain feasibility and prevent runaway reasoning.

Constraint is especially important in the Adaptive Layer, but also applies to the Core Layer branches.

Scope

Constraint may act on:

- S2 (Wisdom)
- S3 (Knowledge)
- S5–S11 (Adaptive Layer)

Constraint **does not** act on S1 (Will) or S4 (Comprehension) directly, but may influence their downstream effects.

Activation Conditions

Constraint is activated when **any** of the following conditions are true:

1. **BranchOverexpansion(S2 or S3) = true**
2. (one branch grows beyond allowed bounds)
3. **FeasibilityViolation(Si) = true**
4. (generated options are unrealistic or impossible)
5. **ResourceLimitExceeded(Si) = true**
6. (practical or computational limits violated)
7. **RuleViolation(Si) = true**
8. (domain rules, safety rules, or structural rules violated)
9. **RecursionDepthExceeded(Si) = true**
10. (Adaptive Layer recursion exceeds allowed depth)

State Transformation Rules

Constraint performs:

1. **Identify elements** violating constraints
2. **Prune or restrict** the expansion space
3. **Recompute** the feasible subset
4. **Produce** a constrained state Si'
5. **Return** the constrained state

General form:

Si' = ApplyConstraints(Si)

Constraint ensures that both branches and the Adaptive Layer remain grounded.

6.3 Rollback Operator (R)

Purpose

Restore stability when the system becomes inconsistent, unstable, or unable to proceed.

Rollback is the **last-resort operator**.

Scope

Rollback always returns execution to the **Core Layer**, specifically to one of:

- L1 (Will)
- L2 (Wisdom)
- L3 (Knowledge)
- L4 (Comprehension)

Rollback **never** targets L5–L11.

Activation Conditions

Rollback is activated when **any** of the following conditions are true:

1. **BalanceFailed(S4) = true**
2. (integration cannot be restored)
3. **ConstraintFailed(Si) = true**
4. (constraints cannot be satisfied)

5. **StructuralInvariantViolation = true**
6. (branching geometry violated)
7. **UnstableBranches(S2, S3) = true**
8. (branches diverge uncontrollably)
9. **CriticalError(Si) = true**
10. (state becomes undefined or invalid)

State Transformation Rules

Rollback performs:

1. **Reset** S1–S4
2. **Preserve** structural metadata
3. **Discard** unstable Adaptive Layer states
4. **Restart** execution from the chosen Core Level

General form:

Si → Rollback → S1 or S2 or S3 or S4

Rollback ensures the system never enters an unrecoverable state.

6.4 Operator Interaction Rules

Rule 1 — Balance precedes Constraint

If both are triggered, Balance is applied first.

Rule 2 — Constraint precedes Rollback

Rollback is only triggered if Constraint cannot restore feasibility.

Rule 3 — Operators cannot alter geometry

They modify **state**, not **structure**.

Rule 4 — Operators may act on branches independently

S2 and S3 may receive different operator actions.

Rule 5 — Operators may act on integration

S4 is a primary target for Balance.

Rule 6 — Operators may act recursively

Adaptive Layer recursion may invoke operators independently.

6.5 Operator Determinism

Given the same:

- input state
- structural constraints
- operator conditions

Operators produce the **same output**.

This ensures:

- reproducibility
- predictability
- formal verifiability

Operator determinism is proven in Section 13.

6.6 Summary of Operator Specification

The revised operator model defines:

- how operators act on a **branched Core Layer**
- how they stabilize parallel channels
- how they integrate conflicting outputs
- how they maintain structural invariants
- how they govern transitions into the Adaptive Layer
- how they ensure coherence, feasibility, and stability

Operators are the **dynamic control system** of A11.

7. Execution Cycle (Pseudocode, Revised Canonical Architecture)

This section defines the complete execution cycle of A11 using the revised Core Layer geometry.

The pseudocode reflects:

- parallel processing of Wisdom and Knowledge,
- integration in Comprehension,
- operator-driven stabilization,
- linear and recursive execution in the Adaptive Layer,
- rollback to the structural Core Layer.

The pseudocode is language-agnostic and suitable for implementation in any environment.

7.1 High-Level Execution Loop

```
function A11_Execute(initial_input):

    S0 = InitializeIntent(initial_input)

    repeat:

        S1 = Level1_Will(S0)

        (S2, S3) = ExecuteCoreBranches(S1)

        S4 = IntegrateCoreBranches(S2, S3)

        if S4.RollbackRequired:
            S0 = HandleRollback(S4)
            continue

        S11 = ExecuteAdaptiveLayer(S4)

        if S11.RealizationReady:
            return S11.Realization

        if S11.RollbackRequired:
            S0 = HandleRollback(S11)
            continue

    end repeat
```

7.2 Core Layer Branch Execution (Parallel Processing)

```
function ExecuteCoreBranches(S1):

    S2 = Level2_Wisdom(S1)
    S2 = ApplyOperators(S2)

    S3 = Level3_Knowledge(S1)
    S3 = ApplyOperators(S3)

    return (S2, S3)
```

Properties:

- S2 and S3 are computed independently
- operators may act on either branch
- both branches must complete before integration

7.3 Integration Node (Comprehension)

```
function IntegrateCoreBranches(S2, S3):

    if not (S2.ready and S3.ready):
        return WaitForBranches()

    S4 = Level4_Comprehension(S2, S3)
    S4 = ApplyOperators(S4)

    return S4
```

Integration includes:

- contradiction resolution
- semantic alignment
- structural balancing
- context construction

If integration fails, S4.RollbackRequired is set.

7.4 Adaptive Layer Execution (Linear + Recursive)

```
function ExecuteAdaptiveLayer(S4):

    S5 = Level5_ProjectiveFreedom(S4)
    S5 = ApplyOperators(S5)

    S6 = Level6_ProjectiveConstraint(S5)
    S6 = ApplyOperators(S6)

    S7 = Level7_Balance(S6)
    S7 = ApplyOperators(S7)

    S8 = Level8_PracticalFreedom(S7)
    S8 = ApplyOperators(S8)

    S9 = Level9_PracticalConstraint(S8)
    S9 = ApplyOperators(S9)

    S10 = Level10_Foundation(S9)
    S10 = ApplyOperators(S10)

    S11 = Level11_Realization(S10)
    S11 = ApplyOperators(S11)

    return S11
```

7.5 Operator Application Logic

```
function ApplyOperators(S):

    if BalanceConditions(S):
        S = Balance(S)

    if ConstraintConditions(S):
        S = ApplyConstraints(S)

    if RollbackConditions(S):
        S.RollbackRequired = true
        return S

    return S
```

Operators may act:

- independently on S2 and S3

- jointly on S4
- recursively in the Adaptive Layer

7.6 Recursive Expansion Logic (Adaptive Layer Only)

```
function RecursiveExpansion(S, level):

    if not RecursionConditions(S):
        return S

    depth = 0
    S_current = S

    while RecursionConditions(S_current):

        if depth > MAX_RECURSION_DEPTH:
            S_current.RollbackRequired = true
            return S_current

        S_current = ExecuteSubLevel(level, S_current)
        S_current = ApplyOperators(S_current)

        depth = depth + 1

    return S_current
```

The Core Layer **never recurses**.

7.7 Sub-Level Execution (Adaptive Layer)

```
function ExecuteSubLevel(level, S):

    switch(level):

        case 5:
            return Level5_ProjectiveFreedom(S)

        case 6:
            return Level6_ProjectiveConstraint(S)

        case 7:
            return Level7_Balance(S)
        case 8:
            return Level8_PracticalFreedom(S)

        case 9:
            return Level9_PracticalConstraint(S)

        case 10:
            return Level10_Foundation(S)

        case 11:
            return Level11_Realization(S)
```

7.8 Rollback Handling

```
function HandleRollback(S):

    if S.RollbackTarget == 1:
        return ResetToLevel1(S)

    if S.RollbackTarget == 2:
        return ResetToLevel2(S)

    if S.RollbackTarget == 3:
        return ResetToLevel3(S)

    if S.RollbackTarget == 4:
        return ResetToLevel4(S)
```

Rollback always returns to the **Core Layer**, never to L5–L11.

7.9 Realization Logic

```
function Level11_Realization(S10):

    if not RealizationConditions(S10):
        S10.RollbackRequired = true
        S10.RollbackTarget = 4
        return S10

    S11 = ProduceRealization(S10)
    S11.RealizationReady = true

    return S11
```

7.10 Summary of the Execution Cycle

The revised pseudocode defines:

- a **branched Core Layer execution model**,
- parallel processing of Wisdom and Knowledge,
- integration in Comprehension,
- operator-driven stabilization,
- linear and recursive Adaptive Layer execution,
- rollback to the structural Core Layer,
- realization as the final output.

This pseudocode is the **reference implementation** of the canonical A11 architecture.

8. Structural Diagrams (ASCII, Revised Canonical Architecture)

This section provides formal ASCII diagrams representing the canonical geometry of A11.

The diagrams reflect:

- the **branched Core Layer**,
- the **dual weighting system** in the Adaptive Layer,
- the **two balancing nodes** (L4 and L7),
- the **symmetry between semantic and operational reasoning**,
- the **full execution flow** from Will to Realization.

All diagrams are PDF-friendly and stable across editors.

8.1 Full Structural Hierarchy (L1–L11)

```
+--------------------------------------------------+
|                  A11 Structure                   |
+--------------------------------------------------+
| L1  | Will                                        |
|---------------------------------------------------|
| L3  | Knowledge        |        L2  | Wisdom      |
|--------------------------------------------------|
| L4  | Comprehension (Integration Node)           |
|--------------------------------------------------|
| L5  | Projective Freedom                         |
| L6  | Projective Constraint                      |
| L7  | Balance (Operational Integration Node)     |
| L8  | Practical Freedom                          |
| L9  | Practical Constraint                       |
| L10 | Foundation                                 |
| L11 | Realization                                |
+--------------------------------------------------+
```

8.2 Canonical Geometry of the Core Layer (Semantic Weighting System)

```
                 (L1) Will
                   /     \
                  /       \
     (L3) Knowledge       (L2) Wisdom
                  \       /
                   \     /
             (L4) Comprehension
```

Key properties:

- L2 and L3 operate **in parallel**
- L4 integrates both branches
- L4 is the **only gateway** to the Adaptive Layer

8.3 Canonical Geometry of the Adaptive Layer (Operational Weighting System)

The Adaptive Layer contains **two weighting pairs** and a **balancing node**:

```
        Projective Pair (Expansion vs Constraint)
             L6 <---------> L5

                     |
                     v
```

```
                 (L7) Balance

                     |
                     v

        Practical Pair (Expansion vs Constraint)
             L9 <---------> L8
```
This mirrors the Core Layer:

- L5–L6 ↔ L3–L2
- L7 ↔ L4
- L8–L9 ↔ second operational weighting

8.4 Dual Weighting System (Full Symmetry Diagram)

```
                 (L1) Will
                   /     \
                  /       \
     (L3) Knowledge       (L2) Wisdom
                  \       /
                   \     /
             (L4) Comprehension
                     |
                     v
             (L5) Projective Freedom
                     |
             (L6) Projective Constraint
                     |
             (L7) Balance
                     |
             (L8) Practical Freedom
                     |
             (L9) Practical Constraint
                     |
             (L10) Foundation
                     |
             (L11) Realization
```
This diagram shows the **semantic weighting system** (L1–L4) and the **operational weighting system** (L5–L9).

8.5 Parallel Branching and Convergent Integration (Core Layer)

```
         S1 (Will)
         /      \
        /        \
      S3          S2
   (Knowledge) (Wisdom)
        \        /
         \      /
     S4 (Comprehension)
```

This is the **semantic stabilizer**.

8.6 Dual Weighting in the Adaptive Layer

```
         S5 (Projective Freedom)
                   |
         S6 (Projective Constraint)
                   |
         S7 (Balance)
                   |
         S8 (Practical Freedom)
                   |
         S9 (Practical Constraint)
```

This is the **operational stabilizer**.

8.7 Full Execution Cycle (Structural View)

```
+--------------------------------------------------+
|                    START                         |
+--------------------------------------------------+
             |
             v
+---------------------------+
|        CORE LAYER         |
+---------------------------+
| L1 -> (L2 || L3) -> L4    |
|  Will   Wisdom Knowledge  |
|           \     /         |
|          Comprehension      |
+-----------------------------+
             |
             v
+-----------------------------+
|       ADAPTIVE LAYER        |
+-----------------------------+
| L5 -> L6 -> L7 -> L8 ->     |
| L9 -> L10 -> L11            |
+-----------------------------+
             |
             v
+-----------------------------+
|        REALIZATION          |
+-----------------------------+
             |
             v
+-----------------------------+
|        NEW CYCLE            |
+-----------------------------+
```

8.8 Symmetry Between Core and Adaptive Layers

```
      SEMANTIC WEIGHTING (Core Layer)
      Knowledge <-----> Wisdom
               \       /
                \     /
             Comprehension

      OPERATIONAL WEIGHTING (Adaptive Layer)
   Projective <-----> Projective
     Freedom            Constraint
               \       /
                \     /
                Balance
               /       \
   Practical <----------> Practical
     Freedom                Constraint
```

This symmetry is intentional and foundational.

8.9 Fractal Expansion (Recursive Sub-Levels)

```
+-----------------------+
|       Level Li        |
+----------+------------+
           |
           v
+-----------------------+
|      Li (1)           |
+----------+------------+
           |
           v
+-----------------------+
|      Li (2)           |
+----------+------------+
           |
           v
+-----------------------+
|      Li (3)           |
+----------+------------+
           |
           v
+-----------------------+
|    Stable Result      |
+-----------------------+
```

Only Adaptive Layer levels recurse.

8.10 Summary of Structural Diagrams

The revised diagrams show:

- the **branched Core Layer**
- the **dual weighting system**
- the **two balancing nodes**
- the **symmetry between semantic and operational reasoning**
- the **linear + recursive Adaptive Layer**
- the **full execution cycle**

This geometry is the canonical structural representation of A11.

9. Formal Example of a Full Cycle (Revised Canonical Architecture)

This section provides a complete, step-by-step example of how A11 processes an input through all levels, using the revised structural geometry:

- **parallel semantic weighting** (Knowledge ↔ Wisdom),
- **integration in Comprehension**,
- **projective and practical operational weighting**,
- **balancing in L7**,
- **realization in L11**.

The example is abstract and domain-agnostic, suitable for engineering documentation.

9.1 Initial Input

Let the system receive the following abstract input:

Input_0 = "Achieve objective X under uncertain conditions."

This input contains:

- a goal (X),
- uncertainty,
- no predefined method.

A11 must transform this into a coherent, feasible, validated realization.

9.2 Level 1 — Will (Intent Formation)

Will extracts the governing direction:

```
S1.Intent = "Achieve X"
S1.Criteria = {coherence, feasibility, alignment}
S1.Priority = High
```

Will does **not** generate solutions.

It defines **direction and evaluation criteria**.

S1 is broadcast to both branches:

```
S1 → S2 (Wisdom)
S1 → S3 (Knowledge)
```

9.3 Levels 2 and 3 — Parallel Semantic Weighting

L3 — Knowledge (Left Branch)

Knowledge retrieves and structures factual grounding:

```
S3.Facts = {known constraints, resources, environment}
S3.Data = {historical patterns, domain rules}
S3.Structure = preliminary factual map
```

L2 — Wisdom (Right Branch)

Wisdom evaluates meaning, alignment, and high-level coherence:

```
S2.Evaluation = {ethical, strategic, long-term alignment}
S2.DirectionCheck = "Does X make sense?"
S2.Risks = identified conceptual risks
```

Both branches operate **independently**.

9.4 Level 4 — Comprehension (Semantic Integration)

Comprehension integrates S2 and S3:

```
S4 = Integrate(S2, S3)
```

Integration produces:

- a unified semantic model,
- resolved contradictions,
- a coherent interpretation of X,
- a structured problem space.

Example integrated output:

```
S4.Model = "X is achievable if conditions A, B, C are
satisfied."
S4.Contradictions = none
S4.Ready = true
```

If contradictions existed, Balance would intervene.

9.5 Level 5 — Projective Freedom (Expansion of Possibilities)

L5 generates possible approaches:

```
S5.Options = {approach1, approach2, approach3, ...}
S5.Space = expanded conceptual solution space
```

This is **creative expansion**, not yet constrained.

9.6 Level 6 — Projective Constraint (Filtering Possibilities)

L6 filters S5:

```
S6.FeasibleOptions = Filter(S5.Options, S4.Model)
S6.Removed = {options violating constraints}
```

This produces a **feasible conceptual set**.

9.7 Level 7 — Balance (Operational Integration)

L7 integrates the projective pair:

```
S7 = Integrate(S5, S6)
```

Output:

```
S7.StructuredPlan = "Conceptual plan with feasible
boundaries"
S7.Stability = high
```

L7 is the **operational analogue** of L4.

9.8 Level 8 — Practical Freedom (Practical Expansion)

L8 expands S7 into practical variants:

```
S8.Variants = {methodA, methodB, methodC}
S8.Resources = estimated resource needs
```

This is **practical expansion**, not conceptual.

9.9 Level 9 — Practical Constraint (Practical Filtering)

L9 filters S8:

```
S9.ValidVariants = Filter(S8.Variants, real-world
constraints)
S9.Removed = {impractical or resource-invalid variants}
```

This produces a **realistic operational set**.

9.10 Level 10 — Foundation (Validation and Structuring)

L10 validates the remaining variants:

```
S10.Validated = Validate(S9.ValidVariants)
S10.Structure = final structured operational plan
```

Validation includes:

- feasibility,
- safety,
- consistency with S1,
- alignment with S4 and S7.

9.11 Level 11 — Realization (Final Output)

If S10 is valid:

```
S11.Realization = Produce(S10.Structure)
S11.Ready = true
```

Example realization:

```
"Execute method B with parameters P1, P2, P3."
```

This is the final output of the cycle.

9.12 Summary of the Full Cycle

The example demonstrates:

- **parallel semantic reasoning** (L2–L3),

- **semantic integration** (L4),
- **projective operational weighting** (L5–L6),
- **operational integration** (L7),
- **practical operational weighting** (L8–L9),
- **validation** (L10),
- **realization** (L11).

The cycle is:

```
Will → (Wisdom || Knowledge) → Comprehension
→ Projective Freedom → Projective Constraint → Balance
→ Practical Freedom → Practical Constraint
→ Foundation → Realization
```

This is the canonical execution path of A11.

10. Mathematical Guarantees (Revised Canonical Architecture)

This section defines the formal guarantees provided by A11.

These guarantees arise from:

- the **branched semantic weighting system** (L1–L4),
- the **dual operational weighting system** (L5–L9),
- the **two integration nodes** (L4 and L7),
- the **deterministic transition model**,
- the **operator system**,
- the **structural invariants**.

A11 guarantees **stability, coherence, feasibility, determinism, bounded recursion, and structural correctness** for every execution cycle.

10.1 Guarantee 1 — Structural Determinism

A11 guarantees that **the same input always produces the same structural execution path**.

Formally:

For any input I:

```
Path(A11, I) = constant
```
Because:

- the Core Layer geometry is fixed,
- transitions are deterministic,
- operators have deterministic rules,
- recursion is bounded.

This ensures reproducibility and verifiability.

10.2 Guarantee 2 — Semantic Coherence (L1–L4)

The branched Core Layer ensures that **semantic contradictions cannot propagate**.

Let:

- S2 = Wisdom output
- S3 = Knowledge output
- S4 = Comprehension integration

Then:

```
Coherence(S4) = true
```
Because:

- S2 and S3 are independently validated,
- L4 integrates them with contradiction resolution,
- Balance enforces coherence if needed,
- Rollback prevents unstable states from propagating.

Thus, A11 guarantees that **all downstream reasoning is semantically coherent**.

10.3 Guarantee 3 — Operational Coherence (L5–L9)

The dual weighting system ensures that **operational contradictions cannot propagate**.

Let:

- S5–S6 = projective pair
- S8–S9 = practical pair

- S7 = operational integration

Then:

```
Coherence(S7) = true
Coherence(S9) = true
```

Because:

- expansion and constraint are always paired,
- L7 integrates projective reasoning,
- L9 filters practical reasoning,
- operators enforce feasibility and stability.

Thus, A11 guarantees that **all operational reasoning is coherent and feasible**.

10.4 Guarantee 4 — No Contradiction Propagation

A11 guarantees that **no contradiction can pass through an integration node**.

Formally:

If:

```
Contradiction(S2, S3) = true
```

Then:

```
S4 = Balanced(S2, S3)
or
Rollback
```

Similarly for S5–S6 and S8–S9.

Thus:

```
Contradiction(S4) = false
Contradiction(S7) = false
```

This ensures global stability.

10.5 Guarantee 5 — Bounded Recursion

Only Adaptive Layer levels may recurse.

Let depth(n) be recursion depth.

A11 guarantees:

```
depth(n) ≤ MAX_RECURSION_DEPTH
```

If exceeded:

```
Rollback
```

Thus, recursion is **safe, bounded, and non-explosive**.

10.6 Guarantee 6 — Feasibility Preservation

A11 guarantees that **no infeasible solution can reach Realization**.

Let:

- S6 = projective feasibility filter
- S9 = practical feasibility filter
- S10 = final validation

Then:

```
Feasible(S11.Realization) = true
```

Because:

- infeasible options are removed in S6,
- impractical options are removed in S9,
- invalid structures are removed in S10.

Thus, A11 guarantees **feasible realizations only**.

10.7 Guarantee 7 — Alignment with Intent (Will)

A11 guarantees that the final realization is aligned with the original intent S1.

Formally:

```
Alignment(S11.Realization, S1.Intent) = true
```

Because:

- Wisdom checks alignment,
- Comprehension integrates alignment,

- Balance enforces alignment,
- Foundation validates alignment.

Thus, A11 guarantees **intent-aligned outputs**.

10.8 Guarantee 8 — Stability Under Uncertainty

A11 guarantees stability even when:

- input is incomplete,
- data is noisy,
- constraints are unclear,
- environment is uncertain.

Because:

- parallel semantic weighting reduces uncertainty,
- integration nodes stabilize contradictions,
- operators enforce coherence and feasibility,
- rollback prevents unstable propagation.

Thus, A11 guarantees **stable reasoning under uncertainty**.

10.9 Guarantee 9 — Structural Safety

A11 guarantees that **structural invariants cannot be violated**.

Invariants include:

- L2 and L3 are parallel
- L4 integrates both
- L7 integrates operational pairs
- no bypass of L4 or L7
- rollback always returns to Core Layer
- recursion is bounded
- operators cannot alter geometry

Thus, A11 guarantees **structural correctness**.

10.10 Guarantee 10 — Realization Validity

The final output S11 is guaranteed to be:

- coherent (semantic)
- feasible (operational)
- validated (structural)
- aligned (intent)
- stable (operator-verified)

Formally:

```
Valid(S11.Realization) = true
```

This is the strongest guarantee A11 provides.

10.11 Summary of Mathematical Guarantees

A11 guarantees:

- **deterministic execution**,
- **semantic coherence**,
- **operational coherence**,
- **no contradiction propagation**,
- **bounded recursion**,
- **feasibility preservation**,
- **alignment with intent**,
- **stability under uncertainty**,
- **structural safety**,
- **valid realization**.

These guarantees arise directly from the canonical geometry of A11 and form the foundation for its reliability as a reasoning architecture.

11. Proof Sketches (Revised Canonical Architecture)

This section provides proof sketches for the mathematical guarantees defined in Section 10.

The goal is not to present full formal proofs, but to outline the logical and structural reasoning that ensures each guarantee holds under the canonical geometry of A11.

Each proof sketch relies on:

- the **branched semantic weighting system** (L1–L4),
- the **dual operational weighting system** (L5–L9),
- the **integration nodes** (L4 and L7),
- the **operator system** (Balance, Constraint, Rollback),
- the **structural invariants**,
- the **deterministic transition model**.

11.1 Proof Sketch for Structural Determinism

Claim:

For any input I, A11 always follows the same structural execution path.

Reasoning:

1. The Core Layer geometry is fixed:

```
L1 → (L2 || L3) → L4
```

2. The Adaptive Layer geometry is fixed:

```
L5 → L6 → L7 → L8 → L9 → L10 → L11
```

3. Operators cannot alter geometry (Invariant 5).
4. Transitions are deterministic functions:

```
Si → Sj = f(Si)
```

5. Rollback returns to a fixed structural point (L1–L4), not arbitrary states.

Conclusion:

No execution path variation is possible.

Thus, **structural determinism holds**.

11.2 Proof Sketch for Semantic Coherence (L1–L4)

Claim:

Contradictions between Wisdom and Knowledge cannot propagate beyond L4.

Reasoning:

1. S2 and S3 are independent semantic channels.
2. L4 integrates S2 and S3 using contradiction resolution.
3. If contradictions remain:
    - Balance modifies S2, S3, or S4
    - or Rollback resets the Core Layer
4. No transition to S5 is allowed unless S4 is stable (Invariant 3).

Conclusion:

Semantic contradictions are eliminated or contained.

Thus, **semantic coherence holds**.

11.3 Proof Sketch for Operational Coherence (L5–L9)

Claim:

Operational contradictions cannot propagate beyond L7 or L9.

Reasoning:

1. L5 expands possibilities; L6 constrains them.
2. L7 integrates the projective pair (L5–L6).
3. L8 expands practical variants; L9 constrains them.
4. Operators enforce feasibility at each stage.
5. No transition to L10 is allowed unless S9 is stable.

Conclusion:

Operational contradictions are eliminated or contained.

Thus, **operational coherence holds**.

11.4 Proof Sketch for No Contradiction Propagation

Claim:

Contradictions cannot pass through integration nodes (L4, L7).

Reasoning:

1. Integration nodes require both inputs to be coherent.
2. If contradictions exist:
    - Balance resolves them
    - or Rollback resets the system
3. Integration nodes have the property:

```
Integrate(Sa, Sb) = stable only if Coherent(Sa, Sb)
```

Conclusion:

Contradictions cannot propagate.

Thus, **the guarantee holds**.

11.5 Proof Sketch for Bounded Recursion

Claim:

Recursion depth is always bounded.

Reasoning:

1. Only Adaptive Layer levels may recurse.
2. MAX_RECURSION_DEPTH is a fixed constant.
3. If recursion exceeds the bound:

   ```
   Rollback
   ```

4. Rollback resets to the Core Layer, preventing infinite loops.

Conclusion:

Recursion is bounded and safe.

Thus, **bounded recursion holds**.

11.6 Proof Sketch for Feasibility Preservation

Claim:

No infeasible solution can reach Realization.

Reasoning:

1. L6 removes infeasible conceptual options.
2. L9 removes infeasible practical options.
3. L10 validates feasibility again.
4. If any infeasibility remains:
   - Constraint prunes it
   - or Rollback resets the system

5. L11 only accepts validated inputs.

Conclusion:

Only feasible solutions reach S11.

Thus, **feasibility preservation holds**.

11.7 Proof Sketch for Alignment with Intent

Claim:

The final realization always aligns with Will (S1).

Reasoning:

1. Wisdom checks alignment at L2.
2. Comprehension integrates alignment at L4.
3. Balance enforces alignment if deviations appear.
4. Foundation (L10) validates alignment again.
5. Realization (L11) requires alignment to be true.

Conclusion:

Alignment is enforced at multiple levels.

Thus, **intent alignment holds**.

11.8 Proof Sketch for Stability Under Uncertainty

Claim:

A11 remains stable even with incomplete or noisy input.

Reasoning:

1. Parallel semantic channels reduce uncertainty.
2. Integration nodes stabilize contradictions.
3. Operators enforce coherence and feasibility.
4. Rollback prevents unstable propagation.

5. Adaptive Layer weighting absorbs uncertainty.

Conclusion:

The architecture is robust to uncertainty.

Thus, **stability under uncertainty holds**.

11.9 Proof Sketch for Structural Safety

Claim:

Structural invariants cannot be violated.

Reasoning:

1. Geometry is fixed and cannot be modified.
2. Operators modify state, not structure.
3. Transitions are restricted by invariants.
4. Rollback enforces structural correctness.
5. No bypass of L4 or L7 is allowed.

Conclusion:

Structural safety is guaranteed.

11.10 Proof Sketch for Realization Validity

Claim:

The final output S11 is always valid.

Reasoning:

1. Semantic coherence enforced at L4.
2. Operational coherence enforced at L7 and L9.
3. Validation enforced at L10.
4. Realization requires all conditions to be satisfied.
5. If any condition fails → Rollback.

Conclusion:

Realization is always coherent, feasible, aligned, and validated.

11.11 Summary of Proof Sketches

The proof sketches demonstrate that A11's guarantees arise from:

- its **canonical geometry**,
- its **integration nodes**,
- its **operator system**,
- its **deterministic transitions**,
- its **structural invariants**,
- its **bounded recursion**,
- its **dual weighting systems**.

Together, these elements ensure that A11 is:

- stable,
- coherent,
- deterministic,
- safe,
- aligned,
- feasible,
- and mathematically grounded.

12. Formal Definitions (Revised Canonical Architecture)

This section defines the formal mathematical objects used throughout the A11 specification.

All definitions are canonical and apply to every implementation of A11, regardless of domain or modality.

Definitions are grouped into:

1. **Structural Definitions**

2. **State Definitions**
3. **Transition Definitions**
4. **Operator Definitions**
5. **Integration Definitions**
6. **Recursion Definitions**
7. **Validity Definitions**

These definitions form the formal basis for the guarantees and proofs in Sections 10 and 11.

12.1 Structural Definitions

Definition 12.1.1 — Levels

A11 consists of 12 levels:

```
Core Layer:     L1, L2, L3, L4
Adaptive Layer: L5, L6, L7, L8, L9, L10, L11
```

Definition 12.1.2 — Core Layer Geometry

The Core Layer is a directed acyclic graph:

```
L1 → {L2, L3}
{L2, L3} → L4
```

Definition 12.1.3 — Adaptive Layer Geometry

The Adaptive Layer is a linear chain:

```
L5 → L6 → L7 → L8 → L9 → L10 → L11
```

Definition 12.1.4 — Integration Nodes

There are two integration nodes:

- **L4** — semantic integration
- **L7** — operational integration

Both require multiple inputs.

12.2 State Definitions

Definition 12.2.1 — State

A state at level Li is a tuple:

```
Si = (Input, Output, Context, Constraints, Metadata)
```

Definition 12.2.2 — Global State

The global state S is:

```
S = {S0, S1, S2, …, S11}
```

Definition 12.2.3 — Branch States

Semantic branches:

```
S2 = Wisdom state
S3 = Knowledge state
```

Operational branches:

```
S5—S6 = projective pair
S8—S9 = practical pair
```

12.3 Transition Definitions

Definition 12.3.1 — Transition Function

A transition from Li to Lj is:

```
T(Li → Lj): Si → Sj
```

Definition 12.3.2 — Parallel Transitions

Parallel transitions occur when:

```
T(L1 → L2) and T(L1 → L3)
```

execute independently.

Definition 12.3.3 — Convergent Transitions

A convergent transition requires multiple inputs:

```
T({S2, S3} → S4)
T({S5, S6} → S7)
```

12.4 Operator Definitions

Definition 12.4.1 — Operator

An operator Ok is a function:

```
Ok: Si → Si'
```

Definition 12.4.2 — Balance Operator

Balance resolves contradictions:

```
Balance(Si) = Si' such that Coherent(Si') = true
```

Definition 12.4.3 — Constraint Operator

Constraint restricts state:

```
Constraint(Si) = Si' such that Feasible(Si') = true
```

Definition 12.4.4 — Rollback Operator

Rollback resets the Core Layer:

```
Rollback(Si) → S1 or S2 or S3 or S4
```

12.5 Integration Definitions

Definition 12.5.1 — Integration Function

Integration of two states Sa and Sb is:

```
Integrate(Sa, Sb) = Sc
```

Definition 12.5.2 — Integration Validity

Integration is valid iff:

```
Coherent(Sa, Sb) = true
```

Definition 12.5.3 — Integration Nodes

- **L4** integrates S2 and S3
- **L7** integrates S5 and S6

12.6 Recursion Definitions

Definition 12.6.1 — Recursive Sub-Level

A recursive sub-level is:

```
Li(n) = recursive instance of Li
```

Definition 12.6.2 — Recursion Depth

Depth is:

```
depth = max(n)
```

Definition 12.6.3 — Recursion Bound

Recursion is bounded:

```
depth ≤ MAX_RECURSION_DEPTH
```

12.7 Validity Definitions

Definition 12.7.1 — Semantic Coherence

A state S is semantically coherent if:

```
No contradictions exist in semantic structure
```

Definition 12.7.2 — Operational Coherence

A state S is operationally coherent if:

```
No contradictions exist in operational structure
```

Definition 12.7.3 — Feasibility

A state S is feasible if:

```
S satisfies domain constraints
```

Definition 12.7.4 — Alignment

A state S is aligned if:

```
Aligned(S, S1.Intent) = true
```

Definition 12.7.5 — Valid Realization

A realization R is valid if:

```
Coherent(R) ∧ Feasible(R) ∧ Aligned(R) ∧ Stable(R)
```

12.8 Summary of Formal Definitions

This section defines:

- the structural objects of A11,
- the state model,
- the transition model,
- the operator system,
- the integration functions,
- the recursion model,
- the validity conditions.

These definitions form the mathematical foundation for:

- guarantees (Section 10),
- proof sketches (Section 11),
- implementation (Section 13),
- verification (Section 14).

13. Implementation Notes (Revised Canonical Architecture)

This section provides practical engineering guidance for implementing A11 in software systems.

It does not prescribe a specific programming language or platform.

Instead, it defines **implementation-critical constraints**, **recommended data structures**, **execution patterns**, and **failure-handling strategies**.

These notes ensure that any implementation of A11 remains faithful to the canonical architecture.

13.1 Core Principles for Implementation

Any implementation of A11 must satisfy the following principles:

1. **Structural invariants must be preserved**
    - Core Layer geometry cannot be altered
    - Adaptive Layer must remain linear
    - Integration nodes must remain convergent
2. **Operators must be deterministic**
    - same input → same output
3. **Parallelism must be explicit**
    - L2 and L3 must run independently
4. **Integration must be strict**
    - L4 and L7 must validate coherence
5. **Recursion must be bounded**
    - MAX_RECURSION_DEPTH must be enforced
6. **Rollback must be safe and complete**
    - rollback resets S1–S4 only

13.2 Recommended Data Structures

13.2.1 State Representation

Each state Si should be represented as a structured object:

```
State {
    Input
    Output
    Context
    Constraints
    Metadata
}
```

Recommended formats:

- JSON
- Protocol Buffers
- Typed classes/structs

13.2.2 Global State Container

Global state S should be stored in:

- a dictionary keyed by level index, or
- a fixed-size array of 12 elements.

Example:

```
S[0..11] = State
```

13.2.3 Operator Flags

Each state should include flags:

```
State {
    ...
    RollbackRequired: bool
    RollbackTarget: LevelID
    Stable: bool
}
```

13.3 Execution Model

13.3.1 Parallel Execution of L2 and L3

Implementations should use:

- threads,
- async tasks,
- coroutines,
- or parallel compute units

to execute:

```
S2 = Wisdom(S1)
S3 = Knowledge(S1)
```

in parallel.

13.3.2 Synchronization Point

L4 must wait for both branches:

```
wait until S2.ready AND S3.ready
```

13.3.3 Integration Logic

Integration must:

- detect contradictions
- merge structures
- produce a unified semantic model

Integration must be atomic.

13.4 Operator Implementation

13.4.1 Balance

Balance should:

- detect contradictions
- modify state minimally
- preserve semantic meaning
- ensure coherence

Recommended approach:

- rule-based conflict resolution
- weighted merging
- semantic unification

13.4.2 Constraint

Constraint should:

- prune infeasible elements
- enforce domain rules
- reduce search space

Recommended approach:

- constraint propagation
- rule filtering
- feasibility scoring

13.4.3 Rollback

Rollback must:

- reset S1–S4
- preserve metadata
- discard unstable Adaptive Layer states

Rollback must be:

- atomic
- idempotent
- safe

13.5 Integration Node Implementation

13.5.1 L4 — Semantic Integration

L4 must:

- merge S2 and S3
- resolve contradictions
- produce a coherent semantic model

Recommended techniques:

- graph merging
- semantic unification
- conflict resolution rules

13.5.2 L7 — Operational Integration

L7 must:

- merge S5–S6
- stabilize operational structure
- prepare for practical reasoning

Recommended techniques:

- weighted averaging
- constraint-aware merging

- operational scoring

13.6 Recursion Implementation

13.6.1 Recursive Sub-Levels

Recursive execution should be implemented as:

```
function ExecuteRecursive(level, state):
    for depth in 1..MAX_RECURSION_DEPTH:
        state = Execute(level, state)
        if state.Stable:
            return state
    state.RollbackRequired = true
    return state
```

13.6.2 Recursion Safety

Implementations must:

- track recursion depth
- enforce upper bounds
- detect infinite loops

13.7 Error Handling

13.7.1 Soft Errors

Soft errors trigger:

- Balance
- Constraint

Examples:

- minor contradictions
- small feasibility violations

13.7.2 Hard Errors

Hard errors trigger:

- Rollback

Examples:

- structural invariant violation
- unresolvable contradiction
- recursion overflow

13.8 Logging and Traceability

Implementations should log:

- state transitions
- operator activations
- integration results
- rollback events
- recursion depth

Logs must be:

- timestamped
- structured
- machine-readable

13.9 Performance Considerations

13.9.1 Parallelism

Parallel execution of L2 and L3 significantly improves performance.

13.9.2 Pruning

Constraint operator reduces computational load by pruning infeasible branches early.

13.9.3 Caching

Caching intermediate results is recommended for:

- S2
- S3
- S5

- S8

13.10 Implementation Checklist

A correct implementation must:

- [ ] preserve Core Layer geometry
- [ ] preserve Adaptive Layer sequence
- [ ] implement parallel L2/L3
- [ ] implement integration nodes
- [ ] implement all operators
- [ ] enforce recursion bounds
- [ ] support rollback
- [ ] maintain determinism
- [ ] ensure coherence and feasibility
- [ ] validate realization

13.11 Summary of Implementation Notes

This section provides:

- structural constraints,
- recommended data structures,
- execution patterns,
- operator logic,
- recursion handling,
- error handling,
- performance guidelines.

Together, these notes ensure that any implementation of A11 remains faithful to the canonical architecture and preserves all mathematical guarantees.

14. Verification and Testing (Revised Canonical Architecture)

This section defines the verification and testing methodology required to ensure that any implementation of A11 conforms to the canonical architecture.

Verification focuses on **correctness**, while testing focuses on **behavior**.

The goal is to ensure:

- structural correctness,
- operator correctness,
- transition correctness,
- recursion safety,
- determinism,
- coherence,
- feasibility,
- alignment,
- stability.

14.1 Verification Overview

Verification ensures that the implementation satisfies:

1. **Structural invariants**
2. **Transition rules**
3. **Operator semantics**
4. **Integration correctness**
5. **Recursion bounds**
6. **Determinism**
7. **Validity conditions**

Verification is **architecture-level**, not domain-level.

14.2 Structural Verification

14.2.1 Core Layer Geometry Verification

The implementation must satisfy:

```
L1 → {L2, L3}
{L2, L3} → L4
```
Tests:

- L2 and L3 must not depend on each other
- L4 must depend on both L2 and L3
- no bypass of L4 is allowed

14.2.2 Adaptive Layer Geometry Verification

The implementation must satisfy:

```
L5 → L6 → L7 → L8 → L9 → L10 → L11
```
Tests:

- transitions must be strictly sequential
- rollback must not target L5–L11

14.2.3 Integration Node Verification

L4 and L7 must:

- accept multiple inputs
- validate coherence
- reject contradictions

14.3 Transition Verification

14.3.1 Deterministic Transitions

For any state Si:

```
T(Si) must be deterministic
```
Test:

- run T(Si) multiple times
- outputs must match bit-for-bit

14.3.2 Parallel Transition Verification

L2 and L3 must:

- run independently
- produce consistent results regardless of scheduling

Test:

- run L2 and L3 in different orders
- outputs must remain identical

14.3.3 Convergent Transition Verification

L4 must:

- wait for both S2 and S3
- reject incomplete inputs

14.4 Operator Verification

14.4.1 Balance Operator Verification

Balance must:

- detect contradictions
- resolve them
- produce coherent output

Tests:

- inject semantic contradictions
- verify that Balance resolves them
- verify that unresolved contradictions trigger Rollback

14.4.2 Constraint Operator Verification

Constraint must:

- prune infeasible elements
- enforce domain rules

Tests:

- inject infeasible options
- verify pruning

- verify feasibility of output

14.4.3 Rollback Operator Verification

Rollback must:

- reset S1–S4
- preserve metadata
- discard unstable Adaptive Layer states

Tests:

- inject hard errors
- verify rollback target
- verify state reset

14.5 Integration Verification

14.5.1 Semantic Integration (L4)

Tests:

- provide coherent S2 and S3 → expect stable S4
- provide contradictory S2 and S3 → expect Balance or Rollback
- verify that S4 is always coherent

14.5.2 Operational Integration (L7)

Tests:

- provide coherent S5 and S6 → expect stable S7
- provide contradictory S5 and S6 → expect Balance or Rollback
- verify that S7 is always coherent

14.6 Recursion Verification

14.6.1 Depth Bound Verification

Test:

- force recursion

- verify that depth never exceeds MAX_RECURSION_DEPTH
- verify rollback on overflow

14.6.2 Stability Verification

Test:

- recursive sub-levels must converge
- unstable recursion must trigger rollback

14.7 Determinism Verification

14.7.1 Input-Output Determinism

For any input I:

```
All(I) must be identical across runs
```

Test:

- run full cycle multiple times
- compare S11 outputs

14.7.2 Branch Determinism

Parallel execution must not affect results.

Test:

- randomize scheduling of L2 and L3
- verify identical S4

14.8 Coherence Verification

14.8.1 Semantic Coherence

Test:

- inject semantic noise
- verify that L4 or Balance resolves it

14.8.2 Operational Coherence

Test:

- inject operational contradictions
- verify that L7 or Balance resolves it

14.9 Feasibility Verification

14.9.1 Projective Feasibility

Test:

- inject infeasible conceptual options
- verify L6 pruning

14.9.2 Practical Feasibility

Test:

- inject impractical variants
- verify L9 pruning

14.10 Alignment Verification

14.10.1 Intent Alignment

Test:

- modify S2 to misalign with S1
- verify Balance correction
- verify L10 validation

14.11 Stability Verification

14.11.1 Noise Injection

Inject noise into:

- S2
- S3
- S5
- S8

Verify:

- Balance stabilizes
- Constraint prunes
- Rollback triggers if necessary

14.11.2 Stress Testing

Test:

- large input spaces
- deep recursion
- high branching complexity

Verify:

- no structural violations
- no infinite loops
- no unstable states

14.12 Realization Verification

14.12.1 Validity Check

S11 must satisfy:

- coherence
- feasibility
- alignment
- stability

Test:

- inject invalid S10
- verify rejection
- verify rollback

14.13 Verification Checklist

A correct implementation must pass:

- [ ] Core Layer geometry tests
- [ ] Adaptive Layer geometry tests
- [ ] operator correctness tests
- [ ] integration tests
- [ ] recursion tests
- [ ] determinism tests
- [ ] coherence tests
- [ ] feasibility tests
- [ ] alignment tests
- [ ] stability tests
- [ ] realization validity tests

14.14 Summary of Verification and Testing

This section defines:

- structural verification,
- transition verification,
- operator verification,
- integration verification,
- recursion verification,
- determinism verification,
- coherence and feasibility verification,
- alignment and stability verification,
- realization verification.

Together, these procedures ensure that any implementation of A11 is:

- correct,
- stable,
- deterministic,

- coherent,
- feasible,
- aligned,
- safe,
- and fully compliant with the canonical architecture.

15. Glossary (Revised Canonical Architecture)

This glossary defines all canonical terms used throughout the A11 specification.

Each definition is concise, formal, and implementation-agnostic.

15.1 Structural Terms

A11

A hierarchical reasoning architecture consisting of 12 levels, combining semantic weighting, operational weighting, integration nodes, and deterministic transitions.

Core Layer

Levels L1–L4.

Performs semantic grounding, evaluation, and integration.

Adaptive Layer

Levels L5–L11.

Performs expansion, constraint, balancing, validation, and realization.

Level (Li)

A functional processing unit with a defined role, input domain, output domain, and transition rules.

Integration Node

A level that merges multiple inputs into a coherent state.

There are two: L4 (semantic) and L7 (operational).

Branch

A parallel processing path in the Core Layer (Wisdom, Knowledge) or paired operational paths in the Adaptive Layer (L5–L6, L8–L9).

15.2 Core Layer Terms

Will (L1)

The governing intent, direction, and evaluation criteria for the entire cycle.

Wisdom (L2)

Semantic evaluation, alignment, coherence, and high-level judgment.

Knowledge (L3)

Factual grounding, data retrieval, structural information.

Comprehension (L4)

Semantic integration of Wisdom and Knowledge into a unified model.

15.3 Adaptive Layer Terms

Projective Freedom (L5)

Generation of conceptual possibilities.

Projective Constraint (L6)

Filtering conceptual possibilities for feasibility.

Balance (L7)

Operational integration of L5 and L6; stabilizes the projective structure.

Practical Freedom (L8)

Generation of practical variants and actionable options.

Practical Constraint (L9)

Filtering practical variants for real-world feasibility.

Foundation (L10)

Validation and structuring of the final operational plan.

Realization (L11)

Production of the final output aligned with intent and validated for feasibility.

15.4 State Terms

State (Si)

A structured object representing the internal condition at level Li.

Global State (S)

The set of all states S0–S11.

Semantic State

States S2, S3, S4 — representing semantic reasoning.

Operational State

States S5–S9 — representing operational reasoning.

Stable State

A state that satisfies coherence, feasibility, and alignment conditions.

15.5 Transition Terms

Transition

A deterministic function mapping Si → Sj.

Parallel Transition

Simultaneous independent transitions (L1 → L2 and L1 → L3).

Convergent Transition

A transition requiring multiple inputs (L2 + L3 → L4).

Recursive Transition

A bounded repetition of a level in the Adaptive Layer.

15.6 Operator Terms

Operator

A function modifying state without altering structure.

Balance Operator

Resolves contradictions and restores coherence.

Constraint Operator

Prunes infeasible or invalid elements.

Rollback Operator

Resets execution to a Core Layer level when stability cannot be restored.

15.7 Validity Terms

Coherence

Absence of contradictions in semantic or operational structure.

Feasibility

Satisfaction of domain constraints and real-world limitations.

Alignment

Consistency with the governing intent defined in S1.

Stability

A state that is coherent, feasible, aligned, and recursion-safe.

Valid Realization

A final output satisfying all validity conditions.

15.8 Invariant Terms

Structural Invariant

A rule that cannot be violated (e.g., L2 and L3 must be parallel).

Recursion Bound

The maximum allowed recursion depth.

No-Bypass Rule

No level may bypass L4 or L7.

15.9 Execution Terms

Cycle

A full execution from S0 to S11.

Sub-Cycle

A recursive execution within a single Adaptive Layer level.

Restart

Beginning a new cycle after realization.

15.10 Summary of Glossary

This glossary defines:

- structural components,
- semantic and operational levels,
- state objects,
- transitions,
- operators,
- validity conditions,
- invariants,
- execution concepts.

These definitions form the canonical vocabulary of A11 and ensure consistency across implementation, verification, and documentation.

16. Canonical Notation (Revised Canonical Architecture)

This section defines the canonical notation used throughout the A11 specification.

All mathematical expressions, diagrams, pseudocode, and implementation references must use this notation.

The goal is to ensure:

- consistency,
- clarity,
- unambiguity,
- formal rigor,
- interoperability across implementations.

16.1 Level Notation

16.1.1 Levels

Levels are denoted as:

```
Li
```

Where:

- **i** ∈ **{1…11}**
- L1–L4 = Core Layer
- L5–L11 = Adaptive Layer

16.1.2 Integration Nodes

Integration nodes are denoted:

```
L4ᵢ (semantic integration)
L7ᵢ (operational integration)
```

The subscript “$_i$” indicates “integration node”.

16.2 State Notation

16.2.1 State at Level Li

A state is denoted:

```
Si
```

16.2.2 State Structure

A state is a tuple:

```
Si = (In, Out, Ctx, Cstr, Meta)
```

Where:

- **In** — input
- **Out** — output
- **Ctx** — context
- **Cstr** — constraints
- **Meta** — metadata

16.2.3 Global State

The global state is:

```
S = {S0, S1, S2, …, S11}
```

16.3 Transition Notation

16.3.1 Basic Transition

A transition from Li to Lj is:

```
Li → Lj
```

16.3.2 Transition Function

The transition function is:

```
T(Li → Lj): Si → Sj
```

16.3.3 Parallel Transitions

Parallel transitions are denoted:

```
L1 ⇢ {L2, L3}
```

or explicitly:

```
L1 → L2   and   L1 → L3
```

16.3.4 Convergent Transitions

Convergent transitions are denoted:

```
{L2, L3} ⇢ L4
{L5, L6} ⇢ L7
```

16.4 Operator Notation

16.4.1 Operator Application

An operator Ok applied to state Si is:

```
Ok(Si)
```

16.4.2 Balance Operator

```
B(Si)
```

16.4.3 Constraint Operator

```
C(Si)
```

16.4.4 Rollback Operator

```
R(Si)
```

16.4.5 Operator Pipeline

Operators applied in sequence:

```
Si' = C(B(Si))
```

16.5 Integration Notation

16.5.1 Integration Function

Integration of states Sa and Sb:

```
I(Sa, Sb)
```

16.5.2 Integration at L4

```
S4 = I(S2, S3)
```

16.5.3 Integration at L7

```
S7 = I(S5, S6)
```

16.6 Recursion Notation

16.6.1 Recursive Sub-Level

A recursive instance of Li is:

```
Li(n)
```

Where:

- **n** = recursion depth
- **0 ≤ n ≤ MAX_RECURSION_DEPTH**

16.6.2 Recursive Execution

```
Li → Li(1) → Li(2) → … → Li(n)
```

16.6.3 Recursion Termination

Termination condition:

```
Stable(Si(n))
```

16.7 Validity Notation

16.7.1 Coherence

```
Coherent(Si)
```

16.7.2 Feasibility

```
Feasible(Si)
```

16.7.3 Alignment

```
Aligned(Si, S1)
```

16.7.4 Stability

```
Stable(Si) = Coherent(Si) ∧ Feasible(Si) ∧ Aligned(Si, S1)
```

16.7.5 Valid Realization

```
Valid(S11)
```

16.8 Error and Recovery Notation

16.8.1 Soft Error

```
Err_soft(Si)
```

16.8.2 Hard Error

```
Err_hard(Si)
```

16.8.3 Rollback Trigger

```
Err_hard(Si) ⇒ R(Si)
```

16.9 Execution Cycle Notation

16.9.1 Full Cycle

```
S0 → L1 → {L2, L3} → L4 → L5 → L6 → L7 → L8 → L9 → L10 → L11
```

16.9.2 Cycle with Operators

```
S0 → L1 → B(L2) || C(L3) → I(L2, L3) → … → R(L9) → L1
```

16.9.3 Cycle with Recursion

```
L8 → L8(1) → L8(2) → … → L8(n) → L9
```

16.10 Diagram Notation

16.10.1 Parallel Branches

```
A ⇢ {B, C}
```

16.10.2 Convergent Integration

```
{B, C} ⇢ D
```

16.10.3 Linear Chain

```
A → B → C → D
```

16.11 Summary of Canonical Notation

This section defines the canonical notation for:

- levels,
- states,
- transitions,
- operators,
- integration,
- recursion,
- validity,
- errors,
- execution cycles,
- diagrams.

This notation is mandatory for:

- all A11 specifications,
- all implementations,
- all proofs,
- all diagrams,
- all publications.

17. Canonical Examples (Revised Canonical Architecture)

This section provides canonical examples demonstrating how A11 processes different types of inputs using the full architecture:

- semantic weighting (L1–L4),

- operational weighting (L5–L9),
- validation and realization (L10–L11),
- operators (Balance, Constraint, Rollback),
- recursion,
- integration nodes.

Each example is domain-agnostic and focuses on the **mechanics** of A11.

17.1 Example A — Simple Goal Under Uncertainty

Input$_0$:

"Find a stable way to achieve objective X."

17.1.1 Core Layer

L1 — Will

Extracts intent:

```
Intent = "Achieve X"
Criteria = {coherence, feasibility}
```

L2 — Wisdom (parallel)

Evaluates meaning:

```
Evaluation = "X is meaningful if conditions A, B hold"
Risks = {R1}
```

L3 — Knowledge (parallel)

Retrieves facts:

```
Facts = {A=true, B=unknown, C=false}
```

L4 — Comprehension

Integrates:

```
Model = "X is achievable if B is resolved"
```

17.1.2 Adaptive Layer

L5 — Projective Freedom

Generates conceptual options:

```
Options = {O1, O2, O3}
```

L6 — Projective Constraint

Filters:

```
Feasible = {O1, O3}
```

L7 — Balance

Integrates:

```
ConceptualPlan = O1 with fallback O3
```

L8 — Practical Freedom

Generates variants:

```
Variants = {V1, V2}
```

L9 — Practical Constraint

Filters:

```
Valid = {V2}
```

L10 — Foundation

Validates:

```
ValidatedPlan = V2
```

L11 — Realization

Produces:

```
Realization = “Execute V2”
```

17.2 Example B — Contradictory Input (Balance Activation)

Input$_0$:

“Achieve Y, but Y contradicts known constraints.”

17.2.1 Core Layer

L1 — Will

Intent = “Achieve Y”

L2 — Wisdom

Evaluation = “Y is misaligned with long-term criteria”

L3 — Knowledge

Facts = “Y violates constraint C”

L4 — Comprehension

Integration detects contradiction:

```
Contradiction(S2, S3) = true
```

Balance Activation

Balance modifies S2 and S3:

```
ReframedIntent = “Achieve Y' (aligned variant of Y)”
```

Integration succeeds.

17.2.2 Adaptive Layer

Process continues normally with Y'.

17.3 Example C — Infeasible Options (Constraint Activation)

Input$_0$:

“Generate a plan Z with unlimited resources.”

17.3.1 Core Layer

L1–L4 produce a coherent semantic model.

17.3.2 Adaptive Layer

L5 — Projective Freedom

Generates unrealistic options:

```
Options = {Z1, Z2, Z3_unbounded}
```

L6 — Projective Constraint

Prunes:

```
Feasible = {Z1, Z2}
Removed = {Z3_unbounded}
```

Constraint ensures feasibility.

17.4 Example D — Recursive Expansion

Input$_0$:

“Find the optimal configuration for objective Q.”

17.4.1 Core Layer

L1–L4 produce a coherent model.

17.4.2 Adaptive Layer

L8 — Practical Freedom

Triggers recursion:

```
L8 → L8(1) → L8(2) → L8(3)
```

Each iteration refines variants.

Termination Condition

```
Stable(S8(3)) = true
```

L9 — Practical Constraint

Filters refined variants.

17.5 Example E — Rollback Trigger

Input$_0$:

“Perform task W with contradictory constraints.”

17.5.1 Core Layer

L1–L4 detect unresolved contradictions.

Balance fails

Contradiction persists.

Rollback Triggered

```
RollbackTarget = L1
```

System resets to Will and reframes the task.

17.6 Example F — Full Cycle Summary

This example shows the entire architecture in one compact flow.

```
S0 → L1
L1 ⇢ {L2, L3}
{L2, L3} ⇢ L4
L4 → L5 → L6 → L7 → L8 → L9 → L10 → L11
```

Operators intervene as needed:

```
Balance at L4
Constraint at L6, L9
Rollback at any unstable point
```

Realization is always:

- coherent,
- feasible,
- aligned,
- validated.

17.7 Summary of Canonical Examples

These examples demonstrate:

- parallel semantic reasoning,
- semantic integration,
- projective and practical operational weighting,
- operational integration,
- recursion,
- operator activation,
- rollback,
- full-cycle execution.

They serve as reference patterns for:

- implementation,
- testing,
- verification,
- teaching,
- documentation.

18. Canonical Diagrams (Graphical Specification)

This section defines the canonical graphical diagrams of A11.

All diagrams are described textually in a precise, implementation-agnostic format suitable for:

- vector graphics,
- UML tools,

- LaTeX/TikZ,
- PDF diagrams,
- engineering documentation.

Each diagram is a **normative element** of the A11 standard.

18.1 Diagram 1 — Canonical Core Layer Geometry

Purpose: Show the semantic weighting system and the branching structure.

Elements:

- Node L1 at the top center.
- Two downward diagonal arrows from L1 to L2 (right) and L3 (left).
- Two upward diagonal arrows from L2 and L3 converging into L4 (center bottom).

Textual Layout:

```
          [L1 Will]
           /      \
          /        \
[L3 Knowledge]   [L2 Wisdom]
          \        /
           \      /
     [L4 Comprehension]
```

Interpretation:

Parallel semantic branches converge into a semantic integration node.

18.2 Diagram 2 — Canonical Adaptive Layer Geometry

Purpose: Show the operational weighting system and the linear chain.

Elements:

- Vertical chain of nodes L5 → L6 → L7 → L8 → L9 → L10 → L11.
- L5–L6 form the projective pair.
- L8–L9 form the practical pair.
- L7 is the operational integration node.

Textual Layout:

```
[L5 Projective Freedom]
           |
[L6 Projective Constraint]
           |
[L7 Balance]
           |
[L8 Practical Freedom]
           |
[L9 Practical Constraint]
           |
[L10 Foundation]
           |
[L11 Realization]
```

18.3 Diagram 3 — Full Architecture (L1–L11)

Purpose: Show the entire A11 structure in one diagram.

Elements:

- Core Layer at the top (branching).
- Adaptive Layer below (linear).
- L4 connects Core to Adaptive.

Textual Layout:

```
                 [L1 Will]
                  /     \
                 /       \
     [L3 Knowledge]     [L2 Wisdom]
                 \       /
                  \     /
          [L4 Comprehension]
                  |
                  v
        [L5 Projective Freedom]
                  |
        [L6 Projective Constraint]
                  |
             [L7 Balance]
                  |
        [L8 Practical Freedom]
                  |
        [L9 Practical Constraint]
                  |
           [L10 Foundation]
                  |
           [L11 Realization]
```

18.4 Diagram 4 — Dual Weighting System

Purpose: Show the symmetry between semantic and operational weighting.

Elements:

- Upper weighting pair: L3 ↔ L2.
- Lower weighting pairs: L5 ↔ L6 and L8 ↔ L9.
- Integration nodes L4 and L7.

Textual Layout:

```
      Semantic Weighting (Core Layer)
         [L3] <---------> [L2]
               \       /
                \     /
              [L4 Integration]

      Operational Weighting (Adaptive Layer)
         [L6] <---------> [L5]
               \       /
                \     /
              [L7 Integration]
               /       \
         [L9] <---------> [L8]
```

18.5 Diagram 5 — Transition Model

Purpose: Show all transitions, including parallel, convergent, and linear.

Elements:

- Parallel transitions from L1.
- Convergent transitions into L4 and L7.
- Linear transitions in Adaptive Layer.

Textual Layout:

```
L1 → L2
L1 → L3
{L2, L3} → L4
L4 → L5 → L6
{L5, L6} → L7
L7 → L8 → L9 → L10 → L11
```

18.6 Diagram 6 — Operator Activation Points

Purpose: Show where Balance, Constraint, and Rollback may intervene.

Elements:

- Balance at L4 and L7.
- Constraint at L6 and L9.
- Rollback anywhere except L11.

Textual Layout:

```
Balance:   L4, L7
Constraint: L6, L9
Rollback:   L1–L10 (never L11)
```

Graphical representation:

Код

```
[L1] → [L2] → [L4] → [L5] → [L6] → [L7] → [L8] → [L9] → [L10]
→ [L11]
             ^B               ^C       ^B               ^C
^R
```

18.7 Diagram 7 — Recursion Model

Purpose: Show recursive sub-levels in Adaptive Layer.

Elements:

- Recursion only allowed in L5–L10.
- Depth bounded.

Textual Layout:

```
[L8] → [L8(1)] → [L8(2)] → … → [L8(n)] → [L9]
```

18.8 Diagram 8 — Error Handling and Recovery

Purpose: Show how errors propagate and how rollback resets the system.

Textual Layout:

```
Error at Lk:
    if soft → Balance or Constraint
    if hard → Rollback → L1/L2/L3/L4
```

Graphical:

```
[L8] --hard error--> [Rollback] → [L1]
```

18.9 Diagram 9 — Full Execution Cycle

Purpose: Show the entire cycle from S0 to S11.

Textual Layout:

```
S0 → L1 → {L2, L3} → L4 → L5 → L6 → L7 → L8 → L9 → L10 → L11
→ Restart
```

18.10 Summary of Canonical Diagrams

This section defines the canonical graphical representation of A11:

- Core Layer geometry
- Adaptive Layer geometry
- Full architecture
- Dual weighting system
- Transition model
- Operator activation
- Recursion model
- Error handling
- Full execution cycle

These diagrams are **normative** and must be used in all official A11 documents.

19. Canonical Use Cases (Revised Canonical Architecture)

This section defines the canonical use cases for A11.

These use cases describe **problem classes**, not domain-specific implementations.

Each use case demonstrates how A11's architecture provides:

- semantic grounding (L1–L4),
- operational structuring (L5–L9),
- validation and realization (L10–L11),
- stability under uncertainty,
- deterministic reasoning,
- coherent integration of parallel branches.

Use cases are grouped into **six canonical categories**.

19.1 Use Case Category A — Ambiguous or Underspecified Goals

Problem Class

Tasks where the objective is unclear, incomplete, or ambiguous.

Examples

- "Find the best way to approach X."
- "What is a reasonable strategy for Y?"
- "Help me understand how to begin Z."

Why A11 is suited

- L2 and L3 independently clarify meaning and facts.
- L4 integrates them into a coherent interpretation.
- L5–L9 explore and constrain possible approaches.
- L10–L11 validate and produce a stable plan.

Architectural Strength

A11 transforms ambiguity into structured clarity.

19.2 Use Case Category B — Multi-Constraint Reasoning

Problem Class

Tasks requiring balancing multiple constraints, trade-offs, or conflicting requirements.

Examples

- “Optimize X under constraints A, B, C.”
- “Find a feasible solution that satisfies all conditions.”
- “Balance cost, time, and quality.”

Why A11 is suited

- L3 handles factual constraints.
- L2 evaluates alignment and priorities.
- L4 resolves contradictions.
- L6 and L9 enforce feasibility.
- L7 integrates conceptual constraints.

Architectural Strength

A11 guarantees feasibility and alignment simultaneously.

19.3 Use Case Category C — High-Uncertainty Environments

Problem Class

Tasks with incomplete data, unknown variables, or unstable conditions.

Examples

- “Plan under uncertainty.”

- “Propose a strategy when information is missing.”
- “Handle conflicting or noisy inputs.”

Why A11 is suited

- Parallel semantic branches reduce uncertainty.
- Integration nodes stabilize contradictions.
- Operators enforce coherence.
- Rollback prevents unstable propagation.

Architectural Strength

A11 remains stable even when the environment is not.

19.4 Use Case Category D — Multi-Step Planning and Strategy

Problem Class

Tasks requiring structured planning, multi-step reasoning, or hierarchical decision-making.

Examples

- “Develop a multi-phase plan.”
- “Create a strategy with contingencies.”
- “Design a workflow or process.”

Why A11 is suited

- L5–L6 generate and constrain conceptual plans.
- L7 integrates them into a stable structure.
- L8–L9 generate and constrain practical steps.
- L10 validates the full plan.

Architectural Strength

A11 naturally produces structured, multi-layered strategies.

19.5 Use Case Category E — Optimization and Variant Selection

Problem Class

Tasks requiring exploration of alternatives and selection of the best option.

Examples

- “Find the optimal configuration.”
- “Compare variants and choose the best.”
- “Generate alternatives and filter them.”

Why A11 is suited

- L5 and L8 generate variants.
- L6 and L9 prune infeasible ones.
- L7 balances conceptual and practical trade-offs.
- L10 validates the final choice.

Architectural Strength

A11 performs structured exploration and deterministic selection.

19.6 Use Case Category F — Complex Reasoning with Recursion

Problem Class

Tasks requiring iterative refinement, recursive exploration, or multi-layered reasoning.

Examples

- “Refine a solution until stable.”
- “Iteratively improve a design.”
- “Search for a stable configuration.”

Why A11 is suited

- Adaptive Layer supports bounded recursion.

- Operators ensure stability at each iteration.
- Rollback prevents runaway recursion.

Architectural Strength

A11 performs safe, bounded, deterministic recursive reasoning.

19.7 Use Case Category G — Alignment-Critical Tasks

Problem Class

Tasks where alignment with intent, ethics, or long-term goals is essential.

Examples

- “Ensure the solution aligns with criteria X.”
- “Evaluate the ethical implications of Y.”
- “Check if the plan is consistent with intent.”

Why A11 is suited

- L2 evaluates alignment.
- L4 integrates alignment into the semantic model.
- L10 validates alignment before realization.

Architectural Strength

A11 guarantees alignment at multiple levels.

19.8 Use Case Category H — Error-Sensitive or Safety-Critical Tasks

Problem Class

Tasks where errors must be detected early and prevented from propagating.

Examples

- “Detect contradictions.”
- “Ensure the plan is safe.”

- "Prevent invalid states."

Why A11 is suited

- Integration nodes detect contradictions.
- Operators correct or rollback.
- Structural invariants prevent unsafe transitions.

Architectural Strength

A11 is inherently safe and self-correcting.

19.9 Summary of Canonical Use Cases

A11 is suited for tasks involving:

- ambiguity,
- uncertainty,
- multi-constraint reasoning,
- multi-step planning,
- optimization,
- recursion,
- alignment,
- safety.

These use cases demonstrate the **universality** of A11 as a reasoning architecture.

20. Limitations and Boundary Conditions (Revised Canonical Architecture)

This section defines the inherent limitations and boundary conditions of the A11 architecture.

These limitations are **structural**, not implementation-specific.

They arise from the canonical geometry, operator model, and deterministic transition system.

Understanding these boundaries is essential for correct use, implementation, and evaluation of A11.

20.1 Structural Limitations

20.1.1 Fixed Geometry (Corrected)

A11 has a fixed **12-stage execution structure**, consisting of:

- **S0** — Input Normalization (initialization stage)
- **L1–L11** — the **11 functional reasoning levels** of the architecture

The architecture contains **exactly 11 functional levels**, and this geometry cannot be modified.

S0 is a mandatory **pre-reasoning stage**, not a reasoning level.

Any attempt to:

- add or remove levels,
- reorder levels,
- bypass integration nodes (L4 or L7),
- merge or split levels,
- alter the branching structure of the Core Layer,
- alter the linear structure of the Adaptive Layer

violates the canonical A11 architecture and invalidates all guarantees.

20.2 Semantic Limitations

20.2.1 No Domain Knowledge by Itself

A11 does **not** contain domain knowledge.

It only structures reasoning **around** whatever knowledge is provided.

20.2.2 No Ontology Creation

A11 cannot invent ontologies.

It can only:

- integrate,
- evaluate,
- structure,
- constrain

existing semantic content.

20.2.3 No Guarantee of Truth

A11 guarantees **coherence**, not **truth**.

If the input is false, A11 will produce a coherent structure around false premises.

20.3 Operational Limitations

20.3.1 No Infinite Exploration

A11 cannot explore unbounded search spaces.

Constraint and recursion bounds enforce finiteness.

20.3.2 No Self-Modification

A11 cannot:

- modify its own operators,
- change recursion limits,
- alter its own invariants.

20.3.3 No Autonomous Optimization

A11 does not optimize itself.

It optimizes **solutions**, not **its own architecture**.

20.4 Computational Limitations

20.4.1 Determinism Over Flexibility

A11 prioritizes:

- determinism,
- reproducibility,
- structural safety

over:

- creativity,
- stochastic exploration,

- probabilistic search.

20.4.2 No Probabilistic Reasoning

A11 does not include:

- probability distributions,
- Bayesian inference,
- stochastic sampling.

It is a **deterministic reasoning architecture**.

20.4.3 No Parallelism Beyond L2–L3

Only the Core Layer supports parallel execution.

Adaptive Layer is strictly sequential.

20.5 Operator Limitations

20.5.1 Operators Cannot Alter Structure

Balance, Constraint, and Rollback modify **state**, not **architecture**.

20.5.2 Balance Cannot Create Meaning

Balance resolves contradictions but cannot:

- invent new semantics,
- generate new facts,
- override Will.

20.5.3 Constraint Cannot Guarantee Optimality

Constraint ensures feasibility, not optimality.

20.5.4 Rollback Cannot Repair Invalid Inputs

Rollback restores stability but cannot fix:

- fundamentally contradictory goals,
- invalid premises,
- impossible tasks.

20.6 Integration Limitations

20.6.1 Integration Requires Coherence

L4 and L7 cannot integrate incoherent states.

If contradictions persist → Rollback.

20.6.2 No Multi-Branch Integration Beyond L4 and L7

A11 supports exactly two integration nodes.

No additional integration layers exist.

20.7 Recursion Limitations

20.7.1 Bounded Recursion

Recursion depth is strictly limited.

A11 cannot perform:

- unbounded search,
- infinite refinement,
- open-ended iteration.

20.7.2 No Cross-Level Recursion

Recursion is allowed only within:

- L5
- L6
- L7
- L8
- L9
- L10

Never in L1–L4 or L11.

20.8 Input Limitations

20.8.1 A11 Requires a Goal

A11 cannot operate without:

- intent,
- direction,
- objective.

20.8.2 A11 Cannot Resolve Undefined Tasks

If the input is:

- contradictory,
- meaningless,
- structurally invalid,

A11 will rollback indefinitely until reframed.

20.8.3 A11 Cannot Infer Missing Context

If essential context is missing, A11 cannot:

- guess,
- assume,
- hallucinate.

It will produce a minimal coherent structure or request reframing.

20.9 Output Limitations

20.9.1 Realization Is Not Execution

A11 produces:

- structured plans,
- validated outputs,
- coherent realizations.

It does **not** execute them.

20.9.2 No Guarantee of Optimality

A11 guarantees:

- coherence,
- feasibility,
- alignment,
- stability.

It does **not** guarantee:

- optimality,
- maximal efficiency,
- global optimum.

20.9.3 No Multi-Objective Optimization

A11 balances constraints but does not perform:

- Pareto optimization,
- weighted multi-objective search.

20.10 Boundary Conditions

20.10.1 A11 Fails Safely

If A11 cannot produce a valid realization:

- it rolls back,
- stabilizes,
- reframes,
- or terminates safely.

20.10.2 A11 Requires Deterministic Environments

A11 is not designed for:

- stochastic systems,
- probabilistic inference,
- random exploration.

20.10.3 A11 Is Not a Learning System

A11 does not:

- update itself,
- learn from experience,
- adapt its architecture.

It is a **reasoning standard**, not a learning algorithm.

20.11 Summary of Limitations and Boundary Conditions

A11 is:

- deterministic,
- structured,
- safe,
- coherent,
- bounded,
- non-probabilistic,
- non-adaptive (architecturally),
- non-self-modifying.

It is **not**:

- a learning system,
- a probabilistic model,
- an optimizer,
- a stochastic search engine,
- a self-evolving architecture.

These limitations ensure that A11 remains:

- predictable,
- verifiable,
- mathematically grounded,
- safe for high-stakes reasoning.

21. Future Extensions and Research Directions (Revised Canonical Architecture)

This section outlines potential extensions, research opportunities, and long-term development paths for A11.

These directions do not modify the canonical architecture but expand its applicability, tooling, and theoretical foundations.

21.1 Extension Category A — Domain-Specific Profiles

A11 can be extended with specialized profiles for:

- autonomous systems
- robotics
- aerospace
- medicine
- legal reasoning
- scientific research
- multi-agent coordination

Each profile defines:

- domain constraints
- operator tuning
- validation rules
- safety boundaries

These profiles remain fully compatible with the canonical architecture.

21.2 Extension Category B — Multi-Agent A11 Systems

Research direction:

- A11 as a coordination protocol between agents
- semantic alignment across agents
- distributed integration nodes

- shared Will and local Will variants
- conflict resolution between agents

This opens the path to **A11-based multi-agent ecosystems**.

21.3 Extension Category C — Hybrid A11 + Learning Systems

A11 is not a learning system, but it can integrate with:

- LLMs
- RL agents
- symbolic systems
- knowledge graphs

Research directions:

- A11 as a reasoning layer on top of learning models
- A11 as a safety filter
- A11 as a planning module
- A11 as a deterministic controller for stochastic systems

21.4 Extension Category D — Formal Verification and Proof Systems

Future work:

- full formal proofs of invariants
- model checking
- theorem-proving integration
- formal semantics for operators
- category-theoretic interpretation of A11

21.5 Extension Category E — Tooling and Infrastructure

Potential tools:

- A11 debugger
- A11 trace visualizer
- A11 compliance validator
- A11 reference implementation
- A11 simulation environment

21.6 Extension Category F — Mathematical Generalizations

Possible research:

- generalized weighting systems
- multi-branch semantic layers
- multi-stage integration
- fractal A11 architectures
- continuous A11 (differentiable operators)

21.7 Summary of Future Extensions

A11 can evolve in:

- domain specialization
- multi-agent systems
- hybrid architectures
- formal verification
- tooling
- mathematical generalization

These extensions expand A11 without altering its canonical core.

22. Implementation Profiles (Revised Canonical Architecture)

This section defines standardized implementation profiles for A11.

Profiles allow developers to choose the appropriate level of complexity and rigor for their use case.

22.1 Profile A — A11-Lite (Minimal Reasoning Engine)

Purpose:

- lightweight reasoning
- embedded systems
- low-resource environments

Characteristics:

- simplified operators
- no recursion
- minimal metadata
- reduced logging

Guarantees preserved:

- determinism
- coherence
- feasibility

22.2 Profile B — A11-Core (Standard Implementation)

Purpose:

- general reasoning
- planning
- decision-making

Characteristics:

- full Core Layer
- full Adaptive Layer
- recursion enabled
- standard operators

- full logging

Guarantees preserved:

- all canonical guarantees

22.3 Profile C — A11-Pro (Advanced Engineering Profile)

Purpose:

- complex planning
- multi-constraint reasoning
- high-uncertainty environments

Characteristics:

- enhanced operators
- extended metadata
- advanced constraint propagation
- operator tuning
- extended validation

Guarantees preserved:

- all canonical guarantees
- extended stability guarantees

22.4 Profile D — A11-Safety (Safety-Critical Systems)

Purpose:

- aerospace
- robotics
- medical systems
- autonomous systems

Characteristics:

- strict rollback rules

- extended invariants
- multi-layer validation
- deterministic operator pipelines
- formal verification hooks

Guarantees preserved:

- all canonical guarantees
- safety-critical invariants

22.5 Profile E — A11-Research (Experimental Profile)

Purpose:

- academic research
- experimental operators
- new integration models

Characteristics:

- pluggable operators
- experimental recursion modes
- extended logging
- relaxed invariants (except structural)

Guarantees preserved:

- structural invariants
- determinism (optional)

22.6 Summary of Implementation Profiles

Profiles allow A11 to be:

- lightweight (Lite)
- standard (Core)
- advanced (Pro)
- safety-critical (Safety)

- experimental (Research)

All profiles remain compatible with the canonical architecture.

23. Compliance Checklist (Revised Canonical Architecture)

This section defines the official compliance checklist for verifying that an implementation conforms to the A11 standard.

An implementation is **A11-compliant** only if it satisfies **all mandatory items**.

23.1 Structural Compliance

- [ ] S0 is implemented as a separate initialization stage
- [ ] L1–L11 exist and are not merged or reordered
- [ ] L2 and L3 execute in parallel
- [ ] L4 integrates S2 and S3
- [ ] L7 integrates S5 and S6
- [ ] No bypass of L4 or L7
- [ ] Adaptive Layer is strictly linear

23.2 Operator Compliance

- [ ] Balance operator implemented
- [ ] Constraint operator implemented
- [ ] Rollback operator implemented
- [ ] Operators modify state, not structure
- [ ] Operators are deterministic

23.3 Transition Compliance

- [ ] All transitions are deterministic
- [ ] Parallel transitions implemented correctly

- [ ] Convergent transitions validated
- [ ] Recursion bounded by MAX_RECURSION_DEPTH

23.4 Validity Compliance

- [ ] Coherence checks implemented
- [ ] Feasibility checks implemented
- [ ] Alignment checks implemented
- [ ] Stability checks implemented
- [ ] Realization validated at L10

23.5 Error Handling Compliance

- [ ] Soft errors trigger Balance or Constraint
- [ ] Hard errors trigger Rollback
- [ ] Rollback resets S1–S4 only
- [ ] No invalid state reaches L11

23.6 Logging and Traceability Compliance

- [ ] All transitions logged
- [ ] Operator activations logged
- [ ] Integration results logged
- [ ] Recursion depth logged
- [ ] Rollback events logged

23.7 Documentation Compliance

- [ ] Implementation profile specified
- [ ] Operator definitions documented
- [ ] Transition rules documented
- [ ] Recursion rules documented

- [ ] Error handling documented

23.8 Summary of Compliance Checklist

A compliant implementation must satisfy:

- structural invariants
- operator correctness
- transition determinism
- validity conditions
- error handling rules
- logging requirements
- documentation requirements

Only then it can be certified as **A11-compliant**.