GOODREASON RESEARCH WHITEPAPER

On Symbolically Computable Systems

Representation, analysis, reasoning and execution under partial knowledge

α π χ ΔΨ β φ τ Ω

Eki Laitila, PhD

Independent systems scientist | GoodReason

Version 0.3 | 19 August 2026

Suomenkielinen saate

Tämä whitepaper määrittelee symbolisesti laskettavan systeemin käsitteen. Lähtökohtana on neljä toisiaan täydentävää vaihetta: symbolinen esittäminen, symbolinen analyysi, symbolinen päättely ja symbolinen suorittaminen. Käsitteen taustalla ovat tekijän vuonna 2008 valmistunut symbolisen analyysin väitöskirja, pitkä Visual Prolog -kehitystyö sekä GoodReasonin α-Ω-koordinaatisto.

Keskeinen väite on, ettei laskettavuus edellytä täydellistä tietoa. Muuttuja voi olla vapaa, kysymys voi jäädä avoimeksi ja päättely voi silti jatkua. Epädeterministinen prosessi voi tuottaa suuren joukon vaihtoehtoisia sidontoja, joista jokainen liitetään perusteluun, lähteeseen ja voimassaoloehtoihin. Tällöin myös ratkaisematon kysymys voidaan säilyttää ensimmäisen luokan tietokohteena.

Whitepaper ei väitä koko todellisuutta algoritmisesti ratkaistavaksi. Symbolinen laskettavuus on rajatun systeemimallin ja sen päättelysääntöjen ominaisuus. Vaisto, kokemus ja tiedostamattomat prosessit voivat ilmentää latentteja kausaalisia säännönmukaisuuksia, vaikka niiden mekanismeja ei vielä pystytä esittämään muodollisesti. Tietämättömyys mekanismista ei merkitse mekanismin tai logiikan puuttumista.

Versio 0.3 erottaa toisistaan laskettavuuden, todistettavuuden ja käytännöllisen ratkaistavuuden. Looginen todistus on aina suhteessa aksioomiin, semantiikkaan ja esitettyyn kontekstiin. Moduulin tarkoitettu input-output-suhde kuuluu sen määriteltyyn logiikkaan, mutta sivuvaikutusten hallinta on sisällytettävä mallin ehtoihin. NP-kompleksisuus puolestaan kuvaa resurssien kasvua ja todistusten löytämisen vaikeutta, ei yksin sitä, onko ongelma periaatteessa laskettavissa.

Ydinkysymys: Voidaanko systeemiä käsitellä symbolisesti niin, että tarkoitukset, suhteet, avoimet muuttujat, vaihtoehdot ja perustelut säilyvät sekä ihmisen että koneen käytettävissä?

Abstract

Systems science has developed powerful concepts for boundary, purpose, hierarchy, feedback, emergence and change. Computational logic has developed equally powerful mechanisms for variables, unification, inference, nondeterminism and execution. Yet these traditions are rarely integrated at the level where a system model can remain epistemically open and still be computationally active. This whitepaper introduces the concept of a symbolically computable system (SCS): an explicitly delimited system model whose relevant entities, relations, purposes, constraints, unresolved questions and evidence are represented by typed symbols,

and whose knowledge state can be transformed through explicit rules without requiring every variable to be bound or every question to be decidable. The proposed capability chain distinguishes symbolic representation, symbolic analysis, symbolic reasoning and symbolic execution. A relational result semantics differentiates justified answers, refutation under declared assumptions, open questions and supported conflicts. The paper connects this proposal to Turing's account of computability, Robinson's resolution principle, Kowalski's separation of logic and control, classical symbolic execution, Visual Prolog flow semantics, multi-agent systems and systems-language research. It separates computability, formal provability and computational tractability; treats proofs as relative to declared semantics, context, side-effect conditions and resource bounds; and identifies protocol stacks as constrained interpreters of typed message symbols. Justification traces provide an operational bridge to human metacognition and machine self-monitoring. GoodReason supplies a systemic coordinate space through eight perspectives (α-Ω) and seven epistemic rings. The result is presented as a research program rather than a completed universal calculus. Its intended contribution is a bridge between systems inquiry, symbolic AI and implementable knowledge structures.

Keywords: symbolic computation; systems science; computational logic; symbolic analysis; logic programming; partial knowledge; four-valued information; abduction; computational complexity; metacognition; Internet protocols; multi-agent systems; GoodReason

Contents

1. Research problem and contribution

Systems inquiry often begins where complete specification ends. The system of interest may have contested boundaries, several purposes, changing actors, incomplete evidence and multiple plausible futures. Conventional computation, by contrast, is commonly presented as a mapping from specified inputs to determined outputs. This contrast can create the impression that formal logic is too narrow for systemic inquiry and that systems thinking must remain primarily descriptive or interpretive.

That conclusion follows only if logic is equated with a closed binary calculus. Logic programming provides a different starting point. A query may contain free variables; unification may bind them; nondeterministic search may return many admissible bindings; backtracking may restore an earlier state; and failure may be interpreted only relative to the declared knowledge and control regime. This allows computation to proceed without pretending that the model is complete.

Core thesis: Logic does not reduce a system to certainty. It specifies how reasoning can proceed when knowledge is partial, alternatives are multiple and some questions must remain open.

The paper contributes six elements: a four-stage symbolic capability chain; an explicit definition of a symbolically computable system; a result space for partial and conflicting knowledge; an interpretation through GoodReason's α-Ω geometry; an implementation direction based on typed logic, JSON-compatible structures, provenance and multi-agent coordination; and an epistemic-feedback architecture connecting metacognitive monitoring to Internet exchange. The contribution is deliberately cautious. It defines a tractable research object and evaluation program rather than claiming a finished theory of all systems.

2. From computable numbers to computable system models

2.1 Computability and symbols

Turing's 1936 paper defined computable numbers through finite symbol-manipulating procedures and established both the power and limits of mechanical computation [1]. The present proposal does not extend Turing computability by

decree. It changes the object being modelled: from a number-producing machine to an explicitly bounded system representation whose states include claims, relations, questions, assumptions and justifications.

Newell and Simon later framed intelligent action in terms of physical symbol systems and search [2]. Their hypothesis made symbols central to artificial intelligence, but the SCS proposal places equal emphasis on epistemic status. A symbol may denote what is known, what is assumed, what is sought, what is disputed or what remains outside the current model.

2.2 Machine-oriented inference

Robinson's resolution principle provided a machine-oriented foundation for automated theorem proving [3]. Kowalski's formulation algorithm = logic + control then separated declarative knowledge from the strategy used to apply it [4]. This distinction is essential for systems science: two agents may share a logical knowledge model while employing different search, priority, risk or stopping policies.

Visual Prolog makes information flow explicit. Predicate flow patterns must match the free/bound status of arguments; unification binds variables while leaving as much as possible open; and a predicate may succeed more than once through backtracking [5]. These are not merely programming details. They provide an operational vocabulary for incomplete systemic knowledge.

2.3 Computability, provability and tractability

Three questions must be kept separate. Computability asks whether a procedure can in principle produce a result. Provability asks whether a proposition follows from declared axioms and inference rules. Tractability asks how time and memory grow with problem size. Cook's analysis of theorem-proving procedures established the foundation of NP-completeness by relating polynomially checkable certificates to problems whose solutions may be difficult to find [16]. The unresolved P versus NP question therefore limits claims about efficient general solution, but NP membership is not a declaration that a problem is uncomputable.

A formal proof of program or system behavior is also relative to a specification. Floyd and Hoare showed how assertions, preconditions, postconditions and program semantics support proofs of correctness [17,18]. A module's intended input-output relation belongs to its declared logic and natural execution flow. If relevant context, shared-state interactions or side effects are omitted, a proof may be valid for the

represented model while failing to establish the desired claim about the wider system. Local and separation-based reasoning make the boundary of affected state explicit rather than assuming that the untouched context is irrelevant [19].

Logical safeguard: A theorem is proved within a formal system and a declared model. The proof establishes the conclusion only under the represented assumptions, semantics, context and side-effect boundary; complexity determines whether finding or checking that proof is feasible at the required scale.

2.4 Symbolic analysis as an intermediate capability

Laitila's dissertation introduced symbolic analysis and an atomistic hybrid model for program comprehension [6]. Its architecture moved from GrammarWare to ModelWare, SimulationWare and KnowledgeWare. Source programs were transformed into symbolic atoms combining object-based encapsulation with logic-language expressiveness; symbolic simulation then supported analysis and knowledge capture. The present paper generalizes that experience from program comprehension toward system comprehension.

Symbolic analysis is therefore not a synonym for reasoning. It is the interpretive and structural work through which representations are inspected, decomposed, connected, compared and prepared for inference. A human, a machine or a mixed human-machine process may perform it.

3. The four-stage symbolic capability chain

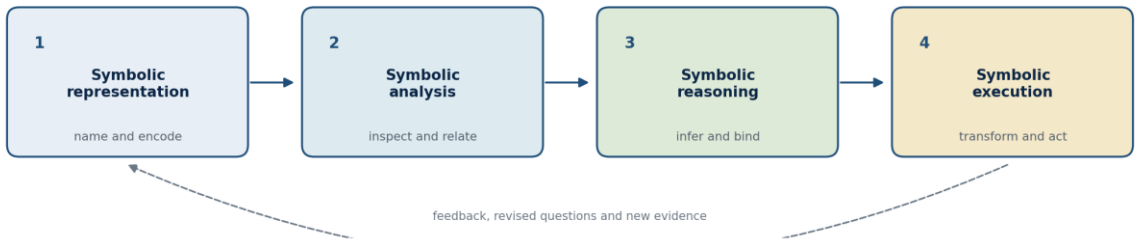

Figure 1. Four symbolic capabilities with a feedback path from execution to revised representation.

Stage	Primary question	Characteristic result
1. Representation	What is named and encoded?	Typed symbols, relations, states and identifiers

Stage	Primary question	Characteristic result
2. Analysis	What structure and meaning can be found?	Decomposition, links, patterns, anomalies and candidate semantics
3. Reasoning	What follows, under which assumptions?	Bindings, alternatives, refutations, open questions and reasons
4. Execution	What symbolic state is transformed or enacted?	Updated knowledge, selected action, simulation trace or agent commitment

3.1 Symbolic representation

Representation names and encodes selected aspects of a system: entities, relationships, purposes, boundaries, contexts, events and states. It is comparable to coding in the broad sense, but it is not yet analysis. A representation is always selective. It should therefore record its SOI boundary, intended use, author or agent, version and evidential status.

A semiotic reading clarifies what representation preserves. In Peircean terms, a sign refers to an object through an interpretant: the developed understanding or further sign through which the relation becomes meaningful [20]. An identifier, diagram, traffic signal or protocol field is therefore not self-explanatory. Its meaning depends on a convention, interpreter and context of use.

3.2 Symbolic analysis

Analysis turns a representation into an inspectable model. It may identify dependencies, control paths, recursive structures, missing links, incompatible types, repeated patterns or semantic roles. In the original program-comprehension work, symbolic execution of an atomistic model exposed program behavior. In a broader SCS, analysis may also compare theories, actor accounts and alternative system boundaries.

Humans and machines may implement this interpretive step differently. A person can recognize a red traffic light as a socially learned sign and relate it to the present traffic situation. A machine can parse a typed field, resolve an identifier or match a symbolic structure against a schema. In both cases analysis establishes relations that reasoning or action can subsequently use.

3.3 Symbolic reasoning

Reasoning applies declared inference rules, constraints and control policies. It can bind free variables, test already bound values, generate alternatives, propagate consequences and retain justification paths. The aim is not necessarily a single answer. A valuable result may be a set of admissible alternatives or a precise description of why the question remains open.

3.4 Symbolic execution

In classical computer science, symbolic execution runs a program with symbolic rather than concrete inputs and accumulates path conditions [7]. This paper preserves that established meaning while defining a broader systemic use. SCS execution transforms a symbolic system state according to an explicit execution semantics. The transformation may simulate a process, activate an inference object, revise a knowledge state, generate an agent commitment or issue an externally executable action. Classical program-path symbolic execution is one special case. In software testing it can explore feasible paths and generate inputs for branches or states that concrete tests have not yet reached; coverage is an evaluation objective, not a synonym for symbolic execution [7].

Terminological safeguard: Representation describes; analysis reveals; reasoning derives; execution changes or enacts a symbolic state. A system may support one stage without supporting all four.

4. Definition of a symbolically computable system

Definition 1: A symbolically computable system is an explicitly delimited system model in which relevant entities, relations, purposes, constraints, unresolved questions and evidence are represented by typed symbols, and in which rule-governed operations can generate and validate justified successor knowledge states without requiring every variable to be bound or every question to be decidable.

The word system refers here to a system model relative to a selected SOI and MOI, not to an ontological assertion that the entire real-world system is algorithmically computable. The phrase symbolically computable identifies a capability of the representation-inference-execution arrangement. Model validity remains empirical, interpretive and purpose-dependent.

The System of Interest (SOI) identifies what is being examined. The Meaning of Interest (MOI) identifies the purpose-dependent interpretation through which a human, community or agent understands that system. MOI is not a static label: it may evolve as symbolic analysis, reasoning, execution and new evidence transform the knowledge state.

$$SCS = (SOI, MOI, \Sigma, T, K, R, C, J, A, E)$$

Element	Meaning
SOI	Declared system of interest, boundary and environment
MOI	Purpose-dependent meaning, viewpoint and evolving interpretation of the SOI
Σ	Symbol vocabulary, identifiers and address space
T	Types, domains and admissible structural forms
K	Partial knowledge state: claims, questions, evidence and statuses
R	Inference and transformation relations
C	Constraints and control policies
J	Justification and provenance structures
A	Human or artificial agents and their local knowledge states
E	Execution semantics connecting symbolic transitions to simulations or actions

4.1 Actor-relative and versioned meaning

In a multi-agent setting, MOI should be represented rather than assumed. Agent a_j may hold an interpretation m_j containing its viewpoint, purpose, present interpretation, provenance and version. New evidence or a new justification may revise that state without silently overwriting another agent's interpretation.

$$m_j = \left(SOI, viewpoint_j, purpose_j, interpretation_j, provenance_j, version_j\right)$$
$$MOI_j(t+1) = revise\left(MOI_j(t), \Delta K_j, J_j, C_j\right)$$

4.2 Explicit and latent logic

Human action is not produced only by conscious deduction. Instinct, experience, dreams, biological regulation and unconscious processes may embody causal and constraint-governed regularities that are not accessible to introspection. This paper calls such organization latent logic. Formal logic is explicit: its symbols, rules and

validity conditions are available for inspection. Latent logic becomes symbolically computable only to the extent that it can be represented, analysed and connected to operational rules.

The same caution applies to randomness. An event described as random occurs within conditions and a probability-generating process, but an individual outcome may remain unpredictable or underdetermined by the available model. Unknown, random and causeless are therefore not treated as synonyms. An SCS records which of these claims is being made and on what grounds.

A statistical language model may assist symbolic analysis by proposing interpretations or candidate relations. Its output enters K as a sourced, provisional contribution; the SCS layer retains explicit types, constraints, provenance, result status and control. A fuller treatment of language-model architecture is outside the scope of this paper.

5. Operational semantics for partial knowledge

5.1 A relational result space

Because reasoning may be nondeterministic, the evaluation of a query is better represented as a relation than as a single-valued function. Let K be a partial knowledge state and q a query. The result relation may yield one or more justified successor states or an explicit epistemic status.

$$Result(K, q) \in answer(theta_i, M_i, K_i next),$$

$$refuted(q, M), open(q, C, M), conflict\big(q, M_p os, M_n eg\big)$$

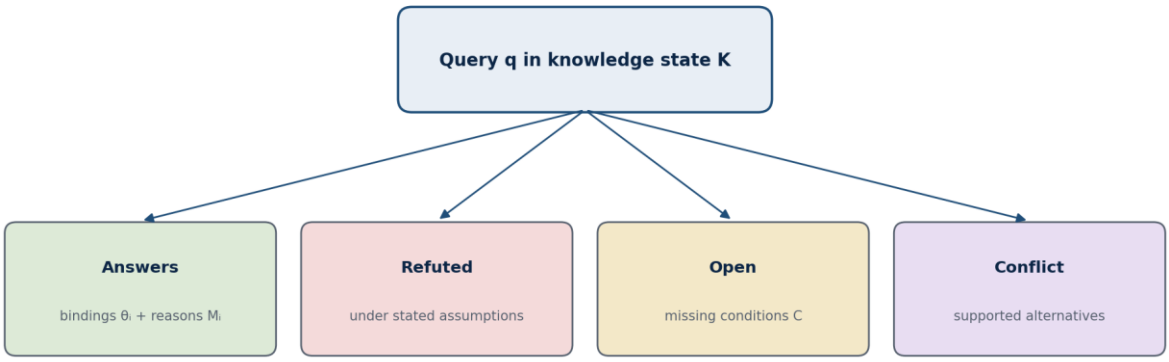

Figure 2. The result of systemic reasoning is not restricted to one answer or binary failure.

An answer contains a substitution θ_i, a justification M_i and a successor knowledge state K'_i. Refutation is valid only under declared assumptions and a specified logical regime. Open means that the question is retained together with missing conditions C and the reason M for its status. Conflict preserves supported positive and negative positions instead of silently overwriting one.

5.2 Neighbouring formalisms: four values and abduction

Belnap's four-valued logic distinguishes information that is supported as true, supported as false, supported both ways or unsupported either way [26]. This is a close neighbour of answer, refuted, conflict and open, but the SCS result objects additionally carry substitutions, conditions, provenance, justifications and successor states. The correspondence is therefore a useful positioning device, not an identity between formalisms.

Abductive logic programming offers another close relation: it searches for hypotheses that, together with a program, explain an observation while satisfying integrity constraints [27]. The missing conditions C in open(q,C,M) may identify candidates for later abduction, but they are not automatically admissible hypotheses. Their status must be declared and constrained.

5.3 Free and bound variables

A variable is free relative to a particular inference state when no value has yet been assigned within that state. It may become bound through unification. On backtracking, the binding can be withdrawn and another admissible value generated. Thus one query may produce a very large set of bound substitutions without implying that the variable possessed all those values at once. Visual Prolog's flow patterns and reference-variable tests make this transition operationally visible [5].

The distinction is epistemically useful. A free variable is not an error or an empty database field. It represents an explicitly open position in a relation. A bound value is not necessarily final truth; it is a value supported in a particular solution state under specified assumptions.

5.4 Circular relations, non-termination and the question as knowledge

$$A = f(Q) \qquad Q = g(A)$$

$$M = \text{justification of the relation and its limits}$$

Circularity may indicate recursion, feedback, mutual definition, an unresolved constraint cycle or an error. An SCS must not decide among these interpretations merely from the shape of the equations. It may seek a justified fixed point, generate several consistent solutions, detect non-termination or preserve the cycle as an open constraint object.

Computability boundary: If SCS execution can simulate arbitrary Turing machines, no universal procedure can decide for every SCS query whether evaluation will terminate or reach a final resolution [1]. A visible cycle alone does not establish this impossibility; the boundary follows from the expressive power of the execution semantics.

This gives a precise interpretation to the proposition that, when the answer is not known, the question itself may be the best available knowledge. The query is retained as open(q,C,M): it has identity, context, missing conditions, provenance and a route for later reactivation. It is not falsely promoted into a resolved answer. When a resource or stopping policy is exhausted, open(q,C,M) records unresolved status and its operational reason; it does not by itself prove that the query is mathematically undecidable.

5.5 Justification as a first-class object

Every consequential binding or state transition should be accompanied by a justification object M. The object may reference a rule, source, observation, agent, timestamp, confidence assessment, counterargument and validity scope. W3C PROV-O offers a standards-based vocabulary for exchanging provenance information across systems [13]. Argumentation frameworks offer a complementary basis for representing attacks and defenses among arguments [9].

An individual justification M is an instance within the broader justification structure J. For a human reasoner, accumulated M objects can support metacognition: knowledge and monitoring of one's own cognitive tasks, strategies and results [24]. In GoodReason this bridge is visible at π4, where reasoning becomes an object of reflection, and χ4, where that reflection becomes communicable information. For a machine, a trace, proof object, diagnostic record or confidence estimate is not evidence of consciousness. It is an operational analogue: the system can inspect records of its own inference and use them to revise control, request evidence or reopen a query [25].

6. GoodReason as a systemic coordinate space

GoodReason supplies an addressable geometry for distributing the elements of an SCS. Its eight perspectives are not inference rules by themselves. They organize what the inference concerns, thereby reducing category mistakes between purpose, theory, evidence, change, structure, design, implementation and evaluation [11].

Coordinate	Role in a symbolically computable system
α Purpose / mind	Declares SOI, MOI, question, intent and success orientation
π Theory / principle	Holds axioms, concepts, models, rules and theoretical commitments
χ Environment	Holds observations, sources, context, external conditions and evidence
ΔΨ Change pressure	Represents open variables, anomalies, tensions, transitions and reframing
β Organization / structure	Defines types, actors, relations, constraints and knowledge architecture
φ Design / solution	Generates and compares candidate bindings, designs and interventions
τ Integration / implementation	Executes selected transformations, simulations, actions and interfaces
Ω Developmental arc / feedback	Validates outcomes, retains provenance, learns and reopens questions

The seven rings add epistemic depth: 1 Vakaus (stability), 2 Epävarmuus (uncertainty), 3 Kompleksisuus (complexity), 4 Herääminen (awakening), 5 Emergenssi (emergence), 6 Murros (transformation) and 7 Pohdinta (reflection). They should not be reduced to a linear difficulty score. A ring indicates the character of awareness and inquiry at a coordinate. For example, φ1 may contain a stable operational solution while φ7 examines what should count as a solution at all.

6.1 One systemic question across seven epistemic rings

A single deliberately open query $-\ q_e con = `What\ is\ an\ economic\ system?'$ —can expose the different contracts of inquiry. A bounded source may return a definition; a programmatic interface may adapt it to declared inputs; different actors may supply incompatible purposes and boundaries; and a multi-agent inquiry may encounter both missing and excessive information. The wording remains fixed while the represented context, permissible operations and result status change.

One question across seven epistemic rings

q_econ = “What is an economic system?”

Ring	Epistemic mode	Computational reference	Inquiry task	Characteristic result
1	Stability	Type 3 / finite-state	bounded definition	answer
2	Uncertainty	Type 2 / pushdown	contextual adaptation	conditioned answer
3	Complexity	Type 1 / linearly bounded	contested interpretation	alternatives
4	**Awakening**	**Type 0 / Turing**	**computational + epistemic watershed**	**answer | refuted | open | conflict**
5	Emergence	—	synthesis with evidence	provenance + revised MOI
6	Transformation	—	multi-agent reconfiguration	distributed viewpoints
7	**Reflection**	**—**	**criteria, governance and ideals**	**sustainable redesign**

Rings 1–4 use the Chomsky hierarchy only as a computational reference. The GoodReason rings are not claimed to be mathematically isomorphic to it.

Figure 3. The same economic-system query is processed under seven different epistemic tasks; ring 4 marks a computational and epistemic watershed.

The references to the Chomsky hierarchy in rings 1–4 concern increasing classes of formal languages and their machine models [28]. They are an analogy for computational reach, not a claim that syntactic language classes, social meanings and GoodReason's rings are mathematically identical. The fourth ring is critical because general symbolic execution encounters possible non-termination, while an epistemically responsible system must also distinguish unresolved and conflicting claims from established answers.

At the outer rings, the inquiry can ask who defines knowledge, whose purposes govern the system and which ideals—such as sustainability—should guide redesign. Fragmentation does not decrease automatically. It may decrease when actor-specific MOIs, assumptions, evidence and conflicts are made inspectable and feedback supports coordinated revision without erasing legitimate difference.

Architectural principle: The α-Ω coordinates organize systemic meaning; logic defines admissible inference; control selects a search strategy; execution changes the symbolic or external state; Ω returns evidence to the next inquiry cycle.

7. Multi-agent reasoning and collective intelligence

A multi-agent SCS does not require all agents to share one complete worldview. Each agent a_j may maintain a local knowledge state K_j, a set of capabilities, priorities and commitments, and an explicit policy for accepting or rejecting information. Shoham's agent-oriented programming demonstrated how beliefs, capabilities, choices and commitments can be treated computationally [8].

$$a_j = \left(MOI_j, K_j, R_j, C_j, goals_j, commitments_j, provenance_j\right)$$

Agents exchange more than conclusions. A useful message contains the query or claim, variable bindings, assumptions, justification, provenance, confidence, requested action and validity interval. This allows another agent to reproduce, challenge, qualify or defer the conclusion. Disagreement becomes structured information rather than a communication failure.

Collective intelligence emerges when several agents can preserve their differences while still coordinating on shared symbols and transition rules. GoodReason's coordinates offer a common address space: one agent may contribute χ-evidence, another π-theory, another φ-design and a fourth Ω-validation. The architecture supports distributed reasoning without assuming that one agent owns the complete model.

8. Implementation architecture

The conceptual model is implementation-neutral, but a practical reference architecture can use typed Visual Prolog objects and predicates for inference, JSON for portable serialization, JSON Schema for structural validation, JMESPath for declarative retrieval and provenance records for traceability. JSON itself is a data-interchange format, not a logic; executable meaning arises from the types, rules, constraints and agents that interpret the data [12].

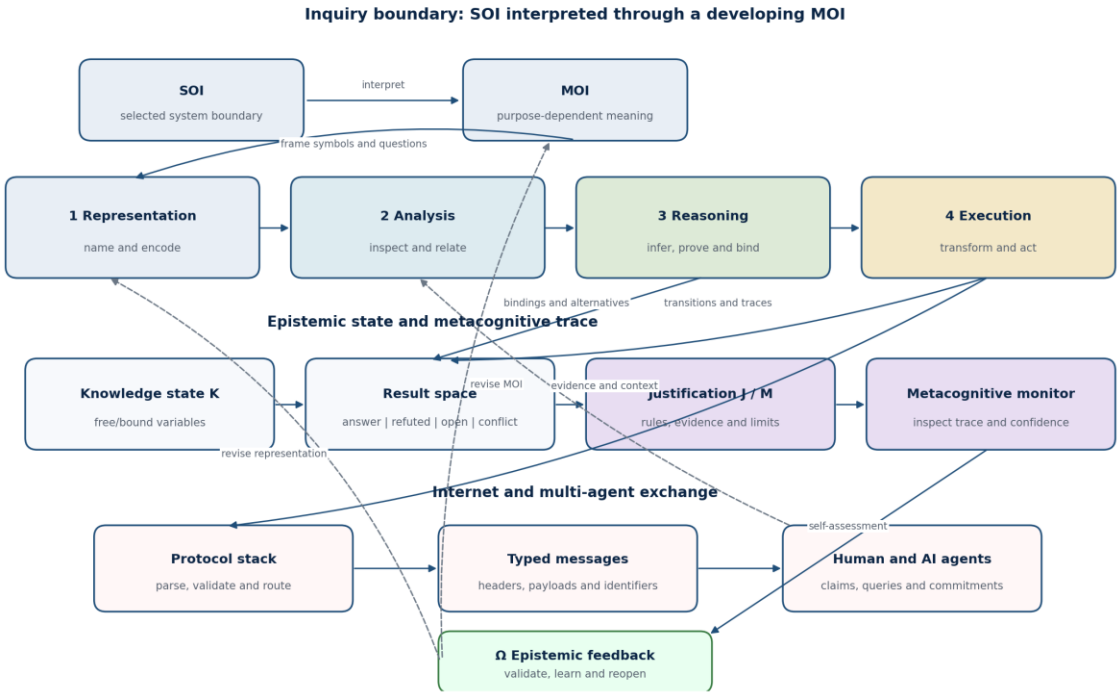

Figure 4. A symbolically computable system connects bounded inquiry, four symbolic capabilities, epistemic outcomes, metacognitive monitoring, Internet exchange and Ω-feedback.

Layer	Responsibility	Candidate mechanism
Symbol registry	Stable identifiers, α-Ω coordinates, ring and version	SuperJSON / JSON objects
Type system	Admissible domains, interfaces and flow patterns	Visual Prolog domains and interfaces
Query layer	Selection and projection of structured content	JMESPath and indexed registries
Inference layer	Unification, rules, alternatives and control	Typed predicates and agent policies
Justification layer	Sources, rules, agents, time and counterarguments	Provenance objects / PROV alignment
Execution layer	Simulation, model transformation or external action	Symbol objects, services and adapters
Protocol layer	Interpretation and stateful handling of message headers	Internet protocols and typed agent messages
Publication layer	Human-readable views and machine-readable exchange	Internet, HTML, diagrams, APIs and repositories

JMESPath can retrieve and transform elements from JSON documents using a specified query language [14]. It is useful for navigation and selection, but it should not be confused with the inference engine. Likewise, JSON Schema validates structure but does not establish the truth of a systemic claim. The separation of representation, analysis, reasoning and execution prevents these tools from being assigned capabilities they do not possess.

8.1 Internet interface and protocol interpretation

The Internet makes the practical importance of symbols visible. A protocol header is a structured set of typed fields whose bit patterns denote such matters as endpoints, sequence positions, flags, lengths and checksums. A protocol stack parses those symbols according to a shared specification, checks conditions, updates connection state and selects a permitted transition. TCP, for example, defines both a header format and required implementation behavior [21]. This is mathematical and algorithmic interpretation of symbols, although it is more constrained than general theorem proving.

The stack can therefore be located across the four capabilities: a received header is represented; its fields and relations are analysed; declared conditions determine admissible transitions; and the transition is executed by accepting, acknowledging, rejecting or routing the message. Higher layers add URLs, JSON objects, prompts and agent messages. They allow an SCS to exchange not only values but also questions, variable bindings, assumptions, provenance and requested actions.

The historical layering is instructive. The domain-name specifications of November 1983 provided a distributed symbolic name space over the Internet [22]. Berners-Lee's Web proposal followed in 1989 and its software was released more broadly in 1991 [23]. The point is not that one invention mechanically caused the other, but that stable symbolic addressing and open protocol interfaces created a platform on which later representational systems could grow.

9. Relationship to System Language

Mobus and Anderson's System Language proposes a lexicon, syntax and semantics for describing system components, flows, boundaries, interfaces and behavior. It aims to support analysis, formal modelling and eventual compilation into executable simulation code [10]. This is a close and valuable neighboring program.

The present proposal should therefore not claim that executable systems modelling is absent from ISSS work. The distinction is narrower. System Language primarily develops a general descriptive and simulation-oriented language of systemness. SCS places logic-programming states and epistemic outcomes at the center: free and bound variables, unification, nondeterministic solutions, open queries, supported conflicts and first-class justifications.

Dimension	System Language	Symbolically computable system
Primary emphasis	General language of system structure and dynamics	Operational logic of partial systemic knowledge
Core objects	Components, flows, boundaries, interfaces, processes	Claims, variables, relations, questions, bindings and reasons
Execution path	Formal model compiled toward simulation	Inference and state transition, possibly connected to simulation or action
Epistemic openness	Model-dependent and interpretive	Explicit statuses for open, refuted and conflicting results
Potential relationship	Supplies systemic lexicon and structural semantics	Supplies reasoning and execution semantics over such representations

Possible synthesis: System Language can describe what a system is composed of and how it behaves; an SCS layer can state what is known, what is asked, how alternatives are derived and why a transition is accepted.

10. Limits, validation and research program

10.1 Conceptual limits

- The term symbolically computable does not imply that the real system is fully computable, predictable or controllable.
- A symbol assignment is a semantic convention. For example, φ is an address for design and solution; the golden ratio does not prove a design correct.
- An unresolved query is knowledge about the inquiry state, not evidence that every possible answer is equally plausible.
- Failure under a closed-world policy must not be silently generalized into falsity under an open-world interpretation.
- Contradictory JSON records remain data until an explicit argumentation or inconsistency policy interprets them.

- Latent logic is a philosophical and causal hypothesis; only represented and operationalized parts become symbolically executable.
- Decidability does not imply practical tractability. NP-complete search, path explosion and unbounded interaction can make a formally specified task infeasible at the required scale.
- A proof is only as comprehensive as its specification: omitted context and side effects remain outside the established conclusion.

10.2 Validation criteria

A prototype should be evaluated through observable capabilities rather than conceptual appeal. A candidate SCS implementation should demonstrate at least the following:

- Type-correct representation of a bounded SOI and its α-Ω coordinates.
- Queries containing free variables and repeated production of alternative bindings.
- Separation of answer, refutation, open status and supported conflict.
- Traceable justification from each result to rules, evidence and responsible agents.
- Controlled handling of recursive or circular constraints, including non-termination detection.
- Declared handling of side effects at module, service and agent boundaries; intended input-output relations remain part of the module logic.
- Round-trip publication between machine-readable structures and intelligible human views.
- Reproducible multi-agent exchange in which another agent can inspect or challenge a result.
- A complexity profile recording search growth, proof or certificate checking cost, stopping policy and resource budget.
- Metacognitive inspection in which a trace or justification can trigger revision, an evidence request or an explicit open status.
- Protocol round-trip tests showing that typed messages preserve identifiers, bindings, provenance and requested actions across an Internet interface.

10.3 Proposed research sequence

Systems methods provide preparatory architectures for the phases below. The research object can be examined at micro and macro scales and through black-box, gray-box or transparent internal views. These are alternative boundaries of observation rather than a compulsory linear sequence. The program advances by synthesizing viewpoints, making their assumptions explicit and testing whether the resulting model supports traceable reasoning and action.

Project materials already report preliminary implementation evidence for several phases: a 56-address GoodReason structure, recursive agent profiles, a

GoodReasonLang representation layer, and a Prolog-based diagnostic engine spanning fourteen named frameworks. These artifacts can reduce the distance to demonstration, but they are not yet independent validation; the next step is to publish reproducible datasets, queries, expected results and provenance.

Phase	Research object	Demonstration
A. Formal core	Result relation, free/bound states, proof conditions and complexity profile	Small typed reference model
B. GoodReason mapping	α-Ω coordinates and seven rings	Publish and test the reported 56-address prototype
C. Symbolic analysis	Human-machine inspection and model enrichment	Trace from raw source to structured findings
D. Reasoning	Alternative bindings, open questions and conflicts	Package the reported diagnostic engine as a reproducible query suite
E. Execution	State transitions and action adapters	Simulation or bounded real-world workflow
F. Multi-agent case	Distributed evidence, theory, design, protocol exchange and validation	Internet-connected systemic-change demonstrator
G. Comparative study	SCS, System Language and other formalisms	ISSS workshop and peer-reviewed evaluation

11. Conclusion

A symbolically computable system is neither a diagram alone nor a claim that reality is a computer. It is a bounded, typed and executable knowledge model in which symbols can be represented, analysed, reasoned over and used to transform states. Its distinctive feature is the preservation of epistemic openness: a free variable can remain free, several bindings can coexist, a contradiction can remain visible and a question can be stored as the best available knowledge when its missing conditions are explicit.

This proposal reconnects systems science with a strand of symbolic AI that began with formal computability, machine-oriented inference and logic programming. GoodReason adds a geometric and systemic address space that extends from α-purpose to Ω-feedback and from operational stability to philosophical reflection. The intended outcome is not a closed universal method, but an open architecture in which human interpretation, formal reasoning and agent-based execution can

correct and strengthen one another. Internet protocols extend that architecture from local computation to distributed symbolic interaction, while justification traces allow both people and machines to examine the conditions and limits of earlier reasoning.

Final proposition: A system becomes symbolically computable when its representation preserves enough meaning for analysis, enough structure for reasoning, enough operational semantics for execution and enough provenance for the results to remain open to criticism.

References

[1] Turing, A. M. (1936/1937). On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, 42, 230-265. https://doi.org/10.1112/plms/s2-42.1.230

[2] Newell, A., & Simon, H. A. (1976). Computer Science as Empirical Inquiry: Symbols and Search. Communications of the ACM, 19(3), 113-126. https://doi.org/10.1145/360018.360022

[3] Robinson, J. A. (1965). A Machine-Oriented Logic Based on the Resolution Principle. Journal of the ACM, 12(1), 23-41. https://doi.org/10.1145/321250.321253

[4] Kowalski, R. (1979). Algorithm = Logic + Control. Communications of the ACM, 22(7), 424-436. https://doi.org/10.1145/359131.359136

[5] Prolog Development Center. Visual Prolog Language Reference: terms, flow patterns, unification and free/bound variables. https://wiki.visual-prolog.com/index.php?title=Language_Reference_Book

[6] Laitila, E. (2008). Symbolic Analysis and Atomistic Model as a Basis for a Program Comprehension Methodology. Jyväskylä Studies in Computing 90, University of Jyväskylä. ISBN 978-951-39-3252-7.

[7] King, J. C. (1976). Symbolic Execution and Program Testing. Communications of the ACM, 19(7), 385-394. https://doi.org/10.1145/360248.360252

[8] Shoham, Y. (1993). Agent-Oriented Programming. Artificial Intelligence, 60(1), 51-92.

[9] Dung, P. M. (1995). On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. Artificial Intelligence, 77(2), 321-357.

[10] Mobus, G., & Anderson, K. (2016). System Language: Understanding Systems. Proceedings of the 60th Annual Meeting of the ISSS. https://journals.isss.org/index.php/proceedings60th/article/view/2917

[11] Laitila, E. (2026). Axiomatic Systems Science - Geometry of Thinking. Journal of the International Society for the Systems Sciences. https://journals.isss.org/index.php/jisss/article/view/4627

[12] Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259 / STD 90. https://www.rfc-editor.org/rfc/rfc8259

[13] W3C. (2013). PROV-O: The PROV Ontology. W3C Recommendation. https://www.w3.org/TR/prov-o/

[14] JMESPath. Specification and compliance suite for the JSON query language. https://jmespath.org/specification.html

[15] JSON Schema. Draft 2020-12 specification. https://json-schema.org/draft/2020-12

[16] Cook, S. A. (1971). The Complexity of Theorem-Proving Procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151-158. https://doi.org/10.1145/800157.805047

[17] Floyd, R. W. (1967). Assigning Meanings to Programs. Proceedings of Symposia in Applied Mathematics, 19, 19-32.

[18] Hoare, C. A. R. (1969). An Axiomatic Basis for Computer Programming. Communications of the ACM, 12(10), 576-580. https://doi.org/10.1145/363235.363259

[19] O'Hearn, P. W., Reynolds, J. C., & Yang, H. (2001). Local Reasoning about Programs that Alter Data Structures. Computer Science Logic, 1-19. https://doi.org/10.1007/3-540-44802-0_1

[20] Atkin, A. (2022). Peirce's Theory of Signs. Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/peirce-semiotics/

[21] Eddy, W. (Ed.). (2022). Transmission Control Protocol (TCP). RFC 9293 / STD 7. https://www.rfc-editor.org/rfc/rfc9293

[22] Mockapetris, P. (1983). Domain Names: Concepts and Facilities. RFC 882. https://www.rfc-editor.org/rfc/rfc882

[23] CERN. The Birth of the Web and A Short History of the Web. https://home.cern/science/computing/the-birth-of-the-web/

[24] Flavell, J. H. (1979). Metacognition and Cognitive Monitoring: A New Area of Cognitive-Developmental Inquiry. American Psychologist, 34(10), 906-911. https://doi.org/10.1037/0003-066X.34.10.906

[25] Cox, M. T., Mohammad, Z., Kondrakunta, S., Gogineni, V. R., Dannenhauer, D., & Larue, O. (2022). Computational Metacognition. arXiv:2201.12885. https://doi.org/10.48550/arXiv.2201.12885

[26] Belnap, N. D. (1977). A Useful Four-Valued Logic. In J. M. Dunn & G. Epstein (Eds.), Modern Uses of Multiple-Valued Logic, 8-37. Springer. https://doi.org/10.1007/978-94-010-1161-7_2

[27] Kakas, A. C., Kowalski, R. A., & Toni, F. (1992). Abductive Logic Programming. Journal of Logic and Computation, 2(6), 719-770. https://doi.org/10.1093/logcom/2.6.719

[28] Chomsky, N. (1959). On Certain Formal Properties of Grammars. Information and Control, 2(2), 137-167. https://doi.org/10.1016/S0019-9958(59)90362-6

Status of this document

Version 0.3 is a conceptual research draft intended for discussion and refinement. It preserves the architecture and main argument of version 0.2 while adding a precise Turing-limit statement; relations to Belnap's four-valued logic and abductive logic programming; actor-relative, versioned MOI; explicit side-effect conditions; preliminary implementation evidence; and one running economic-system query across seven epistemic rings. A subsequent version should add a machine-readable reference schema, worked cases, explicit inference contracts and empirical comparison with systems-language and knowledge-representation approaches.