A Specification's Realm: Characterizing the Knowledge Required for Executing a Given Algorithm Specification

Assaf Marron[a] and David Harel[b]
Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, 76100, Israel

Keywords: Algorithm, Specification, Knowledge, Algorithm-Specification Realm.

Abstract: An algorithm specification in natural language or pseudocode is expected to be clear and explicit enough to enable mechanical execution. In this position paper we contribute an initial characterization of the knowledge that an executing agent, human or machine, should possess in order to be able to carry out the instructions of a given algorithm specification as a stand-alone entity, independent of any system implementation. We argue that, for that algorithm specification, such prerequisite knowledge, whether unique or shared with other specifications, can be summarized in a document of practical size. We term this document the *realm* of the algorithm specification. The generation of such a realm is itself a systematic analytical process, significant parts of which can be automated with the help of large language models and the reuse of existing documents. The algorithm-specification's realm would consist of specification language syntax and semantics, domain knowledge restricted to the referenced entities, inter-entity relationships, relevant underlying cause-and-effect rules, and detailed instructions and means for carrying out certain operations. Such characterization of the realm can contribute to methodological implementation of the algorithm specification in diverse systems, to formalization of the algorithm for mechanical verification, and to incorporating the algorithm and its environment in comprehensive system models. The paper also touches upon the question of assessing execution faithfulness, which is distinct from correctness: in the absence of a reference interpretation of natural language or pseudocode specification with a given vocabulary, how can we determine if an observed agent's execution indeed complies with the input specification.

1 INTRODUCTION

In philosophy of computer science, definitions of *an algorithm* commonly require that instructions be explicit and precise, enabling a human or machine agent to execute the algorithm specification without applying ingenuity. For algorithms specified in Turing Machines, Abstract State Machines (Gurevich, 2000), recursors (Moschovakis, 1998), or as the source code for a working computer program, such explicitness and precision are built into the definition of the formalism. The knowledge and skills of the agent are well defined, covering basically the syntax and semantics of the formalism and a respective execution environment, with memory, processing facilities, etc.

However, for algorithm specifications in natural language (NL) or in pseudocode, one senses that what makes them clear to execution agents would vary significantly with the domain, specification style, or choice of agent. Furthermore, describing what is needed for such clarity in any particular case remains tacit. Here, we contribute the following: (i) introducing the concept termed *the realm of an algorithm specification*, which refers to the prerequisite knowledge and skills needed by an executing agent, beyond the algorithm specification artifact itself, to enable execution that is "mechanical", i.e., not requiring ingenuity; (ii) an initial characterization of the kinds of information such realms contain; and (iii) a process for documenting, for any algorithm specification, its corresponding realm. Throughout we use the term algorithm specification to refer to an artifact embodying an algorithm; this artifact and term are distinct from the terms *specification* or *specifications* when used in system and software engineering for documents articulating requirements or properties of the system, its behavior, or its outputs. Fig. 1 illustrates the relations of the algorithm, its specification, the specification's realm, and the executing agent.

In Sec. 3, we cover several categories of realm

[a] https://orcid.org/0000-0001-5904-5105
[b] https://orcid.org/0000-0001-7240-3931

Marron, A. and Harel, D.
A Specification's Realm: Characterizing the Knowledge Required for Executing a Given Algorithm Specification.
DOI: 10.5220/0014306200004058

In *Proceedings of the 14th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2026)*, pages 265-273
ISBN: 978-989-758-798-6; ISSN: 2184-4348

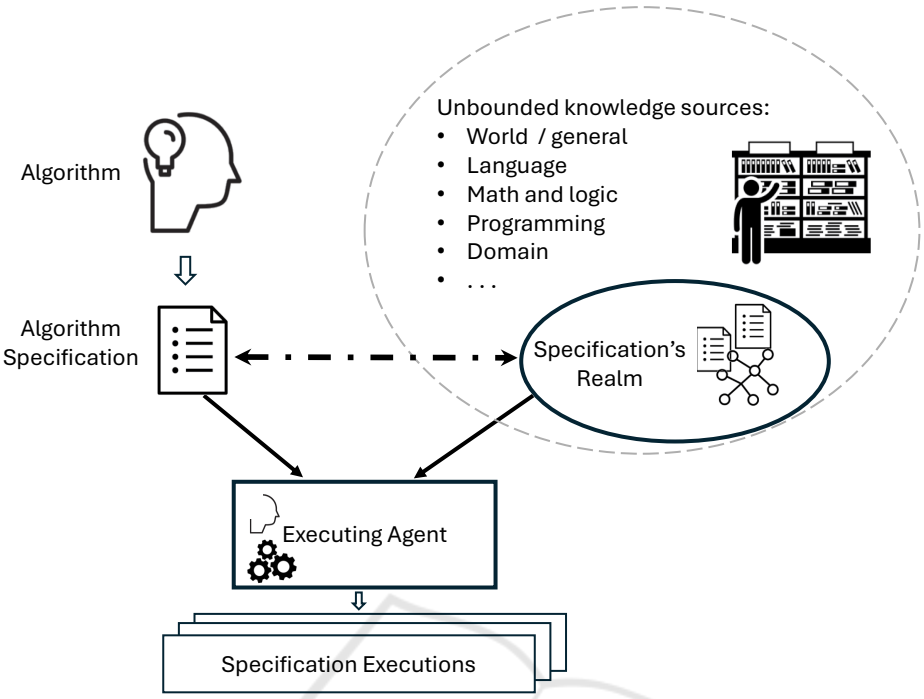

Figure 1: The concept of an algorithm specification's **realm**. A mental conception of **an algorithm** (top left) is concretized in **a specification document**. The specification's **realm**, constructed specially for that specification as described in Sec. 4, contains the knowledge required for executing that specification. The realm is only a **finite subset of the unbounded information** available for the various related disciplines. The executing agent — human or machine — receives both the algorithm specification and the realm constructed for it, and can then **execute the specification repeatedly** on diverse inputs.

contents: language syntax and semantics of the algorithm specification; resolution of entity references; relevant inter-entity relationships and cause-and-effects rules; and tools for carrying out certain specified actions. As the realm concept evolves, this categorization may be refined or extended.

Most importantly, for a given algorithm specification, its realm is finite and bounded: it contains only a small subset of the unbounded information about natural language, mathematics, programming, and diverse engineering and real-world domains that a single hypothetical general-purpose algorithm-specification-execution agent, human or machine, would have to possess.

Documenting an algorithm-specification's finite realm can help guide its implementation in diverse systems and simplify the analysis of its complexity, behavior patterns, special cases, etc. Furthermore, the realm can enrich models of the systems where the algorithm is implemented. The realm can also assist in formalizing the specification and its assumptions and goals toward formal verification of correctness. Finally, unions of a finite number of algorithm specification realms in a given domain, might serve in constructing domain-specific agents for executing not-previously-seen algorithm specifications.

Throughout this paper, the algorithm at hand is assumed to be documented in a physical artifact S — the algorithm's specification — written in NL or pseudocode; we may refer to "the algorithm", expecting the distinction between the concrete specifications and the abstract concept of a named algorithm, like Quicksort or Binary Search, to be understood from the context.

To set the stage for our discussion, Figure 2 lists several diverse algorithm specifications, and a few examples of the kinds of information their realms would contain. Due to space considerations and to retain the flow, we rely on readers' prior familiarity with most such types of specifications and omit images of concrete specification examples. Beyond classic compact sequential pseudocode specifications and an entire written article describing an algorithm for solving a mathematical or technical problem, like (Bie et al., 2023), we highlight two additional styles: One is rule-based or scenario-based specification. The executing agent has to apply a collection of self-standing rules subject to pre-determined semantics. For example, in the right hand rule for maze solving, in order to exit the maze, a human agent must, constantly abide

by two rules (a) keep a hand on the right wall, and (b) proceed forward. In the approach described in (Harel and Marron, 2018; Harel et al., 2021) an intricate algorithm, like Quicksort, which is often described in a procedural step-by-step manner, possibly with recursive calls, is specified as a collection of self-standing scenarios, each of the form *"Always, when* `<condition>` *holds, do* `<action>`*"*. The choice of how the various scenarios are orchestrated is a foundational element of the general approach, and may vary across diverse scenario-based formalisms.

In yet another style, specifications detail artisans' instructions for activities in their trade, say, the making of violins; such instructions may have been passed through generations in various forms, and may rely on extensive tacit domain-specific knowledge and practices and on professional jargon (Denis, 2006).

In discussing what an executing agent may or may not know in advance, we rely on the assumption that an executing agent is able to process new well-specified information, and learn new well-defined skills, and apply these in the mechanical execution of an algorithm specification.

This assumption is needed, as even the most basic skills may be domain specific, or even algorithm specific. Considering the examples in Figure 2, the maze algorithm does not require arithmetic, the drawing of a violin curve does not require understanding arrays or computer memory, and binary search does not require understanding the geometry of a two-dimensional plane with paths and obstacles. We also defer answering whether certain skills or knowledge are common to all agents, or must exist in all algorithm realms, or represent another category of knowledge altogether. Note that the assumed meta-ability of an agent to acquire foundational skills ad-hoc, or interpret additional information, appears quite advanced. It may be accomplished by hand-crafted programming of the agent by humans or by incorporating artificial intelligence and machine learning tools like large language models in the agent. A child or a university student may or may not possess this meta-ability naturally, and may or may not be able to acquire it.

Indeed, even the ability to follow instructions is not a trivial one. Sequential step-by-step processing may appear intuitive, but following nested iterations (loops) or go-to instructions is harder. So is concurrent application of rules, with the agent having to understand issues like order of rule evaluation, rule side effects, action selection, concurrent execution of physical operations, and the semantics of continuous versus discrete and instantaneous operations.

Hence, while the realm concept aims at enabling mechanical execution, we refrain from asserting whether an execution relies on ingenuity or not.

The paper is structured as follows. In Sec. 2 we briefly review the known requirement that an algorithm specification be executable without applying ingenuity. In Sec. 3 we introduce the kinds of elements that comprise the realm. In Sec. 4 we outline a practical constructive process whose input is an algorithm specification and its output is a document containing the realm for that algorithm. In Sec. 5 we discuss related work. In Sec. 6 we discuss the significance of recognizing the concept of an algorithm's realm and several other implications of the current work.

2 THE DEMAND FOR CLARITY

The definition of the general concept of an algorithm draws much discussion, see, e.g., (Harel, 1987a), (Hill, 2016),(Papayannopoulos, 2023),(Primiero, 2020, Ch.6). In this paper we sidestep most of the subtleties associated with the original definitions, their respective contexts and subsequent discussions. Instead, we focus on characterizing the knowledge required of the agent executing an algorithm specification. Still, as a starting point for the current discussion, we present below selected definitions as cited or paraphrased in (Hill, 2016). In each paragraph, we highlight with italics words and phrases that imply the requirement that the instructions be clear to the agent, sufficiently to enable execution.

"**Markov and Nagorny (1988)**: An algorithm is a prescription *uniquely determining* the course of certain constructive processes."

"**Kleene (1967):** If (after the procedure has been described) we select any question of the class, the procedure will then *tell us how* to perform successive steps, after a finite number of which we will have the answer to the questions we selected. In performing the steps, *we have only to follow the instructions mechanically, like robots; no insight or ingenuity or invention is required of us*. After any step, if we don't have the answer yet, the instructions together with the existing situation will tell us what to do next. The instructions will enable us to recognize when the steps come to an end, and to read off from the resulting situation the answer to the question, "yes" or "no". In particular, [.....], the description of the procedure, by a list of rules or instructions, must be finite."

"**Minsky (1967):** An *effective* procedure is a set of *rules which tell us*, from moment to moment, how to behave."

"**Knuth (1997):** 'An algorithm is a finite set of rules

that gives a sequence of operations for solving a specific type of problem,' with five essential features: finiteness, *definiteness*, input, output, effectiveness."

"**Rapaport (2012)** (synthesized definition): An algorithm (for executor E to accomplish goal G) is: A procedure (or method)—i.e., a finite set (or sequence) of statements (or rules, or instructions)—such that (1) each statement is composed of a finite number of symbols (or marks) from a finite alphabet and is *unambiguous* for E—i.e.,: (a) E *knows* how to do it; (b) E *can* do it; (c) it can be done in a finite amount of time; and (d) after doing it, E knows what to do next; (3) the procedure takes a finite amount of time, i.e., halts,; (4) it ends with G accomplished."

"**Hill (2016): Definition 1.** An algorithm is a finite, abstract, *effective*, compound control structure, *imperatively* given."

"**Hill (2016): Definition 2.** An algorithm is a finite, abstract, *effective*, compound control structure, *imperatively* given, accomplishing a given purpose under given provisions."

In these definitions, the demand for clarity of an algorithm-specification's instructions is manifest. However, the characterization of what makes a specification clear, or what an executing agent must know in advance in order to be able to follow certain instructions, remains tacit.

3 CONTENTS OF A SPECIFICATION'S REALM

We assume that once an executing agent G is presented with an algorithm specification S, it can process its realm R acquiring the additional information. Then, in executing S, G can associate the various artifacts of S and R, like objects and actions, with its internal execution templates and processing capabilities. In the coming subsections, we offer an initial classification of the elements in algorithm specifications realms and delineate the realm boundaries.

3.1 Language Syntax and Semantics

The realm contains explanations of all the language constructs that appear in the algorithm specification. They may be general and common in some language domains, or unique to the specification at hand. These include the syntax of individual statements, flow or step-by-step instructions, looping and iteration logic (including termination and break semantics), code blocks notation (begin/end, curly brackets/braces, indentation, etc.), composition of standalone rules, reliance on details that are explained elsewhere (as in function calls, usage of structures like *"where* `<this>` *is* `<that>`*"*, pointers to explanatory footnotes, etc.), and more.

The realm may also contain a preamble explaining the syntax and semantics of other realm contents.

We defer to future assessment whether interpreting an algorithm provided in a multi-page scientific paper is a basic agent skill or should be part of a realm. This includes relying on a host of software libraries and mathematical notations, and the ability to assemble a sequential process from a collection of sub-processes referencing each other in diverse ways.

3.2 Foundational Model Entities

An algorithm specification contains or implies a model of the context in which it operates; where applicable, this context may be thought of as a system and its environment. Such a model includes objects (as understood in object-oriented modeling), properties, property values, object methods, conditions, states, and events. The model also includes actions that are not directly associated with objects, like meta-operations regarding the execution environment, meta-computation like choosing a random value, entity creation and deletion, establishing relations among existing entities, etc.

A realm R of an algorithm specification S should: (i) identify which terms in S serve as objects, properties, methods, events, other actions, etc.; (ii) unify synonymous references to the same entity; (iii) disambiguate cases where the same terminology refers to distinct entities; and (iv) elucidate any implicit or absent entity references making them explicit.

3.3 Entity Relationships

The realm should use domain knowledge to fill in entity relationships, like *contains*, *part-of*, or *kind-of*, that are relevant to execution. For example: that a list A to be sorted *contains* the entities which are being compared and reordered; which arc in a violin-body circumference is *connected to* which other two arcs; or, which set of traffic sensors, like for location and velocity is associated with which monitored vehicle.

3.4 Domain Causality Rules

A deeper layer of information in the algorithm-specification realm is articulation of cause-and-effect rules that hold in the execution environment. Here are a few examples: for the binary search algorithm in

(Hill, 2016) the realm should add the implicit fact that the variable *N* containing the search space length is updated automatically whenever the variables `Lower` or `Upper` marking the search space boundaries are updated; when carrying out bubble sort in a computer memory, storing a value in a memory cell erases irretrievably its previous contents, hence swap operations must be done with temporary holding spaces; and, drawing circles and lines create intersection points that are needed in subsequent steps.

When the algorithm specification is part of a larger system engineering model, some causality rules may be derived from descriptions of dynamic system behavior; see Sec. 5 for more details.

3.5 Instruction Explanations and Tools

The algorithm specification may include instructions that appear explicit and clear to their author, but that the agent at hand cannot carry out. The realm will thus elucidate opaque instructions, provide necessary instrumentation, or explain how to obtain information that is required but not provided. Examples:

1. the realm may explain to a smart and dexterous robot agent the use of a drawing compass: placement of the sharp point, measuring the radius, etc.
2. for a maze-solving software agent, the realm may include a software function for converting a high-resolution maze map image into a smaller matrix capturing the abstracted, schematic maze structure; it may also explain that the "hand-on-wall" is a metaphor for a minimal numerical distance.
3. the realm may show a human or machine agent, how, for the first time, to make a choice, as in the instruction *"pick an arbitrary radius and draw a circle around point C."* (as appears in some algorithms for bisecting an angle). In such a case, the realm will also introduce value range constraints.
4. The realm may explain to an agent drawing a violin body contour that the lengths of the bouts (=arcs) are not specified; instead they are implied by the intersection points of adjacent circles.
5. in (Bie et al., 2023), the algorithm specification for algorithm YOLOv5n-L is a list of modifications to the published algorithm YOLOv5n. An agent executing YOLOv5n-L must have access to the behavior of YOLOv5n.

3.6 Realm Scope and Boundaries

The realm is provided in order to enable mechanical execution. Including additional information about the domain or the problem, the algorithm's specific solution, and perhaps about alternative solutions, is not needed and should preferably be excluded. For example, if the algorithm specification is for carrying out binary search in a list, the realm should not include a function or library that replaces the algorithm-specification's steps with one or more alternative function calls. The algorithm specification defines a particular choice of steps, in a particular order or compositional semantics. Furthermore, the specification may be focused on contrasting this choice with other algorithm specifications for solving the same problem, rather than on explaining for the first time how the problem should be solved. This choice should remain explicit, and the realm should not hide or obscure it.

4 CONSTRUCTING A SPECIFICATION'S REALM

Below we provide an outline of one possible manual and computer assisted structured process for creating a document containing a realm *R* for a given algorithm specification *S*. Our goal here is to emphasize the existence of such processes, and the finiteness and manageability of their output artifacts. Clearly, with present and future large-language-model tools, many of these steps can be readily automated. The description of some of these steps can be aligned with steps in classic object modeling and requirements engineering guidelines, as in (Booch, 2005; Deeptimahanti and Babar, 2009), although the overall process is very different, as discussed in Sec. 5.

1. If *S* is in pseudocode aligned with a particular programming language - add to *R* a language reference manual describing the syntax and semantics of that language. Add paragraphs as needed to highlight differences. You can suffice with those aspects of the language and pseudocode that are used in *S*.
2. Parse *S*; extract nouns, verbs and adjectives, creating a glossary. If *S* is in NL, you may use a large language model (LLM) for this.
3. After each step below, examine *R*'s vocabulary subject to domain knowledge: equate synonyms, add qualifiers to disambiguate similar distinct references, and concretize implicit references.
4. Using domain knowledge, possibly with the help of an LLM, transform the above glossary into an initial model *M* class diagram of objects and their properties and methods, as well as a list of events.

Algorithm specification	Specification format / medium	Examples of Realm Contents
- Bubble sort (Wikipedia, 2025) **- Binary search** (Hill, 2016)	- Pseudocode	**Concept:** List **Semantics:** Indexing is 0-based or 1-based **Object binding:** Input variable names. **Causality:** Changing search space boundaries changes its size.
- Bisecting an angle (Wikipedia, 2025) **- Drawing a violin body** (Denis, 2006)	- Natural language - Drawings	**Physical operation:** Using a drawing compass. **Meta computation:** Choosing an arbitrary value **Domain knowledge:** - Arc lengths are implicit: dictated by mutual intersections
- Maze solving (Wikipedia, 2025) **- Scenario-Based Quicksort** (Harel and Marron, 2021)	- NL rules	**Rule activation semantics:** concurrency; prioritization. **Implicit rules:** "Always advance forward" **Domain abstractions**: - Translating "hand on wall" to diverse physical or virtual agents. **Causalities:** - All actions that change the status of an entity being processed are listed; otherwise, status remains unchanged.
- Vehicle detection (Bie et al. 2023)	- NL - Math equations - In a scientific paper	**Tools:** Libraries for machine learning & math functions. **Object binding:** Tying sensor data to specification variables **Semantics:** Reading scientific papers; interpreting equations

Figure 2: Examples of algorithm specifications and their realms. For each algorithm specification, identified with its source in the first column, the second column describes the style and format of the specification. The third column contains informal statements of a few examples of additional facts and skills that are not part of the algorithm specification, and must be established in the specification's realm in order to enable an agent to execute it.

5. Associate allowed property values with the various properties.
6. Identify entity references that depend on relationships between entities. Use domain knowledge to complete M with these relevant relationships.
7. Determine which invocation of a method call, property change or other action triggers each event that appears in the emerging model M . Make sure that such triggers are indeed specified directly in S or are specified as a result of causality rules in R. When no such cause-and-effect rule is found, use domain knowledge to fill that gap. For each new causality rule, add its triggering conditions and events to M and repeat the above steps to fill any gaps introduced by the new entities.
8. Examine the executing agent's capabilities. For each method call, property change, and, most importantly, action of some other kind, that appears in M, check if the agent is able to carry it out directly. If not, provide the necessary information, like adding a library and an appropriate API, or a procedure for accomplishing the same task, consisting of substeps that the agent can carry out.
9. Finally, for each entity in S and R, ask whether the agent knows what it is or how to handle it. When the answer is no, add an explanatory layer of entities, relationships, causalities, and tools. Deciding when to stop constructing the realm can be understood in the context of vertical and horizontal delineation, termed in (Harel, 1987b) simply as depth and modularity. These are, respectively, the limits forced on the top-down depth in refining hierarchical abstractions, and the scope or contents of a system or a module relative to everything else. In classic modeling, these traits are dictated by an outside-in view of a mix of the known requirements and implementation resources and constraints. By contrast, realm construction induces an inside-out view: the vertical

depth and the horizontal scope of the specification's realm are dictated by the content of *S*, and the capabilities of the chosen execution agent *G*.

Note that LLMs may be used in the construction of *R* but are not used in the execution of *S* given *R*.

5 RELATED WORK

The concept of an algorithm-specification's realm differs from the related concept of Problem Frames (Jackson, 2005) as follows. Realms embody an inside-out view: starting with the specification, missing elements are filled in from multiple dimensions: the entities involved and their relationships as needed for execution, behavior of the tacit, possibly synthetic model of the environment to which the algorithm specification is targeted, and the syntax and semantics of the specification artifact. The parts of the realm that relate to real-world implementation may then be further mapped to the entities of the intended implementation. By contrast, in problem frames the starting point is the real world, and the physical application of the system being developed. Constructing a well-defined model for these, with a consistent terminology can help create useful specifications and detailed models, and eventually, a final working system.

Similarly, as stated in Sec. 4, when constructing a realm for enabling the execution of a specification, lower level activities may be reminiscent of activities in modeling system structure and behavior from initial specifications (Booch, 2005; Deeptimahanti and Babar, 2009). However, the bigger pictures of the inputs and outputs of the modeling process, and subsequent use of the constructed model are different. In system and software engineering, modeling is part of the design and development of a system; the model permeates all aspects of the system and of its development process, where an algorithm-specification's realm is an essential synthetic artifact, which may be detached from any particular system. A realm is governed by an existing algorithm specification for the purpose of executing it.

One may extend the concept of a realm from a single algorithm to an entire system; a discussion of the importance of documenting and precisely scoping assumptions, including deliberately excluded environment and context aspects, are discussed in (Marron et al., 2023; Harel et al., 2025).

One may argue that an agent equipped with a powerful LLM may be able to execute many algorithm specifications, or generate code that does so (Jiang et al., 2024), without depending on any realms, as it may have all the necessary world and domain knowledge (contents of the dashed circle in Figure 1). We believe that identifying what knowledge the agent actually uses and depends on is critical for analyzing the algorithm specification and individual executions (See Sec. 6 below).

6 DISCUSSION

Significance. First, introducing the concept of the realm of an algorithm specification, and the various types of knowledge and skills its execution assumes or requires, elucidates the general concepts of an algorithm, specification, and execution.

Second, having a realm for a given specification assists in developing implementations in diverse systems. Moreover, whether or not LLMs are used in realm construction, articulating the tacit information may help in code generation, formalization for verification, and other algorithm analysis activities. For example, in preparing for formal verification, the elements defined in the algorithm specification and its realm may provide an initial inventory of the states and transitions that constitute the state-transition system underlying the model to be verified (Baier and Katoen, 2008, Ch.2). Naturally, the model to be verified must specify additional aspects, such as desired ("good") and undesired ("bad") states and terminal or stopping states, and it may reflect further refinement or abstraction of the raw elements in the algorithm specification and its realm.

Third, systematic approaches for constructing a specification's realm, for which we have presented a preliminary outline, can be integrated into methodologies for requirements engineering, object-oriented modeling, and the modeling stage in formal methods.

Fourth, the realm contents may contribute to interpretability of an algorithm specification or explainability of its actual behavior. Given that algorithm specifications may themselves be computer generated (e.g., by LLMs), the insights added by the realm can become indispensable.

Finally, it will be interesting to investigate the applicability of the concept of realm as a complement of a narrow specification in constructing test environments for individual software modules and components in system development.

Faithfulness of Execution. Consider an execution log or real-time observation of an arbitrary agent *G* executing an arbitrary algorithm specification *S*, with or without a realm, specified in NL or pseudocode. An external observer — perhaps the very author of *S*, may determine whether the steps taken by *G* are as in-

tended. Assessing faithfulness of execution is distinct from checking correctness of the specification, which is associated with goals, pre- and post-conditions for the overall execution, and related properties.

When the algorithm specification is in a well-defined formalism such as a programming language or a Turing machine, faithfulness is implicit and guaranteed by the formal semantics. For an algorithm specification in NL or pseudocode, one possible approach could call for associating every imperative instruction with pre- and post-conditions defining its intended effects. Nevertheless, while the goal of constructing the realm is to enable faithful execution, in this paper we only highlight the issue, and defer to future research the question of how to define and how to measure faithfulness.

Repetitive Execution. Definitions of the concept of an algorithm imply that an agent executing an algorithm specification can carry out the execution again and again — on the same or diverse inputs. We observe that, for an algorithm specification S, the construction of its realm R is required only once.

Realm Depth. The process of constructing the realm R of an algorithm specification S outlined in Sec. 4 includes an iterative questioning approach as in "what is `<this>`", which may be applied to almost any element of S and R, and thus continue indefinitely. Furthermore, the basic skills of the executing agent may contain patterns that the author of S does not care about, or is willing to accept common defaults for, and thus wishes to stop the flow of questions. For example, realm construction may yield the question: "to what precision should the agent compute the real number V?", or "what rounding protocol should be used when complying with the desired precision?".

We propose several approaches for addressing this dilemma: (1) Continue the process until the binding of the elements of S and R to the capabilities of the agent can be complete, possibly using default values. Some such defaults may not be known and are left to the agent at execution time. Such choices can then be evaluated in the context of execution faithfulness mentioned above. (2) Reexamine the choice of agent! A more capable agent or a more naïve one may be more suitable for the execution of S. (3) Revise S. The challenging questions when constructing the realm may highlight the fact that the algorithm specification is indeed deficient and requires work.

Realm Practicality. One may wonder whether a well-constructed realm, while finite, will be too large to be useful. Here the answer is simple: while human engineers may study the realm and learn from it, it is a resource for specification execution agents, and LLM-based code generation tools. For these, the size of the realm is less of an issue. Also, in many domains realms may be shared across multiple specifications documents. Such reuse then affects not only each realms' sizes, but also the effort to construct them.

7 CONCLUSION AND FUTURE RESEARCH

In this position paper, we have introduced the concept of the realm of an algorithm specification. The realm elucidates and expands the specification with the domain and general information and tools needed for mechanical execution.

Once constructed, the realm can be used in repeated execution, and is a useful resource for implementations and for formal verification. Future research directions include producing detailed, comprehensive realms for a variety of algorithms to serve as a reference for the concept; refining the categorization of realm contents; developing tools for automated realm construction; and establishing a structured process for using an algorithm-specification's realm for constructing a formally verifiable model. The realm concept emerged from the study of the fundamental concept of an algorithm and its specification; it will be interesting to explore the applicability of such realms in diverse areas in system and software engineering.

ACKNOWLEDGMENTS

We thank Kenneth Beckmann for insights and references about algorithms in the art of violin making. We thank Orley K. Marron for valuable discussions and suggestions. This research was funded in part by research grants to DH from Louis J. Lavigne and Nancy Rothman, the Carter Chapman Shreve Family Foundation, Dr. and Mrs. Donald Rivin, and the Estate of Smigel Trust.

REFERENCES

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

Bie, M., Liu, Y., Li, G., Hong, J., and Li, J. (2023). Real-time vehicle detection algorithm based on a lightweight you-only-look-once (yolov5n-l) approach. *Expert Systems with Applications*, 213:119108.

Booch, G. (2005). *The Unified Modeling Language User Guide*. Pearson Education India.

Deeptimahanti, D. K. and Babar, M. A. (2009). An automated tool for generating UML models from natural language requirements. In *2009 IEEE/ACM Intl. Conf. on Automated Software Eng.*, pages 680–682. IEEE.

Denis, F. (2006). Traité de lutherie. *Aladfi, Paris*. https://traitedelutherie.com/en/. Acc. 10/2025.

Gurevich, Y. (2000). Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic (TOCL)*, 1(1):77–111.

Harel, D. (1987a). *Algorithmics: The Spirit of Computing*. Addison-Wesley.

Harel, D. (1987b). Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274.

Harel, D., Aßmann, U., Fournier, F., Limonad, L., Marron, A., and Szekely, S. (2025). Toward methodical discovery and handling of hidden assumptions in complex systems and models. In *Engineering Safe and Trustworthy Cyber Physical Systems: Essays Dedicated to Werner Damm on the Occasion of His 71st Birthday*, pages 147–162. Springer.

Harel, D. and Marron, A. (2018). Toward Scenario-based Algorithmics. In *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, pages 549–567. Springer.

Harel, D., Marron, A., and Yerushalmi, R. (2021). Scenario-Based Algorithmics: Coding Algorithms by Automatic Composition of Separate Concerns. *Computer*, 54(10):95–101.

Hill, R. K. (2016). What an algorithm is. *Philosophy & Technology*, 29(1):35–59.

Jackson, M. (2005). Problem frames and software engineering. *Information and Software Technology*, 47(14):903–912.

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. (2024). A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Marron, A., Cohen, I. R., Frankel, G., Harel, D., and Szekely, S. (2023). Challenges in modeling and unmodeling complex reactive systems: Interaction networks, reaction to emergent effects, reactive rule composition, and multiple time scales. In *International Conference on Model-Driven Engineering and Software Development - Extended Versions of Selected Papers*, pages 137–157. Springer.

Moschovakis, Y. N. (1998). On founding the theory of algorithms. *Truth in Mathematics*, pages 71–104.

Papayannopoulos, P. (2023). On algorithms, effective procedures, and their definitions. *Philosophia Mathematica*, 31(3):291–329.

Primiero, G. (2020). *On the Foundations of Computing*. Oxford University Press.