

Several Strategies for Encoding an Algorithm Using a Pure Web Ontology Language

Olegs Verhodubs

oleg.verhodub@inbox.lv

Abstract. Algorithms are an important form of knowledge. All technological development of our civilization is based on the execution of specific algorithms to achieve desired results. The comprehensive development of technology has generated so many algorithms that preserving them in books and papers is ineffective in terms of accessibility. Semantic Web technologies, specifically the Web Ontology Language (OWL), can help preserve and ensure the accessibility of algorithms. The goal of this paper is to describe several possible approaches for describing algorithms using pure OWL language. In a sense, this study demonstrates the usefulness of a pure OWL language for describing not only static subject areas but also dynamic knowledge, such as algorithms. This research is the initial step in a chain of other steps aimed at the automatic (without human intervention) development of new, non-trivial algorithms.

Keywords: Algorithms, Ontologies, Semantic Web

I. Introduction

The accumulation of knowledge has been a priority for humanity for many millennia. Initially, this knowledge consisted of the simplest relationships between actions and their results, but over time, entire chains of actions and their results became memorized. These became fully-fledged algorithms, executed by people themselves. While the most basic knowledge was passed down orally, various methods of recording it were now used: stone, papyrus, parchment, and later paper. Over time, the accumulated knowledge became so vast that navigating it became extremely difficult. The invention of the computer and then the internet improved the situation, as storing and searching for what you need became convenient and fast. However, increasingly rapidly advancing technologies require new approaches to storing and accessing knowledge, or more precisely, knowledge in the form of algorithms.

Web Ontology Language (OWL) [1] is a well-established tool for describing a specific domain. There are numerous successful and widely used domain ontologies that have proven effective in various fields, such as science, business, and engineering. Examples include the Systematized Nomenclature of Medicine — Clinical Terms (SNOMED CT) [2], the Financial Industry Business Ontology (FIBO) [3], the Conceptual Reference Model [4], the Semantic Sensor Network (SSN/SOSA) [5], and many others. The building blocks of any ontology—concepts, instances, properties, relations, and axioms—are fully sufficient to describe any domain. While there are many domains, the number of algorithms is vastly greater, and algorithms constitute a form of knowledge just as domains do. Despite the importance of algorithms, standard OWL does not have built-in capabilities for describing all programming language constructs, so OWL extensions such as the Semantic Web Rule Language (SWRL) [6], Rule Interchange Format [7], OWL-based Web Services Ontology (OWL-S) [8], Web Services Modeling Ontology (WSMO) [9], or others are typically used to describe a particular algorithm. Creating something new every time the slightest difficulty arises is not a good strategy. This always indicates either insufficient thought about the problem or the failure of previous solutions. This also applies to the possibility of encoding algorithms using existing standard ontology language tools. It seems possible to

encode algorithms using OWL tools without resorting to extensions. This paper describes several strategies, or directions, that can be followed for encoding algorithms using OWL language constructs. The research presented in this paper is not idle fantasy, but also has a practical dimension. In addition to developing an access-efficient method for storing algorithms, this research can serve as a basis for automatically generating algorithms in an alternative way, that is, without using artificial neural networks.

This paper is organized into several sections. The next section briefly describes the algorithms and the algorithmic elements to be encoded. The third section directly describes the types of algorithm encodings. This is followed by conclusions, which summarize the results of this study and outline future prospects.

II. Algorithm and its elements

We are accustomed to thinking that algorithms have become an integral part of our lives with the widespread use of computers. Of course, this is not true, because in fact, algorithms have always been an integral part of our lives. Rarely did a single action lead to the desired result; typically, several actions had to be performed in a strictly defined order to achieve the desired result. Essentially, this is the definition of an algorithm: a sequence of actions in a strict order to achieve the desired result. There are many other definitions, although in reality, most of them are paraphrases of the definition of an algorithm given above. For example, an algorithm is a set of finite, well-defined steps or instructions designed to solve a problem or perform a computation [10]. Another definition of an algorithm is: an algorithm is a set of instructions that is designed to accomplish a task [11]. Despite the existence of algorithms in both real life and the virtual world (the world of computers), in this paper we will focus on algorithms in computers. Therefore, the concept of an algorithm should be narrowed down to this: an algorithm is an ordered sequence of computer instructions for solving a specific problem.

Any algorithm running on a computer can be thought of as a process that transforms input data into output data. There are a wide variety of data representation tools, including OWL, regardless of whether the input or output of an algorithm is being considered. Therefore, it is not so much the data representation that's of interest, but the process of transforming one data set into another. The process of transforming one data set into another is accomplished using standard algorithmic constructs, such as:

- declaring variables and assigning values to them,
- declaring and calling subroutines (functions),
- comparing values and expressions (conditional operators),
- arithmetic and logical operations,
- loops.

More precisely, each of the above-mentioned generalized constructs of any algorithm can be broken down into several related constructs. For example, a variable can contain different values, hence the need for different types. Subroutines may or may not return values, hence the emergence of procedures and functions. Loops, in turn, can be iterative, with a precondition and postcondition.

An algorithm can be expressed on a computer in various ways. It can be a compiled program as an EXE file or a text file with constructs in a high-level programming language. This paper

primarily considers algorithms described in a high-level language, rather than in machine code or assembly. However, any high-level language adds some additional constructs to the standard language constructs. For example, in addition to the standard conditional operator, Python also has a conditional expression, while Pascal or C++ have a multiple-choice operator. These additional constructs are numerous and vary from language to language. Therefore, it is important to define a core set of essential constructs that are indispensable, while other optional, redundant constructs can be neglected if they can be expressed using the core set of constructs. The problem of defining a core set of programming language constructs is solved in two different ways. The first way involves choosing a specific programming language as a basis, all of whose constructs will be taken as a standard. A major drawback of this way is the fact that a specific programming language is chosen. The popularity of the chosen language may decline dramatically in the future, negatively impacting its development. In this case, all new algorithms will have to be described using inappropriate constructs from the dying programming language. The second way is to develop basic algorithm constructs independently. This way is preferable, especially since much of the work in defining a basic, minimal set of abstract constructs has already been completed. Graphical representation of algorithms is one of the most visual ways to display them. A set of basic constructs (elements) has already been developed for this way, allowing any algorithm to be described without being tied to a specific programming language. Graphical descriptions of algorithms are also called flowcharts. The main elements of flowcharts are [12]:

- The beginning and end of an algorithm (oval),
- data input and output (parallelogram),
- execution of operations on data,
- branching of an algorithm or condition (diamond),
- subroutine call,
- loop,
- preparation or initialization of data.

Essentially, this list of basic flowchart elements differs little from the list of algorithmic constructs for transforming one data set into another. Combining the first and second lists of algorithmic constructs, we obtain everything that needs to be implemented using the OWL language:

- The beginning and end of an algorithm,
- declaration and initialization of data,
- data input and output,
- declaring and calling subroutines (functions),
- loop,
- comparing values and expressions (conditional operators),
- arithmetic and logical operations,
- arithmetic, logical, and comparison operations.

It should be noted that the construct for indicating the beginning and end of an algorithm is redundant, since the start and end of the algorithm can be indicated by the initial and final constructs of the OWL code, respectively.

So, the basic constructs of the algorithm are defined above, therefore now it is possible to look at the different strategies of how an algorithm can be coded using pure (without additional add-ons) OWL language.

III. Algorithm coding strategies

Data encoding is the process of transforming data from one form to another. Coding an algorithm using OWL means transforming all constructs of a given algorithm, expressed either in abstract form or in one of the existing programming languages, into the syntax of the OWL language. Despite the finite number of expressive means available in the OWL language, encoding an algorithm using OWL can be implemented in a variety of ways (strategies).

A: Expansion of the OWL

The first strategy for encoding an algorithm in the OWL language is to supplement the OWL language with new missing constructs. For example, the current OWL standard lacks a separate construct for describing a loop, so a new construct for this purpose must be introduced into the OWL standard. Following this strategy, the same approach should be taken for other missing elements of the algorithm.

This strategy has serious drawbacks. First, it contradicts the well-known principle: entities should not be multiplied without necessity. This principle was authored by Ockham and was also known as Ockham's razor. Secondly, without relying on authoritative opinions, the lexicon of any language would grow enormously if each concept were represented by a separate construct. The value of any language lies in having a compact set of constituent elements from which anything can be represented. It's like the periodic table: composing a system of elementary particles from which any substance can be represented. The criterion for introducing a new concept or construct into a language is determining whether the new phenomenon can be represented by the existing language resources. If something cannot be represented by the existing resources, then naturally, an additional construct should be added to the language; otherwise, it is best to refrain from doing so. What are the alternatives? Let us outline other strategies for encoding an algorithm in the syntax of the OWL.

B: Algorithm ontology

OWL is a knowledge representation language for constructing ontologies. Ontology is a formal, explicit specification of a shared conceptualization [13]. In other words, ontologies are intended to describe a subject area in terms of its concepts and the relationships between them. Ontologies consist of concepts (classes), relations (or properties), instances and axioms and from the mathematical point of view, each ontology can be expressed as a 4-tuple $\langle C, R, I, A \rangle$, where C is a set of concepts, R is a set of relations, I is a set of instances and A is a set of axioms [14]. The second strategy for encoding an algorithm in OWL is to develop a dedicated OWL ontology for describing algorithms. This means that a separate class is developed for each element of the algorithm. When describing a specific algorithm, an instance of the corresponding class is used. In practice, this works as follows. A separate class is created for the conditional statement, the loop, the assignment operator, and for the other elements of each algorithm. Schematically, this can be shown as follows (Fig. 1):

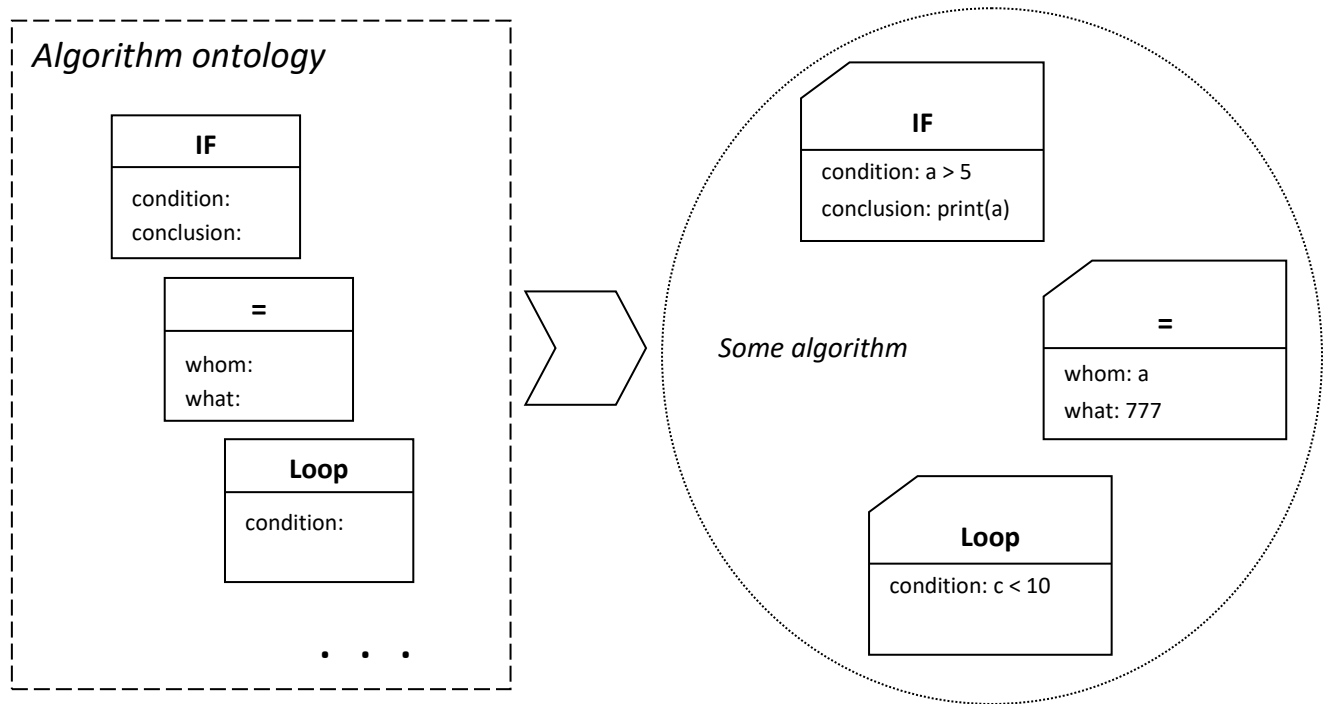


Fig.1. Algorithm ontology and some coded algorithm.

A specific algorithm, for example, such as bubble sort or another, is described by instances of these classes, namely, an instance of the loop class, an instance of the conditional statement class, and other instances of these classes.

This strategy for encoding an algorithm in OWL appears more organic than the first strategy, as it is based on the semantic technologies: that is, defining the parts and the relationships between them, and then describing everything necessary in terms of these parts and the relationships between the parts. A potential drawback of this strategy is the intermediate link between the OWL language and the algorithm being encoded. In this case, the intermediate link is the algorithm ontology. In some cases, an intermediate link can be an obstacle, so if possible, it is better to do without it.

C: Using rules

The essence of the third strategy for encoding an algorithm by means of the OWL is to represent any algorithm as a set of rules in the IF...THEN form, where each rule is expressed by one of the standard OWL constructs. This is, in principle, possible because, firstly, every operator, every statement of the algorithm, can be expressed as a rule, and secondly, a rule can be generated from almost any OWL construct. For example, an assignment operator assigns a value to a variable. But this assignment operator can be written as a conditional statement, where the condition is specified as the constant value, which is equal to True. In Python, this looks like this:

instead of `a = 777`, we should write `if True: a = 777`

Similarly:

instead of `print('Hello, world')`, we should write `if True: print('Hello, world')`

Of course, if an algorithm contains a conditional statement, it should not be modified by adding a second condition; that is, the conditional statement remains unchanged. For example, if there is a conditional statement like this:

if $c < 100$: $a = 777$,

then it remains unchanged.

That is, no one prevents us from inserting a condition before each operator or action that is always True, meaning a condition that is always satisfied. A similar technique can be used with other statements as well. Apparently, the loop operator can present some difficulty. One way to encode a loop using a conditional statement is to describe each loop iteration as a separate conditional statement. This can be cumbersome if the loop has many iterations. A second way to encode a loop using a conditional statement is to use a transition operator (goto operator) to another OWL ontology construct. In OWL ontology terms, a transition operator can appear as a simple reference to the required construct, which can be implemented as a corresponding entry in the comment for the OWL ontology construct. Now, regarding the encoding of rules using ontology. Generally speaking, a rule can be generated from any ontology construct, as has been demonstrated in previous papers [15], [16]. Therefore, to encode the desired rule in an ontology, one simply needs to select the appropriate construct.

To summarize, the procedure for encoding any algorithm by means of the OWL language is the following:

1. have an algorithm described in abstract form,
2. represent this algorithm as a set of rules, that is, as IF...THEN statements,
3. describe each rule as an OWL construct (class, property, relation, etc.)

From the one hand, a disadvantage of this strategy is the potential difficulty in coding some typical algorithmic constructs, such as loops and some other constructs. On the other hand, a major advantage of this strategy is the uniformity in the description of texts, knowledge bases, and algorithms, achieved by using the standard capabilities of the OWL language without resorting to various add-ons.

IV. Conclusion

Three strategies for encoding algorithms using the OWL language were discussed in this paper. Each strategy has its pros and cons, but the third strategy, which treats each algorithm as a set of rules and describes these rules as OWL language constructs, appears to be the most promising. This strategy's potential lies in the lack of distinction between the ontology expressing a given subject area and the ontology expressing a given algorithm. This allows for the transition from one form of information to another depending on the goals of the current task.

Algorithm storage is an important task, but this research opens up some prospects for the automated generation of necessary algorithms. The goal is to automatically (without human intervention) generate an algorithm that achieves a result from a given starting point. Representing an algorithm using an OWL ontology is the first step in this direction.

Acknowledgments

Gratitude is expressed to my friends and family for their long-term moral and material support in the face of numerous illegal and discriminatory actions by the Latvian state, namely the Latvian police, the Latvian prosecutor's office, the Latvian State Security Service, and the Riga Technical University, against the author in particular and against national minorities living in Latvia in general.

The illegal and discriminatory actions by the aforementioned Latvian institutions against the author can be proven by independent, qualified, legal, non-Latvian expert analysis.

References

- [1] Available online: <https://www.w3.org/TR/owl-features/> [Accessed: August 1, 2026]
- [2] Available online: <https://www.nlm.nih.gov/healthit/snomedct/index.html> [Accessed: August 1, 2026]
- [3] Available online: <https://spec.edmcouncil.org/fibo/> [Accessed: August 1, 2026]
- [4] Available online: <https://cidoc-crm.org/> [Accessed: August 1, 2026]
- [5] Available online: <https://www.w3.org/TR/vocab-ssn/> [Accessed: August 1, 2026]
- [6] Available online: <https://www.w3.org/submissions/SWRL/> [Accessed: August 1, 2026]
- [7] Available online: <https://www.w3.org/TR/rif-overview/> [Accessed: August 1, 2026]
- [8] Available online: <https://www.w3.org/submissions/OWL-S/> [Accessed: August 1, 2026]
- [9] Available online: <https://www.w3.org/submissions/WSMO/> [Accessed: August 1, 2026]
- [10] Available online: <https://www.geeksforgeeks.org/dsa/introduction-to-algorithms/> [Accessed: August 1, 2026]
- [11] Available online: <https://www.nlm.nih.gov/resources/data/data-glossary/algorithm> [Accessed: August 1, 2026]
- [12] Available online: <https://pro-prof.com/archives/1462> [Accessed: August 1, 2026] (in Russian)
- [13] F. Bellifemine, G. Caire, T. Trucco and G. Rimassa, "JADE programmer's guide", 2010.
- [14] J. Davis, R. Studer, P. Warren, Semantic Web Technologies Trends and Research in Ontology-based Systems. Chichester: John Wiley & Sons Ltd, 2006.
- [15] O. Verhodubs, J. Grundspenkis, Evolution of ontology potential for generation of rules, 2012.
- [16] O. Verhodubs, Ontology as a source for rule generation, 2014.