Building an Ontology of Computational Complexity

Anton Gnatenko[1,*], Oliver Kutz[1,*] and Nicolas Troquard[2,*]

[1]Free University of Bozen-Bolzano, Italy
[2]Gran Sasso Science Institute, Italy

Abstract

The field of computational complexity theory is a core theoretical subject in computer science with significant impact also for real-world applications. Although a plethora of individual results are known, the conceptual organisation of this knowledge is still lacking. We propose the first steps towards creating an ontologically well-founded knowledge base for the theory of computational complexity that will allow storing, querying and reasoning over the vast knowledge of algorithmic problems, complexity classes and their relationships, developed by human experts. We determine the core concepts and relations of complexity theory and model them on two levels of approximation: the description logic $\mathcal{SROIQ}$ (a.k.a. OWL 2 DL) and first-order logic. Finally, we point out a number of phenomena that require more expressive formalisms beyond the first-order paradigm.

Keywords

Domain modelling, computational complexity, description logic

1. History and Motivation

In two papers, now amongst the seminal contributions in the history of logic and the theory of computation, Alonzo Church and Alan Turing gave negative answers to Hilbert and Ackermann's 1928 decision problem (Entscheidungsproblem): is there a general algorithm to determine whether a formula expressed in first-order logic is a theorem of first-order logic [1, 2]. To answer the question, Church and Turing first had to formally define what an algorithm is. Church developed the λ-calculus and Turing defined a special class of automata, or machines, that now bear his name. Generalising from the Entscheidungsproblem, a *decision problem* is a function that given an input returns 'yes' or 'no'. After Church and Turing, mathematicians had to distinguish two kinds of decision problems: those for which there is a general algorithm to determine whether it returns 'yes' or 'no' on a given input, called *decidable*, and those (like the theoremhood of first-order logic formulas) for which there is no such algorithm, which are now called *undecidable*.

Remarkably, it took nearly three decades for Hartmanis and Stearns [3] to explicitly note a peculiar aspect of decidable decision problems. Some of them appeared to necessitate "more expensive" algorithms than others. They proposed to measure the amount of time used by algorithms as a function of the size of the input. Starting from there, it became possible to study

Proceedings of the Joint Ontology Workshops (JOWO) - Episode X: The Tukker Zomer of Ontology, and satellite events co-located with the 14th International Conference on Formal Ontology in Information Systems (FOIS 2024), July 15-19, 2024, Enschede, The Netherlands.

*Corresponding author.
agnatenko@unibz.it (A. Gnatenko); oliver.kutz@unibz.it (O. Kutz); nicolas.troquard@gssi.it (N. Troquard)
0000-0003-1499-2090 (A. Gnatenko); 0000-0003-1517-7354 (O. Kutz); 0000-0002-5763-6080 (N. Troquard)

and compare algorithms and decision problems based on the computational resources they require. This is the field of *computational complexity*.

At the centre of research in the theory of computational complexity, or complexity theory, for short, are the complexity classes, i.e. collections of problems that feature the same inherent complexity. Throughout more than half a century of its history, the concept of computational complexity has grown refined and detailed, with many definitions depending on diverse models of computation (deterministic, nondeterministic, and probabilistic machines, alternating computations, Boolean and quantum circuits, interactive protocols, etc.) as well as different measures of complexity (time and space, circuit size and depth, the number of random coin tosses used in a computation, the number of messages exchanged between communication agents, and many more). All of this generates a vast variety of complexity classes related to each other in an intricate manner. The discoveries in the field feature both general theorems, that refer to the properties of whole families of complexity classes, as well as concrete facts about mutual connections between two classes or a class and a problem.

In a welcome statement to the Complexity Zoo, a website that attempts to collect all the known results in the field with comments and references, its creator Scott Aaronson writes: "I spent a week trying to put AM outside QMA relative to an oracle, only to learn that this followed trivially from two known results. ⟨...⟩ Some theorists seem able to hold in their minds, in one instant, the results of every FOCS, STOC, and Complexity paper ever published. ⟨...⟩ I am not one of those theorists. ⟨...⟩ And so it's largely for my own benefit that I recorded a chunk of what's known in one unwieldy HTML file".[1]

The Zoo made the life easier for many complexity theorists and many more researchers in other areas of computer science, including the authors of this article. However, it has all the standard drawbacks of old-fashioned plain-text knowledge. First, it does not contain the structural information: if the class P is a subset of the class NP, one can read it in the class description, but the very fact that $\mathrm{P} \subseteq \mathrm{NP}$ is not stored there explicitly, it cannot be searched for or used as part of a more complex query. In fact, there is no way to query the Zoo data apart from using textual search. Second, it is not protected from errors or typos: should someone accidentally write that $\mathrm{P} \not\subseteq \mathrm{NP}$, no one apart from a human user would be able to detect it and report it as a problem. Finally, even though the Zoo can provide the reader with facts and details that help to establish previously unknown relationships between complexity classes, there is no way to do this automatically by *inferring* new facts from given ones.

We aim at creating a *semantic* Complexity Zoo, a knowledge graph populated with decision problems and complexity classes, formalising their properties and relations, and equipped with an ontology that provides the semantics and allows the use of off-the-shelf reasoners for query answering, consistency checking and inference tasks. In fact, some of the phenomena of complexity theory, such as infinite hierarchies of complexity classes, actually *require* ontological axioms even to be described, since they can not be fully represented in the data for obvious reasons. A modest target is building a system capable of answering simple queries and making visualisations, thus facilitating the search of information for researchers and students. An ambitious goal is

[1] See https://complexityzoo.net/Complexity_Zoo_Introduction. AM and QMA are complexity classes, FOCS is the IEEE Symposium on Foundations of Computer Science, STOC—the ACM Symposium on Theory of Computing, and 'Complexity' refers to the Computational Complexity Conference.

using it to discover previously unknown, even if simple, facts of the complexity world that follow from the existing results, but that have been overlooked so far.

This paper is the first step on this way. We discuss an approach to ontological modelling of complexity theory phenomena that is compatible with practical ontological languages and tools, such as OWL 2 [4] and FOWL [5]. The latter is a framework for incorporating first-order axioms into an OWL 2 ontology and alternating between reasoning in a restricted OWL 2 setting or with the full power of first-order logic.

Our work uses elements of Semantic Web technologies and addresses issues of foundational ontologies and the philosophy of mathematics, similar to efforts in formalised mathematics [6]. One of the few examples of ontologies pursuing a similar goal is 'OntoMathPRO' [7], however they do not address computational complexity directly. Indeed, we are not aware of any work that specifically addresses the ontological modelling challenges of computational complexity theory, particularly aimed at concretely modelling the existing factual data.

In contrast, many ontological investigations into mathematical objects focus on foundational issues revolving around set theory, categories, and numbers, and explore the classical philosophical dilemma asking how abstract objects can be so effective and seemingly indispensable in the systems of modern physics, see [8, 9]. Our goal here is more modest, namely to address the gap of ontologically adequate formal modelling of a particular branch of mathematics, complexity theory, so that it can be faithfully and efficiently used in practice.

In Section 2, we introduce, more formally, the notions of complexity theory and highlight the problems that arise when modelling them in an ontology. After that, in Section 3 we axiomatise the domain using a FOWL-inspired approach, that is, writing simpler axioms in OWL 2 DL and the more involved ones in first-order logic. We outline a couple of cases where the power of first-order logic is yet not enough to capture them. The presentation is wrapped up in Section 4 where we briefly discuss possible ways of further deepening and broadening our ontology, including extending it with a second-order layer to express more subtle details of complexity theory.

2. The Domain of Computational Complexity

The primary objects studied in complexity theory are decision problems and complexity classes. We start with the definitions. Let f be a mapping $\{0,1\}^* \to \{0,1\}^*$, i.e. a mapping of finite bit-strings to finite bit-strings. Following the tradition, we will call these bit-strings "words". An *algorithm* computes a function f if on any input word w it outputs $f(w)$. In this article, we will use the notion of Turing machine, which is the most common model of computation, to formalise the term "algorithm", although we plan to include other models to our ontology in the future. For an algorithm formulated as a Turing machine M, the *time complexity* is a mapping $T_M \colon \mathbb{N} \to \mathbb{N}$, such that on an input word of length n the machine performs at most $T_M(n)$ steps before halting. *Space complexity*, instead, gives the upper bound on the number of tape cells used by the machine on an input of length n. The machine itself may be standard deterministic, with no additional features, or allowed to use extra computational powers such as the ability to correctly "guess" some useful information (non-deterministic machines), or use randomness to facilitate its computations (probabilistic machines), etc. Apart from complexity of individual algorithms, computational problems themselves have a kind of inherent complexity, as observed

by Hartmanis and Stearns [3]. The time (space) complexity of a function f is defined as the minimal time (respectively, space) complexity of an algorithm that computes it.

Traditionally, complexity theorists consider a simpler kind of bit-string functions, whose output consists of just one bit. Such a function $a\colon \{0,1\}^* \to \{0,1\}$ is called a *decision problem* (or just a *problem*), since a can be viewed as a characteristic function of a set, and to compute a means to "decide" whether the input is in that set. We thus write $w \in a$ and $w \notin a$ if, respectively, $a(w) = 1$ and $a(w) = 0$. A famous example of a decision problem is SAT: given a Boolean formula, decide whether it is satisfiable. *Complexity classes*, or just classes, for brevity, are defined as sets of decision problems. For example, P is the class of problems decidable by a deterministic Turing machine in polynomial time. Another related class, NP, contains problems decidable in polynomial time by a non-deterministic Turing machine. PSPACE is the class of problems decidable by a deterministic Turing machine using polynomial space. We invite the reader to consult Arora and Barak [10] for a detailed presentation of the domain.

2.1. Modelling the Basics of Complexity Theory

We describe the main relations, facts, and theorems of computational complexity that, in our opinion, constitute the "core knowledge" about complexity that we would like to represent in our ontology, and outline which challenges arise from it.

2.1.1. Problems, Classes, and Their Relationships

The universe of complexity theory contains entities of different types. For example, there are decision problems, and there are classes, that are sets of problems. A problem may be or may be not an element of a class, but for two classes such a question does not make sense. This difference must be preserved in the ontology and standard relations shall be axiomatised in order to respect it.

Challenge 1. *Handling objects of different types.*

Problems are members of classes, and classes can contain, or be subsumed by, other classes. Some set-theoretic relations between them already constitute theorems of complexity theory. For example, it is an easy consequence of the definitions that SAT $\in$ NP and P $\subseteq$ NP, but it requires some effort to show PSPACE $\subseteq$ EXPTIME, i.e. that every problem decidable in polynomial space is also decidable in exponential time. Moreover, we know that if SAT $\in$ P then P $=$ NP [11, 12]. On the other hand, if SAT $\notin$ P then P $\neq$ NP. These are negative statements that are nonetheless important for working in complexity theory. Therefore, set-theoretic relations and their negative counterparts should be the basis of the ontology.

Challenge 2. *Representing and reasoning about basic set-theoretic relations where both the elements and the sets are objects of the ontology.*

2.1.2. Infinite Constructions

Typically, a complexity class is defined with respect to a particular computation model, particular kind of complexity (e.g. time or space) and a bound on that complexity. In this first version of our

ontology we will focus on various Turing machines as the models, and time and space complexities. However, for the bounds things are more interesting. Every function $g\colon \mathbb{N} \to \mathbb{N}$ gives rise to classes $\mathrm{DTIME}(g)$ (that of problems decidable in time $\alpha g(n) + \beta, \alpha, \beta \in \mathbb{N}$, by a deterministic Turing machine), $\mathrm{NTIME}(g)$ (the same for nondeterministic Turing machines, which accept if they are able to guess a way to the accepting state), $\mathrm{coNTIME}(g)$ (for co-nondeterministic machines, that accept if all their guesses bring them to an accepting state), and $\mathrm{ATIME}(g)$ (for alternating machines, that switch between nondeterministic and co-nondeterministic computations), as well as their counterparts for space complexity: $\mathrm{DSPACE}(g)$, $\mathrm{NSPACE}(g)$, $\mathrm{coDSPACE}(g)$ and $\mathrm{ASPACE}(g)$. Some of the fundamental theorems make use of this notation. For example, the time hierarchy: if $g(n) = o(h(n) \cdot \log n)$, then $\mathrm{DTIME}(n) \subsetneq \mathrm{DTIME}(h)$, or the similar space hierarchy: if $g(n) = o(h(n))$, then $\mathrm{DSPACE}(n) \subsetneq \mathrm{DSPACE}(h)$. Other examples include the famous Savitch theorem: $\mathrm{NSPACE}(g) \subseteq \mathrm{DSPACE}\left(g^2\right)$, or the standard inclusions $\mathrm{DTIME}(g) \subseteq \mathrm{DSPACE}(g)$ and a bit more involved $\mathrm{DSPACE}(g) \subseteq \mathrm{DTIME}\left(2^g\right)$. However, most of complexity theory is built around "classes" of bounds: constant, logarithmic, polynomial, exponential, double exponential, and so on, with larger complexity classes defined as, e.g., $\mathrm{P} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}\left(n^k\right)$. Therefore, we can provide a less fine-grained representation of complexity bounds and the related theory, but we will anyway need to model an infinite sequence of these "classes" of complexity bounds.

Challenge 3. *Representing infinite sequences of objects and relations between these objects that depend on their positions in the sequence.*

Some of the classes are defined as infinite hierarchies themselves, i.e. sequences of smaller classes each of which may be of separate theoretical and practical interest. Take, for example, the polynomial hierarchy, PH. It consists of two infinite chains of classes, $\Sigma_0^p \subseteq \Sigma_1^p \subseteq \Sigma_2^p \subseteq \ldots$ and $\Pi_0^p \subseteq \Pi_1^p \subseteq \Pi_2^p \subseteq \ldots$. Each Σ_k^p $\left(\Pi_k^p\right)$ is a class of problems decidable by a polynomial-time alternating Turing machine that starts in the nondeterministic mode (respectively, the co-nondeterministic mode) and performs k alternations (see Chandra and Stockmeyer [13], Stockmeyer [14] for more details). Therefore, $\Sigma_k^p \subseteq \Pi_{k+1}^p$ and $\Pi_k^p \subseteq \Sigma_{k+1}^p$. So the union of all $\{\Sigma_p^k\}_{i\geqslant 0}$ equals the union of all $\{\Pi_p^k\}_{i\geqslant 0}$, and that union is denoted by PH. This gives another challenge: we must state that if $a \in \Sigma_k^p$, then $a \in \mathrm{PH}$, and vice versa, if $a \in \mathrm{PH}$, then there exists Σ_k^p that contains a. The latter is non-trivial: this Σ_k^p must exist not just somewhere in the object universe, but in the sequence of the polynomial hierarchy. Thus, PH is defined as sort of a 'limit' of Σ_k^p with respect to $\subseteq$: to define it, we need to talk about the 'whole' sequence as a second-order entity, not just its separate members.

Challenge 4. *Axiomatising second-order properties.*

2.1.3. Relations of Higher Arity

Arguably, the notions of complexity theory that are the most famous and useful for general computer science are those of reductions and hardness. A problem a is *reducible* to a problem b if there is a function r, called *reduction*, such that for every word w we have $w \in a \iff r(w) \in b$ (this definition is due to Karp [12]). An algorithm that solves b can be used to solve a, save the fact that one needs first to compute the result of applying r to the input word. Therefore, reducibility is always considered in complexity theory with respect to a certain complexity bound.

Most of the standard textbook results concentrate on reductions computable by deterministic Turing machines in polynomial time or logarithmic space [10], although some other types of reductions are widely used in specific fields, such as first-order reductions in database theory [15]. A problem a is *hard* for a class A with respect to a particular kind of reductions, if every problem in A is reducible to a using the reductions of that kind. A *complete* problem for A is such that is hard for A and itself a member of A. Complete, and, in general, hard problems play a crucial role in complexity theory, since they allow in many cases the reduction of the reasoning about classes to that of individual problems. Indeed, as we mentioned earlier, it suffices to prove or disprove SAT $\in$ P to argue that P equals (or is not equal to) NP. Reducibility and hardness are, even if it is not always highlighted, relations of arity 3: they are characterised also by the complexity of reductions they use. Moreover, they may have interesting properties, e.g. a composition of polynomial-time reductions is itself a polynomial-time reduction.

There is a more general notion of reductions between problems, called Turing reducibility (as opposed to Karp reducibility discussed above). Instead of converting an input for a to an input for b and then using the algorithm for b, Turing allows calling the algorithm for b arbitrarily many times in the process of solving a. It is usually assumed that a call to the algorithm for b can be performed within one step of computation. In this case, we say that we are solving a relative to *an oracle* b, as if we had an access to a powerful being that could instantly decide the problem b whenever requested. This notion gives rise to many more complexity classes: for any class A and any problem b there is a class A^b of problems that are decidable under the restrictions of A given the access to the oracle b. Moreover, for a class B we define $A^B = \bigcup_{b \in B} A^b$. Many results of complexity theory feature oracles, e.g. the Baker-Gill-Solovay Theorem: there are oracles a and b such that $\mathrm{P}^a = \mathrm{NP}^a$ and $\mathrm{P}^b \neq \mathrm{NP}^b$ [16]. This is a theorem about relations of arity 3.

Challenge 5. *Representing relations of arity higher than 2 and theorems about them.*

2.1.4. Conditional Statements

Complexity theory is rich with questions for which everyone believes in a certain answer but no one can prove it. This leads to a generous supply of theorems of the form "if α then β", where α and β are certain statements. These theorems state a relationship between two assertions, rather than prove a particular assertion itself. For example, if NP $\subseteq$ coNP then NP = coNP, P = NP then ExpTime = NExpTime. If NP $\subseteq$ P/Poly then AM = MA. The first statement is an easy corollary of the definitions, the second is a good example for the use of the so-called "padding argument" [10], and the latter is due to Arvind et al. [17]. Another, more involved, example of a conditional statement is that if P $\neq$ NP then *there exists* a problem in NP $\setminus$ P that is not NP-complete (Ladner's Theorem) [18].

Challenge 6. *Representing conditional statements.*

In the next section we address these challenges from the FOWL perspective.

3. Formalisation in Ontology Languages

The "real-world" ontology languages, e.g. description logics (DLs) [19] and related formalisms such as OWL 2 [4], are limited to work with binary relations, while, as we have discussed

in Section 2.1.3, some of our relations are ternary. This limitation can be circumvented via techniques like reification [20, 21], but only to a certain extent. Second, even the most expressive DLs, such as $\mathcal{SROIQ}$ [22, 23], put a lot of restrictions on axiomatising binary relations. A simple example of this is the equality relation. As a set-theoretic relation between classes, it should be in a correspondence to the subset relation, i.e. we would like to say equals $\equiv$ subset $\sqcap$ subset$^-$, but $\mathcal{SROIQ}$ does not have the construct $\sqcap$ to express the intersection of binary relations.

Issues of this kind have recently received considerable attention in the literature [24, 5]. Experiments suggest that formulating the forbidden axioms in first-order logic allows one to perform many inferences and query answering, using a first-order prover, in acceptable time. Specifically, the tool presented by Flügel et al. [5] allows the user to write first-order axioms as annotations in an `.owl` file, and then use, depending on one's needs, OWL reasoners with the part of the ontology written in OWL, or Vampire, a first-order reasoner [25], on the whole first-order ontology. With this approach at hand, we turn to writing down the axioms.

3.1. Axioms

We go through the challenges listed in Section 2, repeating them and providing (partial) solutions. We use $\mathcal{SROIQ}$, where possible, but retreat to first-order logic when necessary. The reader is invited to consult Rudolph [23] for a detailed definition of $\mathcal{SROIQ}$.

3.1.1. Problems, Classes, and Their Relationships

Types of objects can be naturally represented by using disjoint concepts, addressing Challenge 1. For example, for problems and classes we use concepts Prob and Class, and state that their intersection is an empty concept:

$$\text{Prob} \sqcap \text{Class} \sqsubseteq \bot \tag{1}$$

As for Challenge 2, the set-theoretic relations can be naturally viewed from a second-order perspective. To stay on the first-order and even description logic level, we simulate the second-order relations $\in$ and $\subseteq$ via dedicated binary relations (or *roles*, as binary relations are called in description logics) 'in' and 'subset'. In fact, there is not much that we want from these roles. The subset relation is transitive, and the composition of 'in' and 'subset' gives a 'in':

$$\text{subset} \circ \text{subset} \sqsubseteq \text{subset} \qquad \text{in} \circ \text{subset} \sqsubseteq \text{in} \tag{2}$$

We also ensure that these roles respect types of the objects, e.g. the domain of 'in' consists of problems and its range—of classes, while for 'subset' both the domain and the range are restricted to the objects of type Class:

$$\exists \text{in} \sqsubseteq \text{Prob} \qquad \exists \text{in}^- \sqsubseteq \text{Class} \tag{3}$$

$$\exists \text{subset} \sqsubseteq \text{Class} \qquad \exists \text{subset}^- \sqsubseteq \text{Class} \tag{4}$$

Simulating equality of classes is less easy. Indeed, we can introduce a role 'equals', but the only thing we can say is that it is a sub-relation of both the subset and the superset relations:

$$\text{equals} \sqsubseteq \text{subset} \qquad \text{equals} \sqsubseteq \text{subset}^- \tag{5}$$

The "full" semantics of equality require reasoning about the intersection of roles 'subset' and 'subset$^-$', which is not allowed in $\mathcal{SROIQ}$. We thus add a first-order axiom:

$$\forall x, y \,.\, \text{equals}(x, y) \iff \text{subset}(x, y) \wedge \text{subset}(y, x) \tag{6}$$

We can describe other peculiar relations in the same way, such as the strict subset relation. In $\mathcal{SROIQ}$, we can say that it is a sub-relation of 'subset':

$$\text{strictSubset} \sqsubseteq \text{subset} \tag{7}$$

and on the first-order level we can use negation to say that $\text{strictSubset}(A, B)$ requires that B is larger than A:

$$\forall x, y \,.\, \text{strictSubset}(x, y) \implies \exists z \,.\, \text{in}(z, y) \wedge \neg\text{in}(z, x) \tag{8}$$

3.1.2. Infinite Constructions

As we have mentioned in Section 2.1.2, every function $\mathbb{N} \to \mathbb{N}$ defines a complexity bound, but most of the theory is built around "classes" of bounds on time or space: constant, logarithmic, polynomial, exponential, double-exponential, triple-exponential, and so on, where by k-exponential bound we mean the function $\exp(n, k)$ defined as $\exp(n, 1) = 2^n$ and $\exp(n, k+1) = 2^{\exp(n,k)}$. Of course, the countable sequence of k-exponentials does not exhaust the world of complexity bounds that are present in the literature, but they are enough to represent most of the results. We come back to this question in Section 4.

We suggest to represent each "class" of complexity bounds by an individual, and subsequent bounds (such as logarithmic and polynomial, or k-exponential and $k+1$-exponential, are connected by a functional role 'nextBound'. Moreover, since we can not explicitly list the whole sequence in the data, we need this role to create an infinite chain. This is done by saying that no object has two or more outgoing or incoming 'nextBound' roles, and that an incoming 'nextBound' implies the existence of an outgoing one:

$$\geqslant 2\ \text{nextBound} \sqsubseteq \bot \qquad \geqslant 2\ \text{nextBound}^- \sqsubseteq \bot \qquad \exists\text{nextBound}^- \sqsubseteq \exists\text{nextBound} \tag{9}$$

To create the sequence, we only need to declare the existence of an outgoing 'nextBound' for the first object:

$$\exists\text{nextBound}(\text{Const}) \tag{10}$$

And to access any object in the chain, say the exponential bound, it suffices to list all the previous:

$$\text{nextBound}(\text{Const}, \text{Log}) \qquad \text{nextBound}(\text{Log}, \text{Poly}) \qquad \text{nextBound}(\text{Poly}, \text{Exp}) \tag{11}$$

Complexity classes can now be defined as individuals "attached" to the chain of bounds via specific roles 'dTime', 'nTime', 'conTime', 'aTime', and 'dSpace', 'nSpace', 'conSpace', 'aSpace' (that are all required to be functional). For example, we assert $\text{dTime}(\text{P}, \text{Poly})$, and, similarly, $\text{conTime}(\text{coNP}, \text{Poly})$, $\text{dTime}(\text{ExpTime}, \text{Exp})$, $\text{dSpace}(\text{L}, \text{Log})$, and so on. We express the properties discussed in Section 2.1.2 via role chains. For example, deterministic classes are

subsets of the respective non-deterministic classes, and a space class is always contained in the time class with an exponentially higher bound:

$$\text{dTime} \circ \text{nTime}^- \sqsubseteq \text{subset} \qquad \text{dSpace} \circ \text{nextBound} \circ \text{dTime}^- \sqsubseteq \text{subset} \tag{12}$$

In particular, the second axiom of (12) ensures that if we go from PSPACE along 'dSpace' (arriving at 'Poly'), then along 'nextBound' (to 'Exp'), and then backwards along 'dTime', we will end up at an individual representing a superclass of PSPACE (in this case it will be the class EXPTIME). Technically, by finding the described chain of three roles, we infer $\text{subset}(\text{PSPACE}, \text{EXPTIME})$. The space and time hierarchies are similar:

$$\text{dSpace} \circ \text{nextBound} \circ \text{dSpace}^- \sqsubseteq \text{strictSubset} \tag{13}$$

$$\text{dTime} \circ \text{nextBound} \circ \text{dTime}^- \sqsubseteq \text{strictSubset} \tag{14}$$

Savitch theorem is more complicated. In our representation, it means that the role inclusion $\text{nSpace} \circ \text{dSpace} \sqsubseteq \text{subset}$ holds for all elements of the bounds sequence *starting from* 'Poly'. To handle this, we introduce a concept name 'Savitch' and assert that, first, it is true for 'Poly' and, second, that it spreads along the 'nextBound' role, thus covering the whole chain from 'Poly' and on:

$$\text{Savitch}(\text{Poly}) \qquad \text{Savitch} \sqsubseteq \exists \text{nextBound}.\text{Savitch} \tag{15}$$

That is, for any individual labelled with 'Savitch' there exists an outgoing 'nextBound' to another individual with this label. Since 'nextBound' is functional by axioms (9), the label is inferred for the next element in the chain. Finally, we must state that a role chain of $\text{nSpace} \circ \text{dSpace}^-$ that passes through the label 'Savitch' induces a subset relation. To do so, we introduce a new role name '$\text{nSpace}_{\text{Savitch}}$' and say:

$$\geqslant 2\ \text{nSpace}^- \sqsubseteq \bot \qquad \text{Savitch} \sqsubseteq \exists \text{nSpace}^-_{\text{Savitch}} \qquad \text{nSpace}_{\text{Savitch}} \sqsubseteq \text{nSpace} \tag{16}$$

Thus, the existence of 'Savitch', e.g., at 'Poly', creates an incoming role '$\text{nSpace}_{\text{Savitch}}$', and by design this role has to come along with the existing 'nSpace' role. Finally:

$$\text{nSpace}_{\text{Savitch}} \circ \text{dSpace}^- \sqsubseteq \text{subset} \tag{17}$$

We believe that many theorems about hierarchies can be expressed using this approach. This settles Challenge 3.

Indeed, other hierarchies, such as PH, can be modelled in a similar way. We first create an infinite chain of the levels of the hierarchy, where the ith level is represented by an object ℓ_i, and the objects ℓ_k and ℓ_{k+1} are connected by the '$\text{nextLevel}_{\text{PH}}$' role, that is functional in both directions and has the same properties as 'nextBound' in (9). We start the chain at the level ℓ_0 and attach the class P to it by two parallel roles: $\text{sigma}^p(\text{P}, \ell_0)$ and $\text{pi}^p(\text{P}, \ell_0)$. For the higher levels, these two roles are used to connect to ℓ_k the two respective classes, Σ^p_k and Π^p_k. It is not hard to express the subset relationships between these classes in the same fashion as we did before.

There are two more tricky issues about PH. The first is that the whole hierarchy is contained in PSPACE. This is easy to say in FOL, but we can also do it in $\mathcal{SROIQ}$. For that we introduce a

role 'higherLevel$_{\text{PH}}$', define it as the transitive closure of 'nextLevel$_{\text{PH}}$', and use it to get from any level of the hierarchy back to P, and then to PSPACE, via a suitable chain of roles:

$$\text{nextLevel}_{\text{PH}} \sqsubseteq \text{higherLevel}_{\text{PH}} \qquad \text{nextLevel}_{\text{PH}} \circ \text{higherLevel}_{\text{PH}} \sqsubseteq \text{higherLevel}_{\text{PH}} \tag{18}$$

$$\text{sigma}^{p} \circ \text{higherLevel}_{\text{PH}}^{-} \circ \text{sigma}^{p\,-} \circ \text{dTime} \circ \text{dSpace}^{-} \sqsubseteq \text{subset} \tag{19}$$

Read (19) as follows: from a class Σ_k^p we step, via 'sigma^p', to ℓ_k; from there we get down to ℓ_0, where we by 'sigma$^{p\,-}$' get to P; and from P we get to PSPACE via dTime $\circ$ dSpace$^-$, passing through the object 'Poly' in the chain of bounds.

The second issue is exactly Challenge 4: to define the class PH itself as a "limit" point of the sequence. In FOWL, we are able to give only a partial solution to this problem. Indeed, like with PSPACE above, we can say that $\Sigma_k^p \subseteq \text{PH}$ for all k. However, it remains to state that PH is subsumed by the union of Σ_k^p, i.e. that if 'in(a, PH)' is true, then there exists Σ_k^p in the chain that contains a. We leave it as a question of further development and briefly discuss it again in Section 4.

3.1.3. Relations of Higher Arity

We now settle Challenge 5. We need first-order logic to work with relations of arity 3, such as reachability and hardness. On the other hand, most of reductions in the "textbook" complexity theory are polynomial time or logarithmic space reductions. Therefore, it makes sense to provide $\mathcal{SROIQ}$ variants for these types of reductions.

We introduce roles isReducible$_{\text{L}}$ and isReducible$_{\text{P}}$, and, similarly, isHard$_{\text{L}}$ and isHard$_{\text{P}}$ for hardness relations with respect to these complexity bounds. We can express some of their properties in $\mathcal{SROIQ}$, e.g. both reducibility relations are transitive, being hard for a class results in being hard for any of its subclasses, and if a hard problem is reducible to another one, than that one is also hard. Thus, for $F \in \{\text{L}, \text{P}\}$:

$$\text{isReducible}_F \circ \text{isReducible}_F \sqsubseteq \text{isReducible}_F \tag{20}$$

$$\text{isHard}_F \circ \text{subset}^{-} \sqsubseteq \text{isHard}_F \tag{21}$$

$$(\text{isReducible}_F)^{-} \circ \text{isHard}_F \sqsubseteq \text{isHard}_F \tag{22}$$

Also, as one can expect, every logspace reduction can be computed in polynomial time:

$$\text{isReducible}_{\text{L}} \sqsubseteq \text{isReducible}_{\text{P}} \qquad \text{isHard}_{\text{L}} \sqsubseteq \text{isHard}_{\text{P}} \tag{23}$$

These roles can be connected back to their first-order originals. For example, for logspace reducibility it is done as follows:

$$\forall x, y \,.\, \text{isReducible}_{\text{L}}(x, y) \iff \text{isReducible}(x, y, \text{L}) \tag{24}$$

A class A^b is defined via two objects, A and b. This can be represented by a pair of roles: 'hasBase' and 'hasPower'. For example NP $=$ P$^{\text{SAT}}$, therefore we write hasBase(A, P), hasPower(A, SAT), equals(NP, A). In this setting, stating Ladner's Theorem is done via ABox assertions using special objects that are mentioned in the theorem, i.e. the Ladner's oracle l and the classes P^l, and NP^l, for which the following is stated: hasBase(P^l, P), hasPower(P^l, l), hasBase(NP^l, NP), hasPower(NP^l, l), notEquals$(\text{P}^l, \text{NP}^l)$.

Table 1
Summary of the modelling challenges and solutions

Challenge	Description	Success?
C1	objects of different types	yes
C2	basic set-theoretic relations	yes
C3	axiomatising infinite constructions	yes
C4	axiomatising second-order properties	partial
C5	relations of higher arity	yes
C6	conditional statements	yes

3.1.4. Conditional Statements

Most of the conditional statements can be treated as implications between ABox assertions, that are easy to reason about within Boolean logic, as in

$$\text{equals}(\text{P}, \text{NP}) \implies \text{equals}(\text{ExpTime}, \text{NExpTime}) \tag{25}$$

In cases that assert existence of an object if a certain condition is true, we can introduce such an object as a "certifier" for the result:

$$\neg\text{equals}(\text{P}, \text{NP}) \implies \text{in}(\text{certifier}, \text{NP}) \wedge \neg\text{isHard}_{\text{Poly}}(\text{certifier}, \text{NP}) \tag{26}$$

This is our approach to Challenge 6.

4. Discussion and Future Work

Table 1 summarises the challenges of the formal modelling of complexity-theoretic phenomena that we identified in this paper. Apart from Challenge 4, they can be successfully addressed within the FOWL paradigm. This allows us to proceed with the actual development of a working knowledge base that can be used for reasoning and query answering. Given the axioms that we have, it remains to collect and represent the factual knowledge on complexity theory in a data instance. Our first goal is to extract RDF triples from the Complexity Zoo website. For this, we plan to experiment with employing large language models, such as GPT [26], to process the text, identify entities and relationships, so to eventually output pertinent RDF triples.

For a more distant future, we foresee four ways of enhancing the ontology:

- **Broadening.** There are many more notions and related to them phenomena of complexity theory that we are not addressing here. From diverse models of computation (such as finite automata, Boolean circuits, probabilistic and quantum machines) to different approaches to define complexity classes (such as descriptive complexity) to connections with algebra and number theory: there is a lot of human knowledge on the subject to systematise in a machine-readable way.
- **Deepening.** As we have mentioned, our current approach focuses on the 'core knowledge' of complexity theory, i.e. the knowledge one usually acquires in an introductory undergraduate course. However, many more subtle details are swept under the carpet with this approach. For example, we work only with "large" classes like P and PSpace, and thus do not mention important facts such as $\text{P} \neq \text{DSpace}(n)$. Indeed, P and PSpace, are

themselves hierarchies of classes $\text{DTIME}(n^k)$ and $\text{DSPACE}(n^k)$. The expressiveness of FOWL is enough to add these and other details.

- **Increasing complexity.** Currently, we were only able to provide a partial solution to Challenge 4. Apart from the polynomial hierarchy, defining 'limits' of chains is important for other aspects of complexity theory. For example, our chain of bounds (Const, Log, Poly, Exp, 2Exp, 3Exp, ...) has, in fact, a limit, the tower function $t(n)$ defined by $t(0) = 1$ and $t(n+1) = 2^{t(n)}$. Functions that require time $t(n)$ to be computed are called *non-elementary* [27]. But this is not the end: there are Ackermann-hard computational problems [28] that have time complexity higher than the tower function, and further on [29, 30]. Representing and axiomatising these phenomena will, we believe, require more expressive formalisms, such as second-order logic.
- **Adding metadata.** In addition to representing the complexity theoretic results, it will be useful to provide meta-information, such as references to papers where a certain fact was (first) demonstrated, the proof techniques used, etc.

A separate research direction is providing a strong ontological and knowledge-engineering basis for our ontology. This would include defining basic notions of computer science, such as algorithms, models of computation, complexity measures and their relationships, in one or several mid-level ontologies, and further connecting them to a top-level ontology.

References

[1] A. Church, A Note on the Entscheidungsproblem, J. Symb. Log. 1 (1936) 40–41. doi:10.2307/2269326.

[2] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proc. London Math. Soc. s2-42 (1937) 230–265. doi:10.1112/PLMS/S2-42.1.230.

[3] J. Hartmanis, R. Stearns, On the computational complexity of algorithms, Transactions of The American Mathematical Society 117 (1965) 285–306. doi:10.2307/1994208.

[4] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: The next step for OWL, Journal of Web Semantics 6 (2008) 309–322. doi:10.1016/j.websem.2008.05.001, Semantic Web Challenge 2006/2007.

[5] S. Flügel, M. Glauer, F. Neuhaus, J. Hastings, When one logic is not enough: Integrating first-order annotations in OWL ontologies, Semantic Web Journal Preprint (2024) 1–16. doi:10.3233/SW-243440.

[6] C. Lange, Ontologies and languages for representing mathematical knowledge on the semantic web, Semantic Web 4 (2013) 119–158. doi:10.3233/SW-2012-0059.

[7] A. M. Elizarov, A. V. Kirillovich, E. K. Lipachev, O. A. Nevzorova, OntoMathPRO: An Ontology of Mathematical Knowledge, Doklady Mathematics 106 (2022) 429–435. URL: https://doi.org/10.1134/S1064562422700016. doi:10.1134/S1064562422700016.

[8] H. Putnam, What is mathematical truth?, Historia Mathematica 2 (1975) 529–533. doi:10.1016/0315-0860(75)90116-0.

[9] J. Landgrebe, B. Smith, Ontologies of common sense, physics and mathematics, arXiv preprint (2023). doi:10.48550/arXiv.2305.01560.

[10] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2006. doi:10.1017/CBO9780511804090.
[11] S. A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, Association for Computing Machinery, New York, NY, USA, 1971, p. 151–158. doi:10.1145/800157.805047.
[12] R. M. Karp, Reducibility among Combinatorial Problems, Springer US, Boston, MA, 1972, pp. 85–103. doi:10.1007/978-1-4684-2001-2_9.
[13] A. K. Chandra, L. J. Stockmeyer, Alternation, in: 17th Annual Symposium on Foundations of Computer Science (sfcs 1976), 1976, pp. 98–108. doi:10.1109/SFCS.1976.4.
[14] L. J. Stockmeyer, The polynomial-time hierarchy, Theoretical Computer Science 3 (1976) 1–22. doi:https://doi.org/10.1016/0304-3975(76)90061-X.
[15] L. Libkin, Elements of Finite Model Theory, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2004. doi:10.1007/978-3-662-07003-1.
[16] T. Baker, J. Gill, R. Solovay, Relativizations of the P = ?NP question, SIAM Journal on Computing 4 (1975) 431–442. doi:10.1137/0204037.
[17] V. Arvind, J. Köbler, U. Schöning, R. Schuler, If NP has polynomial-size circuits, then MA = AM, Theoretical Computer Science 137 (1995) 279–282. doi:https://doi.org/10.1016/0304-3975(95)91133-B.
[18] R. E. Ladner, On the structure of polynomial time reducibility, J. ACM 22 (1975) 155–171. doi:10.1145/321864.321877.
[19] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017. doi:10.1017/9781139025355.
[20] D. Hernández, A. Hogan, M. Krötzsch, Reifying RDF: What Works Well With Wikidata?, in: T. Liebig, A. Fokoue (Eds.), Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems, volume 1457 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 32–47.
[21] D. Brickley, R. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, 2004. URL: https://www.w3.org/2001/sw/RDFCore/Schema/200212bwm/.
[22] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible SROIQ, in: P. Doherty, J. Mylopoulos, C. A. Welty (Eds.), Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006, AAAI Press, 2006, pp. 57–67. URL: http://www.aaai.org/Library/KR/2006/kr06-009.php.
[23] S. Rudolph, Foundations of Description Logics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 76–136. doi:10.1007/978-3-642-23032-5_2.
[24] M. Schneider, S. Rudolph, G. Sutcliffe, Modeling in OWL 2 without restrictions, in: M. Rodriguez-Muro, S. Jupp, K. Srinivas (Eds.), Proceedings of the 10th International Workshop on OWL: Experiences and Directions (OWLED 2013) co-located with 10th Extended Semantic Web Conference (ESWC 2013), Montpellier, France, May 26-27, 2013, volume 1080 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2013. URL: https://ceur-ws.org/Vol-1080.
[25] A. Riazanov, A. Voronkov, The design and implementation of VAMPIRE, AI Commun. 15 (2002) 91–110. URL: https://dl.acm.org/doi/10.5555/1218615.1218620.
[26] OpenAI, GPT-4 Technical Report, 2024. arXiv:2303.08774.

[27] L. J. Stockmeyer, The complexity of decision problems in automata theory and logic, Ph.D. thesis, Massachusetts Institute of Technology, 1974. URL: https://api.semanticscholar.org/CorpusID:62162426.

[28] W. Czerwinski, L. Orlikowski, Reachability in vector addition systems is ackermann-complete, in: 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS), 2022, pp. 1229–1240. doi:10.1109/FOCS52979.2021.00120.

[29] H. Rogers, Theory of recursive functions and effective computability, MIT Press, Cambridge, MA, USA, 1987. URL: https://mitpress.mit.edu/9780262680523/theory-of-recursive-functions-and-effective-computability/.

[30] P. G. Hinman, Classical recursion theory. the theory of functions and sets of natural numbers., Journal of Symbolic Logic 55 (1990). doi:10.2307/2274492.